

Content Server Enterprise Edition

Version: 5.5

CSEE Developer's Tag Reference

Document Revision Date: Oct. 28, 2003



FATWIRE, INC. PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. In no event shall FatWire be liable for any loss of profits, loss of business, loss of use of data, interruption of business, or for indirect, special, incidental, or consequential damages of any kind, even if FatWire has been advised of the possibility of such damages arising from this publication. FatWire may revise this publication from time to time without notice. Some states or jurisdictions do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

Copyright © 2003 FatWire, inc. All rights reserved.

This product may be covered under one or more of the following U.S. patents: 4477698, 4540855, 4720853, 4742538, 4742539, 4782510, 4797911, 4894857, 5070525, RE36416, 5309505, 5511112, 5581602, 5594791, 5675637, 5708780, 5715314, 5724424, 5812776, 5828731, 5909492, 5924090, 5963635, 6012071, 6049785, 6055522, 6118763, 6195649, 6199051, 6205437, 6212634, 6279112 and 6314089. Additional patents pending.

FatWire, Content Server, Content Server Bridge Enterprise, Content Server Bridge XML, Content Server COM Interfaces, Content Server Desktop, Content Server Direct, Content Server Direct Advantage, Content Server DocLink, Content Server Engage, Content Server InSite Editor, Content Server Satellite, and Transact are trademarks or registered trademarks of FatWire, inc. in the United States and other countries.

iPlanet, Java, J2EE, Solaris, Sun, and other Sun products referenced herein are trademarks or registered trademarks of Sun Microsystems, Inc. *AIX, IBM, WebSphere*, and other IBM products referenced herein are trademarks or registered trademarks of IBM Corporation. *WebLogic* is a registered trademark of BEA Systems, Inc. *Microsoft, Windows* and other Microsoft products referenced herein are trademarks or registered trademarks of Microsoft Corporation. *UNIX* is a registered trademark of The Open Group. Any other trademarks and product names used herein may be the trademarks of their respective owners.

This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>) and software developed by Sun Microsystems, Inc. This product contains encryption technology from Phaos Technology Corporation.

You may not download or otherwise export or reexport this Program, its Documentation, or any underlying information or technology except in full compliance with all United States and other applicable laws and regulations, including without limitations the United States Export Administration Act, the Trading with the Enemy Act, the International Emergency Economic Powers Act and any regulations thereunder. Any transfer of technical data outside the United States by any means, including the Internet, is an export control requirement under U.S. law. In particular, but without limitation, none of the Program, its Documentation, or underlying information of technology may be downloaded or otherwise exported or reexported (i) into (or to a national or resident, wherever located, of) Cuba, Libya, North Korea, Iran, Iraq, Sudan, Syria, or any other country to which the U.S. prohibits exports of goods or technical data; or (ii) to anyone on the U.S. Treasury Department's Specially Designated Nationals List or the Table of Denial Orders issued by the Department of Commerce. By downloading or using the Program or its Documentation, you are agreeing to the foregoing and you are representing and warranting that you are not located in, under the control of, or a national or resident of any such country or on any such list or table. In addition, if the Program or Documentation is identified as Domestic Only or Not-for-Export (for example, on the box, media, in the installation process, during the download process, or in the Documentation), then except for export to Canada for use in Canada by Canadian citizens, the Program, Documentation, and any underlying information or technology may not be exported outside the United States or to any foreign entity or "foreign person" as defined by U.S. Government regulations, including without limitation, anyone who is not a citizen, national, or lawful permanent resident of the United States. By using this Program and Documentation, you are agreeing to the foregoing and you are representing and warranting that you are not a "foreign person" or under the control of a "foreign person."

CSEE Developer's Tag Reference

Document Revision Date: Oct. 28, 2003

Product Version: 5.5

Technical Support

<http://fatwire.com/support>

FatWire Software

Mineola, NY

United States of America

Table of

Contents

Summary of Changes	39
New in Version 5.0	40
Chapters	40
XML Tags	40
JSP Tags	41
Modified in Version 5.0	42
XML Tags	42
JSP Tags	43
Documented in Version 5.0	44
XML Tags	44
JSP Tags	46

Section 1: XML Tags

1 Content Server XML Tags	51
XML Tags by Function	52
Variables	52
Lists	52
Counters	52
Sessions and Cookies	53
Database Access	53
Directory Services	53
Execution	53
Files and URLs	54
Events and Email	54
Search Engines	54
Revision Tracking	54

Conditions.	55
Built-in Functions.	55
Miscellaneous Tags	56
ACL Tags	56
Alphabetical List of XML Tags	57
ACL.ARGUMENT	57
ACL.CREATE	58
ACL.DELETE	59
ACL.GATHER.	60
ACL.GET	61
ACL.LIST.	62
ACL.LOAD	63
ACL.SAVE.	65
ACL.SCATTER.	66
ACL.SET	67
APPEVENT	68
ARGUMENT	69
BEGINS	70
CALLAPPLAGIC (Deprecated)	71
CALLELEMENT.	71
CALLJAVA	73
CALLSQL	74
CATALOGDEF	76
CATALOGMANAGER.	78
addrow	80
addrows	83
addrowtext	85
addUser (Deprecated)	87
commit	88
createtable	89
deletelog.	92
deleterevision.	93
deleterow	95
deleterows	97
deletetable	99
delUser (Deprecated).	100
editrow (Deprecated)	101
editrows (Deprecated)	103
exportform	105
exportlog	106
flushcatalog	107
flushpage	108
history	109
lock.	111
login	112

logout	113
mirrorabort	114
mirrorrows	116
modifyUser (Deprecated).	119
release	120
replacerow	121
replaceroes	124
rollback	127
selectrows	128
setversions	130
tracktable	131
untracktable	132
unlockrecord	133
updaterow	134
updaterow2	137
updaterows	139
updaterows2	141
COMMIT	143
CONTENTSERVER (Deprecated)	145
COPYLIST	147
CSVAR	149
DEBABBLE	150
DELETEFILE	151
DELETEREVISION	152
DESTROYEVENT	154
DIR.ADDATTRS	154
DIR.ADDGROUPMEMBER	156
DIR.CHILDREN	158
DIR.CREATE	159
DIR.DELETE	161
DIR.GETATTRS	162
DIR.GROUPMEMBERS	164
DIR.GROUPMEMBERSHIPS	165
DIR.LISTUSERS	167
DIR.REMOVEATTRS	168
DIR.REMOVEGROUPMEMBER	170
DIR.REPLACEATTRS	172
DIR.SEARCH	173
DISABLEEVENT	175
DISABLECACHE	176
EMAILEVENT	176
ENABLEEVENT	178
ENDS	178
EVALSERVER (Deprecated)	179
EXECSQL	181

EXPORT.EXPORT	182
FILELIST	184
FLUSH	186
GOTOROW	187
HISTORY	188
ICS.ARGUMENT	191
ICS.CDATA	192
ICS.CLEARERRNO	193
ICS.ENCODE	193
ICS.GETERRNO	194
ICS.GETPROPERTY	195
ICS.GETVAR	196
ICS.LISTGET	197
ICS.LITERAL	198
ICS.QUERYEVENTS	199
ICS.READEVENT	201
ICS.RESOLVEVARIABLES	202
ICS.SQL	203
ICS.SQLEXP	204
ICS.STREAMHEADER	205
IF	206
INCCOUNTER	207
INDEXADD	208
INDEXCREATE	211
INDEXDESTROY	213
INDEXEXISTS	214
INDEXOF	216
INDEXREMOVE	217
INDEXREPLACE	219
INSERT	221
IsCatalog	222
IsColumn	223
IsCounter	223
IsElement	224
IsError	224
ISINLIST	225
IsList	227
IsObject	227
IsSessionVariable	228
IsTracked	228
IsTree	229
IsVariable	229
LOADPROPERTY	229
LOCALESTRING.ARGUMENT	231
LOCALESTRING.GETSTRING	232

LOCK	233
LOGMSG	234
LOOP	235
mail.send.	237
name.get	238
name.makechild	238
OBJECT.ARGUMENT	239
OBJECT.CHECKIN	240
OBJECT.CHECKOUT	241
OBJECT.CHILDREN	243
OBJECT.CREATE	244
OBJECT.DEINSTALL	246
OBJECT.DELETE	247
OBJECT.GATHER	248
OBJECT.GET	249
OBJECT.INSTALL	250
OBJECT.ISEMPY	251
OBJECT.LIST	252
OBJECT.LOAD	254
OBJECT.ROLLBACK	256
OBJECT.ROOT	258
OBJECT.SAVE	259
OBJECT.SCATTER	260
OBJECT.SET	261
OBJECT.SETOBJECT	262
OBJECT.SETPARENT	263
OBJECT.STREAM	264
OBJECT.TRACK	265
OBJECT.UNDOCHECKOUT	266
OBJECT.UNTRACK	267
portal.logdep	268
PORTAL.RANDOMVAL	269
POST	270
RELEASE	271
REMOVECOOKIE	273
REMOVECOUNTER	274
REMOVESSVAR	274
REMOVEVAR	275
RENAMELIST	276
REPLACE	277
REPLACEALL	278
RESTOREPROPERTY	279
RETRIEVEREVISION	280
ROLLBACK	281
SEARCH	282

SEARCHDATENATIVE.....	285
SELECTTO	286
SENDMAIL (Deprecated)	288
SENDMAIL (Variant 1).....	289
SENDMAIL (Variant 2).....	290
SETCOOKIE	291
SETCOUNTER	293
SETDATE	294
SETROW	295
SETSSVAR	296
SETVAR.....	297
SETVERSIONS	298
SQLXP.....	299
STRINGLIST	301
SUBSTITUTE	303
SUBSTRING	305
SYSEXEC	306
THROWEXCEPTION.....	307
TIME.GET	308
TIME.SET	309
TOLOWER.....	309
TOUPPER	310
TRACKTABLE	310
TREEMANAGER	312
addchild	314
addchildren.....	316
copychild	319
createtree	320
delchild.....	321
delchildren	322
deletetree	324
findnode	325
getchildren	326
getnode.....	329
getparent.....	330
listtrees.....	332
movechild.....	334
nodepath.....	335
setobject	337
validatenode.....	338
verifypath.....	340
UNLOCKRECORD.....	341
UNTRACKTABLE	342
USERISMEMBER.....	343
USER.ADD (Deprecated)	344

USER.CHANGEPASSWORD (Deprecated).....	345
USER.DELETE (Deprecated)	346
USER.LOGIN	346
USER.LOGOUT	347
USER.MODIFY (Deprecated).....	348
USER.SU	349
WRITEFILE.....	350
XMLFILTER	351
Variables and Built-in Functions	353
CS.AMPM	353
CS.Browser.....	353
CS.CacheFile	354
CS.CatalogDir	355
CS.Date.....	355
CS.DDate	356
CS.Day	357
CS.Elementname	357
CS.Empty	358
CS.Header.....	358
CS.Hour	359
CS.Hour24	359
CS.HTTPS	360
CS.IOError	360
CS.IsExporting	361
CS.Min	361
CS.Mon.....	362
CS.PATH_INFO	363
CS.Property	363
CS.QUERY_STRING	364
CS.REMOTE_ADDR	365
CS.REMOTE_HOST.....	365
CS.REQUEST_METHOD.....	366
CS.SERVER_NAME.....	366
CS.SERVER_PORT.....	367
CS.SERVER_PROTOCOL	367
CS.SQLDate.....	368
CS.SystemAssetsRoot	368
CS.UniqueID	369
CS.Year	369
2 CS-Direct XML Tags.....	371
Asset Tags.....	373
ASSET.ADDCHILD	374
ASSET.CANEDIT.....	375
ASSET.CHILDGATHER	376

ASSET.CHILDREN.....	377
ASSET.CHILDTYPES	379
ASSET.CREATE	380
ASSET.DEF	381
ASSET.EXPORT	382
ASSET.GATHER.....	384
ASSET.GET	386
ASSET.GETASSETTYPE.....	387
ASSET.GETLEGALSUBTYPES	387
ASSET.GETPUBDEPS	388
ASSET.GETSITENODE	390
ASSET.GETSUBTYPE	391
ASSET.GETTEMPLATEROOTELEMENT.....	391
ASSET.GETTEMPLATEFORTARGET.....	392
ASSET.IMPORT	394
ASSET.INSPECT.....	397
ASSET.LIST	398
ASSET.LOAD	399
ASSET.LOADALL	401
ASSET.LOADREVISION.....	403
ASSET.REFERENCEDBY	404
ASSET.REMOVECHILDREN	405
ASSET.SAVE	406
ASSET.SAVEALL.....	407
ASSET.SCATTER.....	409
ASSET.SEARCH.....	410
ASSET.SET	412
ASSET.SHARE	413
ASSET.SITEPARENT.....	414
ASSET.SITES	415
ASSET.STREAM.....	416
ASSET.VOID.....	417
CCUser Tags.....	418
CCUSER.CLEARROLES	418
CCUSER.GETEMAIL.....	418
CCUSER.GETID	419
CCUSER.GETNAME	420
CCUSER.GETSITEROLES.....	420
CCUSER.GETSITES.....	421
CCUSER.SETEMAIL	421
CCUSER.SETNAME.....	422
CCUSER.SETSITEROLES	423
Date Format Tags	424
DATEFORMAT.CREATE	424
DATEFORMAT.GETDATE	425

DATEFORMAT.GETDATETIME	426
DATEFORMAT.GETTIME	427
Email Tags	428
EMAIL.GATHER	428
EMAIL.GETBODY	429
EMAIL.GETDESCRIPTION	430
EMAIL.GETID	431
EMAIL.GETNAME	432
EMAIL.GETSUBJECT	433
EMAIL.SCATTER	434
EMAIL.SETBODY	435
EMAIL.SETDESCRIPTION	436
EMAIL.SETNAME	437
EMAIL.SETSUBJECT	438
EMAIL.TRANSLATEBODY	439
EMAIL.TRANSLATESUBJECT	440
Email Manager Tags	441
EMAILMANAGER.CREATE	441
EMAILMANAGER.DELETE	441
EMAILMANAGER.GETALL	442
EMAILMANAGER.LOAD	443
EMAILMANAGER.SAVE	444
InSite Editor Tag	446
INSITE.EDIT	446
Participants Tags	448
PARTICIPANTS.ADDPARTICIPANT	448
PARTICIPANTS.CLEAR	448
PARTICIPANTS.GATHER	449
PARTICIPANTS.GETALLPARTICIPANTS	450
PARTICIPANTS.GETALLROLES	450
PARTICIPANTS.GETPARTICIPANTSFORROLE	451
PARTICIPANTS.REMOVEPARTICIPANT	452
PARTICIPANTS.SCATTER	452
PARTICIPANTS.SETPARTICIPANTSFORROLE	453
Property Tag	454
PROPERTY.GET	454
Render Tags	455
RENDER.CALLELEMENT	456
RENDER.CONTENTSERVER	457
RENDER.DELETECACHE (Deprecated)	459
RENDER.ENCODE	459
RENDER.EXPORT	461
RENDER.FILTER	462
RENDER.GETBLOBURL	463
RENDER.GETPAGEURL	466

RENDER.LOGDEP	468
RENDER.PACKARGS	469
RENDER.SATELLITEBLOB	471
RENDER.SATELLITEPAGE	473
RENDER.STREAM	475
RENDER.UNKNOWNDEPS	476
Role List Tags	477
ROLELIST.ADD	477
ROLELIST.CLEAR	478
ROLELIST.DELETE	479
ROLELIST.GATHER	480
ROLELIST.GETALL	480
ROLELIST.HASROLE	481
ROLELIST.ISALL	482
ROLELIST.SCATTER	483
ROLELIST.SETALL	483
Site List Tags	485
SITELIST.ADD	485
SITELIST.CLEAR	485
SITELIST.DELETE	485
SITELIST.GATHER	486
SITELIST.GETALL	486
SITELIST.HASSITE	487
SITELIST.ISALL	487
SITELIST.SCATTER	488
SITELIST.SETALL	488
Site Plan Tags	489
SITEPLAN.CHILDREN	489
SITEPLAN.GET	491
SITEPLAN.GETPAGENAME	492
SITEPLAN.LISTPAGES	494
SITEPLAN.LOAD	495
SITEPLAN.NODEPATH	497
SITEPLAN.PLACE	498
SITEPLAN.ROOT	499
SITEPLAN.SETOBJECT	500
SITEPLAN.SETPARENT	500
SITEPLAN.UNPLACE	501
SITEPLAN.UNPLACEDANCESTOR	502
Type List Tags	503
TYPELIST.ADDASSETTYPE	503
TYPELIST.CLEARASSETTYPES	503
TYPELIST.DELETEASSETTYPE	503
TYPELIST.GATHER	504
TYPELIST.GETALLASSETTYPES	504

TYPELIST.HASTYPE.....	504
TYPELIST.ISALL.....	505
TYPELIST.SCATTER.....	505
TYPELIST.SETALL.....	506
User List Tags.....	507
USERLIST.ADD.....	507
USERLIST.CLEAR.....	507
USERLIST.DELETE.....	507
USERLIST.GATHER.....	508
USERLIST.GETALL.....	508
USERLIST.SCATTER.....	508
User Manager Tags.....	510
USERMANAGER.DELETESITE.....	510
USERMANAGER.DELETEUSER.....	510
USERMANAGER.GETLOGINUSER.....	511
USERMANAGER.GETLOGINUSERNAME.....	511
USERMANAGER.GETUSER.....	512
USERMANAGER.GETUSERFROMNAME.....	512
USERMANAGER.GETUSERNAME.....	513
USERMANAGER.GETUSERS.....	514
USERMANAGER.LOGINUSER.....	515
USERMANAGER.LOGOUT.....	515
USERMANAGER.SETUSER.....	516
Variables.....	517
c.....	517
cid.....	518
ct.....	518
eid.....	519
p.....	520
rendermode.....	520
tid.....	521
Workflowable Object Tags.....	522
WORKFLOWABLEOBJECT.GETDISPLAYABLENAME.....	522
WORKFLOWABLEOBJECT.GETOBJECTNAME.....	522
Workflow Action Tags.....	524
WORKFLOWACTION.GATHER.....	524
WORKFLOWACTION.GETARGUMENTS.....	525
WORKFLOWACTION.GETDESCRIPTION.....	525
WORKFLOWACTION.GETELEMENTNAME.....	525
WORKFLOWACTION.GETID.....	526
WORKFLOWACTION.GETNAME.....	526
WORKFLOWACTION.SCATTER.....	526
WORKFLOWACTION.SETARGUMENTS.....	527
WORKFLOWACTION.SETDESCRIPTION.....	527
WORKFLOWACTION.SETELEMENTNAME.....	528

WORKFLOWACTION.SETNAME	528
Workflow Action List Tags	529
WORKFLOWACTIONLIST.ADDACTION	529
WORKFLOWACTIONLIST.CLEARACTIONS	529
WORKFLOWACTIONLIST.GATHER	529
WORKFLOWACTIONLIST.GETALLACTIONS	530
WORKFLOWACTIONLIST.REMOVEACTION	530
WORKFLOWACTIONLIST.SCATTER	531
Workflow Asset Tags	532
WORKFLOWASSET.GETASSETID	532
WORKFLOWASSET.GETASSETTYPE	532
Workflow Assignment Tags	534
WORKFLOWASSIGNMENT.GETASSIGNDATE	534
WORKFLOWASSIGNMENT.GETASSIGNEDOBJECT	534
WORKFLOWASSIGNMENT.GETASSIGNEDUSERID	535
WORKFLOWASSIGNMENT.GETASSIGNERUSERID	536
WORKFLOWASSIGNMENT.GETASSIGNMENTCOMMENT	537
WORKFLOWASSIGNMENT.GETDEADLINE	537
WORKFLOWASSIGNMENT.GETDISPOSITION	538
WORKFLOWASSIGNMENT.GETDISPOSITIONCOMMENT	539
WORKFLOWASSIGNMENT.GETDISPOSITIONDATE	539
WORKFLOWASSIGNMENT.GETDISPOSITIONUSERID	540
WORKFLOWASSIGNMENT.GETGROUPEDCOMMENT	541
WORKFLOWASSIGNMENT.GETGROUPEDDEADLINE	541
WORKFLOWASSIGNMENT.GETGROUPID	542
WORKFLOWASSIGNMENT.GETGROUPEDWORKFLOWEDSTEPID	543
WORKFLOWASSIGNMENT.GETID	543
WORKFLOWASSIGNMENT.GETQUEUEDWORKFLOWSTEPID	544
WORKFLOWASSIGNMENT.GETSTATEID	544
WORKFLOWASSIGNMENT.GETSTATUS	545
WORKFLOWASSIGNMENT.SCATTER	546
Workflow Engine Tags	548
WORKFLOWENGINE.ABSTAINFROMVOTING	548
WORKFLOWENGINE.ADDNEWOBJECTTOGROUP	549
WORKFLOWENGINE.ADDOBJECTTOGROUP	550
WORKFLOWENGINE.CANCELASSIGNMENT	551
WORKFLOWENGINE.CANDELETEGROUP	551
WORKFLOWENGINE.CANEDITGROUP	552
WORKFLOWENGINE.CANSTARTWORKFLOW	552
WORKFLOWENGINE.CLOSEASSIGNMENTS	553
WORKFLOWENGINE.COMPLETEASSIGNMENT	554
WORKFLOWENGINE.CREATENEWASSIGNMENTACTION	555
WORKFLOWENGINE.CREATENEWCONDITION	555
WORKFLOWENGINE.CREATENEWDEADLOCKACTION	556
WORKFLOWENGINE.CREATENEWDELEGATEACTION	556

WORKFLOWENGINE.CREATENEWFUNCTION	557
WORKFLOWENGINE.CREATENEWGROUP	557
WORKFLOWENGINE.CREATENEWGROUPACTION	557
WORKFLOWENGINE.CREATENEWPROCESS.....	558
WORKFLOWENGINE.CREATENEWSTATE	558
WORKFLOWENGINE.CREATENEWSTEPACTION	559
WORKFLOWENGINE.DELEGATEASSIGNMENT	559
WORKFLOWENGINE.DELETEASSIGNMENTACTIONID	560
WORKFLOWENGINE.DELETECONDITIONID.....	560
WORKFLOWENGINE.DELETEDEADLOCKACTIONID	560
WORKFLOWENGINE.DELETEDELEGATEACTIONID	561
WORKFLOWENGINE.DELETEFUNCTIONID.....	561
WORKFLOWENGINE.DELETEGROUPACTIONID.....	561
WORKFLOWENGINE.DELETEGROUPID	561
WORKFLOWENGINE.DELETEPROCESSID	562
WORKFLOWENGINE.DELETESTATEID.....	562
WORKFLOWENGINE.DELETESTEPACTIONID.....	562
WORKFLOWENGINE.EXECUTETIMEDACTIONS	563
WORKFLOWENGINE.GETADDOBJECTTOGROUPCANDIDATEASSIGNE ES	563
WORKFLOWENGINE.GETALLFUNCTIONS.....	564
WORKFLOWENGINE.GETALLPROCESSES.....	564
WORKFLOWENGINE.GETALLSTATES	565
WORKFLOWENGINE.GETALLSTATESFORPROCESS	565
WORKFLOWENGINE.GETASSIGNMENTACTIONID	566
WORKFLOWENGINE.GETASSIGNMENTACTIONNAME	566
WORKFLOWENGINE.GETASSIGNMENTID.....	566
WORKFLOWENGINE.GETAVAILABLEASSIGNMENTACTIONS.....	567
WORKFLOWENGINE.GETAVAILABLECONDITIONS	567
WORKFLOWENGINE.GETAVAILABLEDEADLOCKACTIONS.....	567
WORKFLOWENGINE.GETAVAILABLEDELEGATEACTIONS	568
WORKFLOWENGINE.GETAVAILABLEGROUPACTIONS.....	568
WORKFLOWENGINE.GETAVAILABLEGROUPS	568
WORKFLOWENGINE.GETAVAILABLESTEPACTIONS	569
WORKFLOWENGINE.GETBLOCKINGREASON	569
WORKFLOWENGINE.GETCOMPLETEASSIGNMENTCANDIDATEASSIG NEES	570
WORKFLOWENGINE.GETCONDITIONID	571
WORKFLOWENGINE.GETCONDITIONNAME.....	571
WORKFLOWENGINE.GETDEADLOCKACTIONID	572
WORKFLOWENGINE.GETDEADLOCKACTIONNAME	572
WORKFLOWENGINE.GETDELEGATEACTIONID.....	572
WORKFLOWENGINE.GETDELEGATEACTIONNAME	573
WORKFLOWENGINE.GETDELEGATEECANDIDATES	573
WORKFLOWENGINE.GETFILTEREDASSIGNMENTS	574

WORKFLOWENGINE.GETFUNCTIONID	576
WORKFLOWENGINE.GETFUNCTIONNAME.....	577
WORKFLOWENGINE.GETGROUPACTIONID	577
WORKFLOWENGINE.GETGROUPACTIONNAME	577
WORKFLOWENGINE.GETGROUPID.....	578
WORKFLOWENGINE.GETGROUPMEMBERS	578
WORKFLOWENGINE.GETGROUPMEMBERSHIP	579
WORKFLOWENGINE.GETGROUPNAME	579
WORKFLOWENGINE.GETLEGALNEXTWORKFLOWSTEPS	579
WORKFLOWENGINE.GETMULTISITEPROCESSES	580
WORKFLOWENGINE.GETOBJECTHISTORY.....	581
WORKFLOWENGINE.GETOBJECTPARTICIPANTS	581
WORKFLOWENGINE.GETOBJECTSTATEID	582
WORKFLOWENGINE.GETOBJECTWORKFLOWPROCESS.....	582
WORKFLOWENGINE.GETPOTENTIALPARTICIPANTS.....	582
WORKFLOWENGINE.GETPROCESSID	583
WORKFLOWENGINE.GETPROCESSNAME	583
WORKFLOWENGINE.GETSTARTWORKFLOWCANDIDATEASSIGNEES .	584
WORKFLOWENGINE.GETSTATEID	584
WORKFLOWENGINE.GETSTATENAME.....	585
WORKFLOWENGINE.GETSTEPACTIONID	585
WORKFLOWENGINE.GETSTEPACTIONNAME.....	585
WORKFLOWENGINE.GETSTEPID	586
WORKFLOWENGINE.ISFUNCTIONLEGAL	586
WORKFLOWENGINE.ISOBJECTASSIGNED.....	587
WORKFLOWENGINE.ISOBJECTASSIGNEDTOGROUP	587
WORKFLOWENGINE.ISOBJECTASSIGNEDTOUSER.....	587
WORKFLOWENGINE.ISOBJECTDEADLOCKED.....	588
WORKFLOWENGINE.REMOVEOBJECTFROMGROUP	588
WORKFLOWENGINE.SETASSIGNMENTACTION.....	589
WORKFLOWENGINE.SETASSIGNMENTDEADLINE	589
WORKFLOWENGINE.SETCONDITION	589
WORKFLOWENGINE.SETDEADLOCKACTION	590
WORKFLOWENGINE.SETDELEGATEACTION	590
WORKFLOWENGINE.SETFUNCTION	590
WORKFLOWENGINE.SETGROUP	591
WORKFLOWENGINE.SETGROUPACTION.....	591
WORKFLOWENGINE.SETOBJECTPARTICIPANTS.....	591
WORKFLOWENGINE.SETPROCESS	592
WORKFLOWENGINE.SETSTATE.....	592
WORKFLOWENGINE.SETSTEPACTION.....	592
WORKFLOWENGINE.STARTOBJECTWORKFLOWPROCESS	593
Workflow Function Tags	595
WORKFLOWFUNCTION.GATHER.....	595

WORKFLOWFUNCTION.GETDESCRIPTION	596
WORKFLOWFUNCTION.GETID	596
WORKFLOWFUNCTION.GETNAME	596
WORKFLOWFUNCTION.SCATTER	597
WORKFLOWFUNCTION.SETDESCRIPTION	597
WORKFLOWFUNCTION.SETNAME	598
Workflow Group Tags	599
WORKFLOWGROUP.GATHER	599
WORKFLOWGROUP.GETDELETEROLES	600
WORKFLOWGROUP.GETDESCRIPTION	601
WORKFLOWGROUP.GETEDITROLES	601
WORKFLOWGROUP.GETGROUPACTIONS	602
WORKFLOWGROUP.GETID	603
WORKFLOWGROUP.GETPARTICIPANTS	603
WORKFLOWGROUP.GETNAME	604
WORKFLOWGROUP.GETSITES	605
WORKFLOWGROUP.GETWORKFLOWPROCESSID	605
WORKFLOWGROUP.SCATTER	606
WORKFLOWGROUP.SETDELETEROLES	607
WORKFLOWGROUP.SETDESCRIPTION	608
WORKFLOWGROUP.SETEDITROLES	608
WORKFLOWGROUP.SETGROUPACTIONS	609
WORKFLOWGROUP.SETNAME	610
WORKFLOWGROUP.SETPARTICIPANTS	610
WORKFLOWGROUP.SETSITES	611
WORKFLOWGROUP.SETWORKFLOWPROCESSID	612
Workflow Priv Tags	613
WORKFLOWPRIV.GATHER	613
WORKFLOWPRIV.GETALLOWED	614
WORKFLOWPRIV.GETFUNCTIONID	614
WORKFLOWPRIV.GETROLE	614
WORKFLOWPRIV.GETSTATEID	615
WORKFLOWPRIV.SCATTER	615
WORKFLOWPRIV.SETALLOWED	616
WORKFLOWPRIV.SETFUNCTIONID	616
WORKFLOWPRIV.SETROLE	616
WORKFLOWPRIV.SETSTATEID	617
Workflow Privs Tags	618
WORKFLOWPRIVS.ADD	618
WORKFLOWPRIVS.CLEAR	618
WORKFLOWPRIVS.CREATE	618
WORKFLOWPRIVS.GATHER	619
WORKFLOWPRIVS.GETALLPRIVS	619
WORKFLOWPRIVS.REMOVE	620
WORKFLOWPRIVS.SCATTER	620

Workflow Process Tags	622
WORKFLOWPROCESS.ADDSTEP	622
WORKFLOWPROCESS.CLEARSTEPS	623
WORKFLOWPROCESS.DELETESTEP	623
WORKFLOWPROCESS.GATHER	624
WORKFLOWPROCESS.GETADMINROLES	625
WORKFLOWPROCESS.GETDELEGATIONACTIONS	625
WORKFLOWPROCESS.GETFUNCTIONPRIVS	626
WORKFLOWPROCESS.GETID	626
WORKFLOWPROCESS.GETINITIALSTEPID	627
WORKFLOWPROCESS.GETPROCESSDESCRIPTION	627
WORKFLOWPROCESS.GETPROCESSNAME	628
WORKFLOWPROCESS.GETROLES	629
WORKFLOWPROCESS.GETSITES	629
WORKFLOWPROCESS.GETSOURCEROLES	630
WORKFLOWPROCESS.GETSTEP	630
WORKFLOWPROCESS.GETSTEPS	631
WORKFLOWPROCESS.GETTYPES	632
WORKFLOWPROCESS.SCATTER	632
WORKFLOWPROCESS.SETADMINROLES	633
WORKFLOWPROCESS.SETDELEGATIONACTIONS	634
WORKFLOWPROCESS.SETFUNCTIONPRIVS	634
WORKFLOWPROCESS.SETPROCESSDESCRIPTION	635
WORKFLOWPROCESS.SETPROCESSNAME	635
WORKFLOWPROCESS.SETROLES	636
WORKFLOWPROCESS.SETSITES	636
WORKFLOWPROCESS.SETSTEPS	637
WORKFLOWPROCESS.SETTYPES	637
Workflow State Tags	639
WORKFLOWSTATE.GATHER	639
WORKFLOWSTATE.GETACTIONS	640
WORKFLOWSTATE.GETESTIMATEDSTATETIME	640
WORKFLOWSTATE.GETID	640
WORKFLOWSTATE.GETSTATEDESCRIPTION	641
WORKFLOWSTATE.GETSTATENAME	641
WORKFLOWSTATE.SCATTER	641
WORKFLOWSTATE.SETACTIONS	642
WORKFLOWSTATE.SETESTIMATEDSTATETIME	642
WORKFLOWSTATE.SETSTATEDESCRIPTION	643
WORKFLOWSTATE.SETSTATENAME	643
Workflow Step Tags	644
WORKFLOWSTEP.GATHER	644
WORKFLOWSTEP.GETAUTHORIZATIONROLES	645
WORKFLOWSTEP.GETCOMPLETIONACTIONS	645
WORKFLOWSTEP.GETCONDITIONS	645

WORKFLOWSTEP.GETDEADLOCKACTIONS	646
WORKFLOWSTEP.GETGROUPLOCKFLAG	646
WORKFLOWSTEP.GETID	646
WORKFLOWSTEP.GETNAME	647
WORKFLOWSTEP.GETNOTIFYROLES	647
WORKFLOWSTEP.GETENDSTATE	647
WORKFLOWSTEP.GETOVERRIDEFLAG	648
WORKFLOWSTEP.GETSTARTSTATE	648
WORKFLOWSTEP.GETSTEPTYPE	649
WORKFLOWSTEP.SCATTER	649
WORKFLOWSTEP.SETAUTHORIZATIONROLES	650
WORKFLOWSTEP.SETCOMPLETIONACTIONS	650
WORKFLOWSTEP.SETCONDITIONS	651
WORKFLOWSTEP.SETDEADLOCKACTIONS	651
WORKFLOWSTEP.SETENDSTATE	651
WORKFLOWSTEP.SETGROUPLOCKFLAG	652
WORKFLOWSTEP.SETNAME	652
WORKFLOWSTEP.SETNOTIFYROLES	652
WORKFLOWSTEP.SETOVERRIDEFLAG	653
WORKFLOWSTEP.SETSTARTSTATE	653
WORKFLOWSTEP.SETSTEPTYPE	653
Workflow Timed Action Tags	654
WORKFLOWTIMEDACTION.GATHER	654
WORKFLOWTIMEDACTION.GETASSIGNMENTACTION	654
WORKFLOWTIMEDACTION.GETRELATIVETIME	655
WORKFLOWTIMEDACTION.SCATTER	655
WORKFLOWTIMEDACTION.SETASSIGNMENTACTION	656
WORKFLOWTIMEDACTION.SETRELATIVETIME	656
Workflow Timed Action List Tags	657
WORKFLOWTIMEDACTIONLIST.ADDTIMEDACTION	657
WORKFLOWTIMEDACTIONLIST.CLEARTIMEDACTIONS	657
WORKFLOWTIMEDACTIONLIST.CREATE	657
WORKFLOWTIMEDACTIONLIST.GATHER	658
WORKFLOWTIMEDACTIONLIST.GETALLTIMEDACTIONS	658
WORKFLOWTIMEDACTIONLIST.SCATTER	659
WORKFLOWTIMEDACTIONLIST.REMOVETIMEDACTION	659
3 CS-Direct Advantage XML Tags	661
Assetset Tags	662
ASSETSET.GETASSETCOUNT	662
ASSETSET.GETASSETLIST	663
ASSETSET.GETATTRIBUTEVALUECOUNT	664
ASSETSET.GETATTRIBUTEVALUES	665
ASSETSET.GETMULTIPLEVALUES	666
ASSETSET.SETASSET	667

ASSETSET.SETEMPTY	668
ASSETSET.SETLISTEDASSETS	668
ASSETSET.SETSEARCHEDASSETS	669
Blob Service Tags	671
BLOBSERVICE.GETIDCOLUMN	671
BLOBSERVICE.GETTABLENAME	671
BLOBSERVICE.GETURLCOLUMN	672
BLOBSERVICE.READDATA	672
Calculator Tags	673
CALCULATOR.GO	673
Cart Tags	674
CART.ADDITEM	675
CART.CHECK	676
CART.CLEAR	677
CART.CLEARDISCOUNTS	678
CART.COMMIT	678
CART.CREATE	679
CART.DELETEITEM	679
CART.GETCARTDISCOUNTS	680
CART.GETCARTDISCOUNTTOTAL	680
CART.GETERRORS	681
CART.GETFINALTOTAL	681
CART.FROMSTRING	682
CART.GETITEMDISCOUNTS	682
CART.GETITEMDISCOUNTTOTAL	682
CART.GETITEMERRORS	683
CART.GETITEMLEGALVALUES	683
CART.GETITEMPARAMETER	684
CART.GETITEMPARAMETERS	684
CART.GETITEMS	685
CART.GETITEMTOTAL	686
CART.GETLEGALVALUES	686
CART.GETPARAMETER	687
CART.GETPARAMETERS	688
CART.GETPRELIMINARYTOTAL	688
CART.GETSHIPPINGTOTAL	689
CART.GETSTORES	689
CART.GETTAXTOTAL	690
CART.GETTRANSACTIONID	690
CART.GETTRANSACTIONIDS	691
CART.MERGE	691
CART.SETITEMPARAMETERS	691
CART.SETITEMQUANTITY	692
CART.SETPARAMETER	692
CART.SETPARAMETERS	693

CART.TOSTRING.....	693
Cart Set Tags.....	694
CARTSET.ADDCART	694
CARTSET.CARTIDS	694
CARTSET.DELETECART	695
CARTSET.GETCART.....	695
CARTSET.GETSET	695
Commerce Context Tags.....	697
COMMERCECONTEXT.GETABANDONEDCARTSESSIONS	697
COMMERCECONTEXT.GETCURRENTCART	698
COMMERCECONTEXT.GETSESSIONCART.....	698
COMMERCECONTEXT.LOGOUT.....	699
COMMERCECONTEXT.SAVEALL	699
COMMERCECONTEXT.SETCURRENTCART.....	700
CSMAC Tags	701
CSMAC.COOKIEVALUE	701
CSMAC.PAYLOAD	702
CSMAC.STRING.....	703
CSMAC.TRANSACTIONSERVER.....	703
CSUSER Tags.....	704
CSUSER.AUTHENTICATE	704
CSUSER.CLEAR.....	705
CSUSER.CREATE	705
CSUSER.CREATEPMTACCT	705
CSUSER.DELETEPMTACCT	706
CSUSER.FROMSTRING	707
CSUSER.GETALLPMTACCTS	707
CSUSER.GETERRORS.....	708
CSUSER.GETLEGALPMTBRANDS.....	708
CSUSER.GETPARAMETER	709
CSUSER.GETPARAMETERS	709
CSUSER.LOOKUP	710
CSUSER.MODIFY	710
CSUSER.MODIFYPMTACCT.....	711
CSUSER.REGISTER.....	712
CSUSER.SETPARAMETER.....	713
CSUSER.SETPARAMETERS	713
CSUSER.TOSTRING	714
Currency Tags.....	715
CURRENCY.CREATE	715
CURRENCY.GETCURRENCY	716
CURRENCY.GETISOCODE	716
CURRENCY.GETSYMBOL.....	717
CURRENCY.READCURRENCY.....	717
CURRENCY.ROUND.....	717

CURRENCY.ROUNDDOWN.....	718
CURRENCY.ROUNDUP	718
Date Converter Tags	720
DATE.CLOCKLIST	720
DATE.CONVERT	721
DATE.DEFAULTTZ	722
Decimal Tags	723
DECIMAL.CREATE	723
DECIMAL.GETDECIMAL	723
DECIMAL.GETMAXPLACES	724
DECIMAL.GETMINPLACES	724
DECIMAL.GETTHOUSANDS	724
DECIMAL.READDECIMAL	725
DECIMAL.SETMAXPLACES	725
DECIMAL.SETMINPLACES	726
DECIMAL.SETTHOUSANDS	726
Hash Table Tags	727
HASH.ADD	727
HASH.CONTAINS	727
HASH.CREATE	728
HASH.HASDATA	728
HASH.TOSTRING	729
List Object Tags	730
LISTOBJECT.ADDROW	730
LISTOBJECT.CREATE	731
LISTOBJECT.TOLIST	731
Locale Tags	733
LOCALE.CREATE	733
Miscellaneous Tags	734
MISC.GETLISTCOLUMNNAME	734
Searchstate Tags	735
SEARCHSTATE.ADDLIKECONSTRAINT	735
SEARCHSTATE.ADDNESTEDCONSTRAINT	736
SEARCHSTATE.ADDRANGECONSTRAINT	737
SEARCHSTATE.ADDRICHTEXTCONSTRAINT	738
SEARCHSTATE.ADDSIMPLELIKECONSTRAINT	739
SEARCHSTATE.ADDSIMPLESTANDARDCONSTRAINT	740
SEARCHSTATE.ADDSTANDARDCONSTRAINT	741
SEARCHSTATE.CREATE	742
SEARCHSTATE.DELETECONSTRAINT	743
SEARCHSTATE.DUPLICATE	744
SEARCHSTATE.FINDCONSTRAINT	744
SEARCHSTATE.FROMSTRING	746
SEARCHSTATE.TOSTRING	746
Session Tag	747

SESSIONOBJECTS.FLUSH	747
String tags	748
STRING.FROMOBJECT	748
STRING.LENGTH	748
STRING.STREAM	749
STRING.TOOBJECT	750
Text Format Tags	751
TEXTFORMAT.LENGTH (Deprecated)	751
TEXTFORMAT.TOSTRING	751
URL Tags (Deprecated)	753
URL.PACK (Deprecated)	753
URL.UNPACK (Deprecated)	753
Visitor Data Manager Tags	755
VDM.FLUSHINACTIVE	756
VDM.GETACCESSID	756
VDM.GETALIAS	757
VDM.GETCOMMERCEID	757
VDM.SETACCESSID	758
VDM.SETALIAS	759
VDM.SETCOMMERCEID	759
4 CS-Engage XML Tags	761
Assetset Tags	762
ASSETSET.ESTABLISHRATINGS	762
Commerce Context tags	763
COMMERCECONTEXT.CALCULATEPROMOTIONS	763
COMMERCECONTEXT.CALCULATESEGMENTS	764
COMMERCECONTEXT.DISCOUNTCART	765
COMMERCECONTEXT.DISCOUNTTEMPCARD	765
COMMERCECONTEXT.GETPROMOTIONS	766
COMMERCECONTEXT.GETRATINGS	766
COMMERCECONTEXT.GETRECOMMENDATIONS	767
COMMERCECONTEXT.GETSEGMENTS	768
COMMERCECONTEXT.GETSINGLERECOMMENDATION	768
COMMERCECONTEXT.SETPROMOTIONCUTOFF	769
COMMERCECONTEXT.SETSEGMENTCUTOFF	770
Visitor Data Manager Tags	771
VDM.GETHISTORYCOUNT	772
VDM.GETHISTORYEARLIEST	773
VDM.GETHISTORYLATEST	773
VDM.GETHISTORYSUM	774
VDM.GETSCALAR	775
VDM.LOADSCALAROBJECT	775
VDM.RECORDHISTORY	776
VDM.SAVESCALAROBJECT	776

VDM.SETSCALAR.....	777
5 Analysis Connector XML Tags	779
Analysis Connector Security.....	780
Administrative tags.....	781
CDM.CREATETABLES.....	781
CDM.DELETETABLES.....	781
CDM.FLUSHTABLEROWS.....	782
Data Retrieval Tags.....	783
CDM.GETCOMMERCE DATA.....	783
CDM.GETTABLEDEF.....	784
Data Storage Tags.....	785
CDM.WRITECOMMERCE DATA.....	785
CDM.FITTOINTERVAL.....	786
CDM.CONVERTDATETOLONG.....	786
6 CS-Satellite XML Tags	789
Alphabetical List of CS-Satellite XML Tags.....	790
satellite.blob.....	790
satellite.cookie.....	794
satellite.flush.....	795
satellite.link.....	796
satellite.page.....	797
satellite.tag.....	799
7 CS-Bridge XML Tags	801
Alphabetical List of CS-Bridge XML Tags.....	802
XMLUTILS.PARSE.....	802
XMLUTILS.TRANSFORM.....	804
XMLUTILS.TRANSFORMANDPRINT.....	805
8 Content Server SOAP Web Services	807
SOAP tags.....	808
SOAP.ARGUMENT.....	808
SOAP.BODY.....	809
SOAP.FAULT.....	810
SOAP.HEADER.....	811
SOAP.MESSAGE.....	812
Web Services tags.....	813
WEBSERVICES.INVOKE.....	813
WEBSERVICES.PARAMETER.....	814

Section 2: JSP Tags

9 Content Server JSP Tags	817
JSP Tags by Function	818
ics Tags	818
object Tags	819
user Tags	820
time Tags	820
ftcs Tags	820
acl Tags	820
Directory Service Tags	821
Alphabetical List of JSP Tags	822
acl:argument	822
acl:create	823
acl:delete	824
acl:gather	825
acl:get	826
acl:list	827
acl:load	829
acl:save	830
acl:scatter	831
acl:set	832
dir:addattrs	833
dir:addgroupmember	834
dir:children	837
dir:create	838
dir:delete	840
dir:getattrs	842
dir:groupmembers	844
dir:groupmemberships	845
dir:listusers	847
dir:removeattrs	849
dir:removegroupmember	851
dir:replaceattrs	853
dir:search	855
ftcs1_0:flush	857
ftcs1_0:ftcs	857
ics:argument	858
ics:callelement	859
ics:callsql	860
ics:catalogbatch	862
ics:catalogdef	863
ics:catalogmanager	864
addrow	865
addrows	867

addrowtext	868
addUser (Deprecated)	869
commit	870
createtable	871
deletelog	872
deleterevision	873
deleterow	874
deleterows	876
deletetable	877
delUser (Deprecated)	877
editrow	878
editrows	880
exportform	881
exportlog	882
flushcatalog	883
flushpage	883
history	884
lock	885
login	886
logout	887
mirrorabort	887
mirrorrows	889
modifyUser (Deprecated)	891
release	892
replacerow	893
replaceroes	894
rollback	896
selectrows	897
setversions	898
tracktable	899
untracktable	900
unlockrecord	901
updaterow	902
updaterow2	903
updaterows	905
updaterows2	907
ics:cdata	909
ics:clearerrno	909
ics:copylist	910
ics:dateargument	911
ics:debug	912
ics:disablecache	913
ics:else	913
ics:encode	914
ics:fieldargument	915

ics:fileargument	916
ics:flushcatalog	917
ics:flushstream	917
ics:geterrno	918
ics:getproperty	918
ics:getssvar	919
ics:getvar	920
ics:if	921
ics:indexadd	922
ics:indexcreate	924
ics:indexdestroy	926
ics:indexexists	927
ics:indexremove	929
ics:indexreplace	930
ics:insertpage	932
ics:listget	933
ics:listloop	934
ics:literal	934
ics:logmsg	936
ics:mirror	936
ics:mirrorlist	938
ics:mirrortreenode	938
ics:popvars	939
ics:pushvars	940
ics:removessvar	940
ics:removevar	941
ics:resolvevariables	941
ics:search	943
ics:searchdatetolive	945
ics:selectto	947
ics:sendmail	949
ics:setssvar	950
ics:setvar	951
ics:sql	952
ics:sqlexp	953
ics:textargument	954
ics:then	955
ics:queryevents	955
ics:readevent	957
ics:treemanager	958
addchild	959
addchildren	960
copychild	962
createtree	963
delchild	964

delchildren	965
deletetree	966
findnode	967
getchildren	968
getnode	970
getparent	971
listtrees	972
movechild	973
nodepath	974
setobject	975
validatenode	977
verifypath	978
localestring:argument	979
localestring:getstring	979
name:get	980
name:makechild	981
object:argument	982
object:checkin	983
object:checkout	984
object:children	985
object:create	987
object:deinstall	988
object:delete	989
object:gather	990
object:get	991
object:install	992
object:isempty	994
object:list	995
object:load	996
object:rollback	999
object:root	1000
object:save	1002
object:scatter	1003
object:set	1004
object:setobject	1005
object:setparent	1006
object:stream	1007
object:track	1008
object:undocheckout	1009
object:untrack	1010
portal:logdep	1011
portal:randomval	1013
time:get	1013
time:set	1014
user:add (Deprecated)	1015

user:changepassword (Deprecated)	1016
user:delete (Deprecated).....	1017
user:login	1017
user:logout	1018
user:modify (Deprecated).....	1019
user:su.....	1020
10 CS-Direct JSP Tags	1021
Asset Tags.....	1022
asset:addchild	1023
asset:canedit	1024
asset:childgather	1024
asset:children	1025
asset:childtypes.....	1028
asset:create	1028
asset:def	1029
asset:export.....	1030
asset:gather	1031
asset:get	1032
asset:getassettype	1033
asset:getexportfile.....	1033
asset:getlegalsubtypes.....	1034
asset:getpubdeps.....	1035
asset:getsitenode.....	1036
asset:getsubtype	1037
asset:gettemplaterootelement	1038
asset:gettemplatefortarget.....	1039
asset:import.....	1040
asset:inspect	1041
asset:list	1042
asset:load	1043
asset:loadall	1045
asset:loadrevision	1047
asset:referencedby	1048
asset:removechildren	1048
asset:save	1049
asset:saveall	1050
asset:scatter.....	1051
asset:search.....	1053
asset:set.....	1054
asset:share.....	1055
asset:siteparent	1055
asset:sites	1057
asset:stream.....	1057
asset:void	1058

InSite Editor Tag	1059
insite:edit	1059
Property Tags	1061
property:get.	1061
Render Tags	1062
render:callelement	1063
render:contentserver.	1064
render:encode	1065
render:export.	1067
render:filter.	1068
render:getbloburl	1070
render:getpageurl	1072
render:logdep	1074
render:packargs.	1075
render:satelliteblob	1077
render:satellitepage.	1079
render:stream	1081
render:unknowndeps.	1082
Site Plan Tags	1083
siteplan:children	1084
siteplan:get	1085
siteplan:getpagename	1086
siteplan:listpages.	1088
siteplan:load	1089
siteplan:nodepath	1091
siteplan:place	1092
siteplan:root	1093
siteplan:setobject	1094
siteplan:setparent	1094
siteplan:unplace	1095
siteplan:unplacedancestor.	1096
11 CS-Direct Advantage JSP Tags	1097
Assetset Tags.	1098
assetset:getassetcount	1098
assetset:getassetlist	1099
assetset:getattributevaluecount	1100
assetset:getattributevalues	1100
assetset:getmultiplevalues	1101
assetset:setasset.	1102
assetset:setempty	1103
assetset:setlistedassets	1104
assetset:setsearchedassets.	1105
Blob Service Tags.	1106
blobservice:getidcolumn.	1106

blobservice:gettablename	1106
blobservice:geturlcolumn	1107
blobservice:readdata	1107
Calculator Tags	1108
calculator:go	1108
Cart Tags	1110
cart:additem	1110
cart:check	1112
cart:clear	1113
cart:cleardiscounts	1113
cart:commit	1114
cart:create	1114
cart:deleteitem	1115
cart:fromstring	1116
cart:getitemdiscounts	1116
cart:getcartdiscounts	1117
cart:getcartdiscounttotal	1117
cart:geterrors	1118
cart:getfinaltotal	1118
cart:getitemdiscounttotal	1119
cart:getitemerrors	1119
cart:getitemlegalvalues	1120
cart:getitemparameter	1120
cart:getitemparameters	1121
cart:getitems	1122
cart:getitemtotal	1123
cart:getlegalvalues	1123
cart:getparameter	1124
cart:getparameters	1125
cart:getpreliminarytotal	1126
cart:getshippingtotal	1126
cart:getstores	1127
cart:gettaxtotal	1127
cart:gettransactionid	1128
cart:gettransactionids	1128
cart:merge	1129
cart:setitemparameters	1129
cart:setitemquantity	1130
cart:setparameter	1130
cart:setparameters	1131
cart:tostring	1131
Cart Set Tags	1132
cartset:addcart	1132
cartset:cartids	1133
cartset:deletecart	1133

cartset:getcart	1134
cartset:getset	1134
Commerce Context Tags	1135
commercecontext:getabandonedcartsessions	1135
commercecontext:getcurrentcart	1136
commercecontext:getsessioncart	1137
commercecontext:logout	1137
commercecontext:saveall	1137
commercecontext:setcurrentcart	1138
CSMAC Tags	1139
csmac:cookievalue	1139
csmac:payload	1140
csmac:string	1141
csmac:transactserver	1141
CSUSER Tags	1143
csuser:authenticate	1143
csuser:clear	1144
csuser:create	1144
csuser:createpmtacct	1144
csuser:deletepmtacct	1145
csuser:fromstring	1146
csuser:getallpmtaccts	1146
csuser:geterrors	1147
csuser:getlegalpmtbrands	1147
csuser:getparameter	1148
csuser:getparameters	1148
csuser:lookup	1149
csuser:modify	1149
csuser:modifypmtacct	1150
csuser:register	1151
csuser:setparameter	1152
csuser:setparameters	1152
csuser:tostring	1153
Currency Tags	1154
currency:create	1154
currency:getcurrency	1155
currency:getisocode	1156
currency:getsymbol	1156
currency:readcurrency	1157
currency:round	1157
currency:rounddown	1158
currency:roundup	1158
Date Converter Tags	1160
date:clocklist	1160
date:convert	1161

date:defaulttz	1162
Decimal Tags	1163
decimal:create	1163
decimal:getdecimal	1163
decimal:getmaxplaces	1164
decimal:getminplaces	1164
decimal:getthousands	1165
decimal:readdecimal	1165
decimal:setmaxplaces	1166
decimal:setminplaces	1166
decimal:setthousands	1166
Hash Table Tags	1167
hash:add	1167
hash:contains	1167
hash:create	1168
hash:hasdata	1168
hash:toString	1169
List Object Tags	1170
listobject:addrow	1170
listobject:create	1171
listobject:toList	1172
Locale Tags	1173
locale:create	1173
Miscellaneous Tags	1174
misc:getlistcolumnnames	1174
Searchstate Tags	1175
searchstate:addlikeconstraint	1175
searchstate:addnestedconstraint	1176
searchstate:addrangeconstraint	1177
searchstate:addrichtextconstraint	1178
searchstate:addsimplelikeconstraint	1180
searchstate:addsimplestandardconstraint	1181
searchstate:addstandardconstraint	1182
searchstate:create	1183
searchstate:deleteconstraint	1183
searchstate:duplicate	1184
searchstate:findconstraint	1184
searchstate:fromstring	1186
searchstate:toString	1186
Session Tags	1187
sessionobjects:flush	1187
String Tags	1188
string:fromobject	1188
string:length	1188
string:stream	1189

string:toobject	1190
Text Format Tags	1191
textformat:length	1191
textformat:toString	1191
URL Tags (Deprecated)	1193
url:pack (Deprecated)	1193
url:unpack (Deprecated)	1193
Visitor Data Manager Tags	1195
vdm:flushinactive	1196
vdm:getaccessid	1197
vdm:getalias	1197
vdm:getcommerceid	1198
vdm:setaccessid	1198
vdm:setalias	1199
vdm:setcommerceid	1200
12 CS-Engage JSP Tags	1201
Assetset Tags	1202
assetset:establishratings	1202
Commerce Context Tags	1204
commercecontext:calculatepromotions	1205
commercecontext:calculatesegments	1205
commercecontext:discountcart	1206
commercecontext:discounttempcart	1206
commercecontext:getpromotions	1207
commercecontext:getratings	1207
commercecontext:getrecommendations	1208
commercecontext:getsegments	1209
commercecontext:getsinglerecommendation	1210
commercecontext:setpromotioncutoff	1211
commercecontext:setsegmentcutoff	1211
Visitor Data Manager Tags	1212
vdm:gethistorycount	1213
vdm:gethistoryearliest	1214
vdm:gethistorylatest	1214
vdm:gethistorysum	1215
vdm:getscalar	1216
vdm:loadscalarobject	1216
vdm:recordhistory	1217
vdm:savescalarobject	1217
vdm:setscalar	1218
13 CS-Satellite JSP Tags	1219
Alphabetical List of CS-Satellite JSP Tags	1220
satellite:blob	1220

satellite:cookie	1223
satellite:flush	1224
satellite:link	1226
satellite:page	1227
satellite:tag	1228
14 Content Server SOAP Web Services	1231
SOAP Tags	1232
SOAP:ARGUMENT	1232
soap:body	1233
soap:fault	1234
soap:header	1235
soap:message	1236
Web Services Tags	1238
webservices:invoke	1238
webservices:parameter	1239

Section 3: AssetMaker Utility: Tags and Overview

15 AssetMaker XML Tags	1243
Overview: AssetMaker and Asset Descriptor Files	1244
ASSET	1245
PROPERTIES	1247
PROPERTY	1247
STORAGE	1248
INPUTFORM	1249
SEARCHFORM	1260
SEARCHRESULTS	1267
Restrictions and Dependencies with TYPE	1268
CHAR (STORAGE TYPE="CHAR")	1268
VARCHAR (STORAGE TYPE="VARCHAR")	1269
SMALLINT	1270
INTEGER (STORAGE TYPE="INTEGER")	1271
BIGINT (STORAGE TYPE="BIGINT")	1272
DOUBLE (STORAGE TYPE="DOUBLE")	1273
TIMESTAMP (STORAGE TYPE="TIMESTAMP")	1274

Section 4: CS-Direct Advantage Parameters

16 CS-Direct Advantage Cart Parameters	1277
Order Parameters	1279

Required Cart Parameters	1279
billToAddress-name1	1279
billToAddress-name2	1279
billToAddress-address1	1280
billToAddress-address2	1280
billToAddress-city	1280
billToAddress-state	1280
billToAddress-postalCode	1281
billToAddress-country	1281
billToAddress-county	1281
billToAddress-emailAddress	1281
billToAddress-telephone	1281
paymentInfo-brandID	1282
paymentInfo-ccn	1282
paymentInfo-expirationDate	1282
paymentInfo-accountID	1282
Buyer Parameters	1283
buyer-userID	1283
buyer-accessName	1283
buyer-password	1283
Optional Cart Parameters	1283
shipToAddress-name1	1283
shipToAddress-name2	1283
shipToAddress-address1	1284
shipToAddress-address2	1284
shipToAddress-city	1284
shipToAddress-state	1284
shipToAddress-postalCode	1284
shipToAddress-country	1284
shipToAddress-county	1284
shipToAddress-emailAddress	1285
shipToAddress-telephone	1285
Parameters Returned by Checking Carts	1285
subTotal	1285
shippingInfo-shippingCost	1285
shippingInfo-shipperID	1285
shippingInfo-shipperName	1285
shippingInfo-shippingTax	1286
tax	1286
totalSalesAmount	1286
store	1286
Other Returned Parameters	1287
Parameters Returned by Committing Carts	1287
orderDate	1287
orderID	1287

orderNumber	1287
smartReceipt	1288
17 CS-Direct Advantage Order Item Parameters	1289
Alphabetical List of Order Item Parameters	1290
Order Item Parameters by Function	1291
General Parameters	1292
Product Description Parameters	1292
Delivery Parameters	1293
Cost and Payment Parameters	1293
Subscription Parameters	1294
Tax Parameters	1295
Custom Field Parameters	1295
Detailed Descriptions of Parameters	1296
AccountRequired	1296
AllowPaymentBrands	1296
AutoRenew	1296
CF__fieldname	1297
CF_R_fieldname	1297
CodeSet	1297
DeliveryMechanism	1297
Detail	1298
ExcludePaymentBrands	1298
FulfillmentURL	1298
GracePeriod	1299
InstallmentCount	1299
InstallmentPeriod	1300
Name	1300
OfferURL	1300
ReceiptDuration	1301
RefundPolicy	1301
ShippingCharge	1302
ShippingWeight	1303
Subscription	1303
SubscriptionDuration	1303
TaxClass	1303
TrialPeriod	1304
Type	1304
UniqueID	1304
VatFlag	1304
VatRate	1305
18 CS-Direct Advantage Buyer Parameters	1307
accessName	1308
password	1308

locale.	1308
address-name1	1308
address-name2	1308
address-address1.	1308
address-address2.	1309
address-city.	1309
address-state	1309
address-postalCode.	1309
address-country.	1309
address-county	1309
address-emailAddress.	1309
address-telephone	1309
address-fax	1309
userID.	1310

Section 5: Appendixes

A Error Conditions.	1313
errno Error Values.	1314
errdetail Error Values	1329
BlobServer and EvalServer	1329
B CSEE Reserved Variables	1331
C TimeZoneID Numbers	1333
Index	1345

Summary of Changes

The *CSEE Developer's Tag Reference* provides reference information for web site developers who are developing elements with CSEE XML and JSP tags.

This chapter describes tags that were new, modified, or documented for the first time in CSEE version 5.0. It contains the following sections:

- [New in Version 5.0](#)
- [Modified in Version 5.0](#)
- [Documented in Version 5.0](#)

New in Version 5.0

This section describes new chapters that were added to the CSEE Developer's Tag Reference for CSEE version 5.0 and lists tags that were introduced in version 5.0.

Chapters

Chapter Name	Product
Chapter 7, "CS-Bridge XML Tags"	CS-Bridge XML
Chapter 8, "Content Server SOAP Web Services"	Content Server
Chapter 14, "Content Server SOAP Web Services"	Content Server
Appendix C, "TimeZoneID Numbers"	Content Server

XML Tags

Tag Name	Product
"ICS.CDATA" on page 192	Content Server
"ICS.CLEARERRNO" on page 193	Content Server
"ICS.GETERRNO" on page 194	Content Server
"ICS.LISTGET" on page 197	Content Server
"ICS.LITERAL" on page 198	Content Server
"ICS.RESOLVEVARIABLES" on page 202	Content Server
"ICS.SQLEXP" on page 204	Content Server
"LOCALESTRING.ARGUMENT" on page 231	Content Server
"LOCALESTRING.GETSTRING" on page 232	Content Server
"mail.send" on page 237	Content Server
"OBJECT.ISEMPY" on page 251	Content Server
"portal.logdep" on page 268	Content Server
"PORTAL.RANDOMVAL" on page 269	Content Server
"DATEFORMAT.CREATE" on page 424	CS-Direct
"DATEFORMAT.GETDATE" on page 425	CS-Direct
"DATEFORMAT.GETDATETIME" on page 426	CS-Direct
"DATEFORMAT.GETTIME" on page 427	CS-Direct
"SITEPLAN.NODEPATH" on page 497	CS-Direct

JSP Tags

Tag Name	Product
“ics:cdata” on page 909	Content Server
“ics:sqlexp” on page 953	Content Server
“object:isempty” on page 994	Content Server
“ics:literal” on page 934	Content Server
“localestring:argument” on page 979	Content Server
“localestring:getstring” on page 979	Content Server
“siteplan:nodepath” on page 1091	CS-Direct

Modified in Version 5.0

The following tags were modified in CSEE version 5.0

XML Tags

Tag Name	Product
“CALLSQL” on page 74	Content Server
“EXECSQL” on page 181	Content Server
“ICS.GETVAR” on page 196	Content Server
“OBJECT.CREATE” on page 244	Content Server
“OBJECT.DEINSTALL” on page 246	Content Server
“OBJECT.INSTALL” on page 250	Content Server
“Release Introduced” on page 252	Content Server
“OBJECT.LOAD” on page 254	Content Server
“replaceroes” on page 124	Content Server
“updaterow” on page 134	Content Server
“updaterow2” on page 137	Content Server
“updaterows” on page 139	Content Server
“updaterows2” on page 141	Content Server
“ASSET.LOAD” on page 399	CS-Direct
“INSITE.EDIT” on page 446	CS-Direct
“RENDER.GETBLOBURL” on page 463	CS-Direct
“RENDER.LOGDEP” on page 468	CS-Direct
“RENDER.SATELLITEBLOB” on page 471	CS-Direct
“ASSETSET.SETASSET” on page 667	CS-Direct Advantage
“ASSETSET.SETEMPTY” on page 668	CS-Direct Advantage
“ASSETSET.SETLISTEDASSETS” on page 668	CS-Direct Advantage
“ASSETSET.SETSEARCHEDASSETS” on page 669	CS-Direct Advantage
“LOCALE.CREATE” on page 733	CS-Direct Advantage
“STRING.STREAM” on page 749	CS-Direct Advantage
“satellite.blob” on page 790	CS-Satellite
“satellite.cookie” on page 794	CS-Satellite

Tag Name	Product
“satellite.page” on page 797	CS-Satellite

JSP Tags

Tag Name	Product
“ics:callsql” on page 860	Content Server
“ics:getvar” on page 920	Content Server
“ics:listget” on page 933	Content Server
“ics:resolvevariables” on page 941	Content Server
“ics:sql” on page 952	Content Server
“object:create” on page 987	Content Server
“object:deinstall” on page 988	Content Server
“object:install” on page 992	Content Server
“object:list” on page 995	Content Server
“object:load” on page 996	Content Server
“render:callelement” on page 1063	CS-Direct
“render:getbloburl” on page 1070	CS-Direct
“render:logdep” on page 1074	CS-Direct
“render:satelliteblob” on page 1077	CS-Direct
“assetset:setasset” on page 1102	CS-Direct Advantage
“assetset:setempty” on page 1103	CS-Direct Advantage
“assetset:setlistedassets” on page 1104	CS-Direct Advantage
“assetset:setsearchedassets” on page 1105	CS-Direct Advantage
“locale:create” on page 1173	CS-Direct Advantage
“string:stream” on page 1189	CS-Direct Advantage
“satellite:blob” on page 1220	CS-Satellite
“satellite:cookie” on page 1223	CS-Satellite
“satellite:link” on page 1226	CS-Satellite
“satellite.page” on page 1227	CS-Satellite

Documented in Version 5.0

The following groups of tags or individual tags were not new in CSEE version 5.0 but were documented for the first time for the 5.0 release.

XML Tags

Group or Tag Name	Product
“ICS.SQL” on page 203	Content Server
Variables and Built-in Functions	Content Server
“ASSET.ADDCHILD” on page 374	CS-Direct
“ASSET.CANEDIT” on page 375	CS-Direct
“ASSET.CHILDGATHER” on page 376	CS-Direct
“ASSET.CHILDTYPES” on page 379	CS-Direct
“ASSET.CREATE” on page 380	CS-Direct
“ASSET.DEF” on page 381	CS-Direct
“ASSET.EXPORT” on page 382	CS-Direct
“ASSET.GATHER” on page 384	CS-Direct
“ASSET.GETASSETTYPE” on page 387	CS-Direct
“ASSET.GETLEGALSUBTYPES” on page 387	CS-Direct
“ASSET.GETLEGALSUBTYPES” on page 387	CS-Direct
“ASSET.GETPUBDEPS” on page 388	CS-Direct
“ASSET.GETSUBTYPE” on page 391	CS-Direct
“ASSET.GETTEMPLATEFORTARGET” on page 392	CS-Direct
“ASSET.GETTEMPLATEROOTELEMENT” on page 391	CS-Direct
“ASSET.IMPORT” on page 394	CS-Direct
“ASSET.INSPECT” on page 397	CS-Direct
“ASSET.LOADALL” on page 401	CS-Direct
“ASSET.LOADREVISION” on page 403	CS-Direct
“ASSET.REFERENCEDBY” on page 404	CS-Direct
“ASSET.REMOVECHILDREN” on page 405	CS-Direct
“ASSET.SAVE” on page 406	CS-Direct
“ASSET.SAVEALL” on page 407	CS-Direct

Group or Tag Name	Product
“ASSET.SEARCH” on page 410	CS-Direct
“ASSET.SET” on page 412	CS-Direct
“ASSET.SHARE” on page 413	CS-Direct
“ASSET.SITES” on page 415	CS-Direct
“ASSET.STREAM” on page 416	CS-Direct
“ASSET.VOID” on page 417	CS-Direct
“PROPERTY.GET” on page 454	CS-Direct
“RENDER.EXPORT” on page 461	CS-Direct
“RENDER.STREAM” on page 475	CS-Direct
“SITEPLAN.PLACE” on page 498	CS-Direct
“SITEPLAN.SETOBJECT” on page 500	CS-Direct
“SITEPLAN.SETPARENT” on page 500	CS-Direct
“SITEPLAN.UNPLACE” on page 501	CS-Direct
“SITEPLAN.UNPLACEDANCESTOR” on page 502	CS-Direct
“WORKFLOWASSIGNMENT.GETDISPOSITIONUSE RID” on page 540	CS-Direct
“WORKFLOWASSIGNMENT.GETID” on page 543	CS-Direct
“WORKFLOWASSIGNMENT.SCATTER” on page 546	CS-Direct
“WORKFLOWGROUP.GATHER” on page 599	CS-Direct
“WORKFLOWGROUP.GETGROUPACTIONS” on page 602	CS-Direct
“WORKFLOWGROUP.GETID” on page 603	CS-Direct
“WORKFLOWGROUP.GETSITES” on page 605	CS-Direct
“WORKFLOWGROUP.SCATTER” on page 606	CS-Direct
“WORKFLOWGROUP.SETDELETEROLES” on page 607	CS-Direct
“WORKFLOWGROUP.SETNAME” on page 610	CS-Direct
CCUser Tags	CS-Direct
Email Manager Tags	CS-Direct
Email Tags	CS-Direct
Participants Tags	CS-Direct

Group or Tag Name	Product
Role List Tags	CS-Direct
Site List Tags	CS-Direct
Type List Tags	CS-Direct
User List Tags	CS-Direct
User Manager Tags	CS-Direct
Workflow Action List Tags	CS-Direct
Workflow Action Tags	CS-Direct
Workflow Engine Tags	CS-Direct
Workflow Function Tags	CS-Direct
Workflow Priv Tags	CS-Direct
Workflow Privs Tags	CS-Direct
Workflow Process Tags	CS-Direct
Workflow State Tags	CS-Direct
Workflow Step Tags	CS-Direct
Workflow Timed Action List Tags	CS-Direct
Workflow Timed Action Tags	CS-Direct

JSP Tags

Group or Tag Name	Product
“asset:addchild” on page 1023	CS-Direct
“asset:canedit” on page 1024	CS-Direct
“asset:childgather” on page 1024	CS-Direct
“asset:childtypes” on page 1028	CS-Direct
“asset:create” on page 1028	CS-Direct
“asset:def” on page 1029	CS-Direct
“asset:export” on page 1030	CS-Direct
“asset:gather” on page 1031	CS-Direct
“asset:getassettype” on page 1033	CS-Direct
“asset:getexportfile” on page 1033	CS-Direct

Group or Tag Name	Product
“asset:getlegalsubtypes” on page 1034	CS-Direct
“asset:getpubdeps” on page 1035	CS-Direct
“asset:getsitenode” on page 1036	CS-Direct
“asset:getsubtype” on page 1037	CS-Direct
“asset:gettemplaterootelement” on page 1038	CS-Direct
“asset:gettemplatefortarget” on page 1039	CS-Direct
“asset:import” on page 1040	CS-Direct
“asset:inspect” on page 1041	CS-Direct
“asset:loadall” on page 1045	CS-Direct
“asset:loadrevision” on page 1047	CS-Direct
“asset:referencedby” on page 1048	CS-Direct
“asset:removechildren” on page 1048	CS-Direct
“asset:save” on page 1049	CS-Direct
“asset:saveall” on page 1050	CS-Direct
“asset:search” on page 1053	CS-Direct
“asset:set” on page 1054	CS-Direct
“asset:share” on page 1055	CS-Direct
“asset:sites” on page 1057	CS-Direct
“asset:stream” on page 1057	CS-Direct
“asset:void” on page 1058	CS-Direct
“insite:edit” on page 1059	CS-Direct
“property:get” on page 1061	CS-Direct
“siteplan:nodepath” on page 1091	CS-Direct
“siteplan:setobject” on page 1094	CS-Direct
“siteplan:setparent” on page 1094	CS-Direct
“siteplan:unplace” on page 1095	CS-Direct
“siteplan:unplacedancestor” on page 1096	CS-Direct

Section 1

XML Tags

The following chapters in this section describe the XML tags used in Content Server, CS-Satellite and the Content Server content applications:

- [Chapter 1, “Content Server XML Tags”](#)
- [Chapter 2, “CS-Direct XML Tags”](#)
- [Chapter 3, “CS-Direct Advantage XML Tags”](#)
- [Chapter 4, “CS-Engage XML Tags”](#)
- [Chapter 5, “Analysis Connector XML Tags”](#)
- [Chapter 6, “CS-Satellite XML Tags”](#)
- [Chapter 7, “CS-Bridge XML Tags”](#)
- [Chapter 8, “Content Server SOAP Web Services”](#)

Chapter 1

Content Server XML Tags

This chapter describes Content Server XML tags.

- [XML Tags by Function](#)
- [Alphabetical List of XML Tags](#)
- [Variables and Built-in Functions](#)

XML Tags by Function

This section groups XML tags by function.

Note

You must type Content Server XML tags in uppercase letters if specified in the syntax. If you use lowercase tags, the debugging and error checking facilities may not detect coding errors in your elements.

Variables

- [BEGINS](#)
- [CSVAR](#)
- [DEBABBLE](#)
- [ENDS](#)
- [INDEXOF](#)
- [REMOVEVAR](#)
- [REPLACE](#)
- [REPLACEALL](#)
- [SETVAR](#)
- [SUBSTITUTE](#)
- [SUBSTRING](#)
- [TOLOWER](#)
- [TOUPPER](#)

Lists

- [COPYLIST](#)
- [FILELIST](#)
- [FLUSH](#)
- [GOTOROW](#)
- [ISINLIST](#)
- [LOOP](#)
- [RENAMELIST](#)
- [SETROW](#)
- [STRINGLIST](#)

Counters

- [INCCOUNTER](#)
- [REMOVECOUNTER](#)

- [SETCOUNTER](#)

Sessions and Cookies

- [REMOVECOOKIE](#)
- [REMOVESSVAR](#)
- [SETCOOKIE](#)
- [SETSSVAR](#)

Database Access

- [CALLSQL](#)
- [CATALOGDEF](#)
- [CATALOGMANAGER](#)
- [EXECSQL](#)
- [LOADPROPERTY](#)
- [RESTOREPROPERTY](#)
- [SELECTTO](#)
- [SQLEXP](#)

Directory Services

- [DIR.ADDATTRS](#)
- [DIR.ADDGROUPMEMBER](#)
- [DIR.CHILDREN](#)
- [DIR.CREATE](#)
- [DIR.DELETE](#)
- [DIR.GETATTRS](#)
- [DIR.GROUPMEMBERS](#)
- [DIR.GROUPMEMBERSHIPS](#)
- [DIR.LISTUSERS](#)
- [DIR.REMOVEATTRS](#)
- [DIR.REMOVEGROUPMEMBER](#)
- [DIR.REPLACEATTRS](#)
- [DIR.SEARCH](#)

Execution

- [ARGUMENT](#)
- [CALLAPPLLOGIC](#) (Deprecated)
- [CALLELEMENT](#)
- [CALLJAVA](#)

- [CATALOGMANAGER](#)
- [CONTENTSERVER \(Deprecated\)](#)
- [FILELIST](#)
- [SYSEXEC](#)
- [THROWEXCEPTION](#)
- [TREEMANAGER](#)

Files and URLs

- [DELETEFILE](#)
- [INSERT](#)
- [POST](#)
- [WRITEFILE](#)
- [XMLFILTER](#)

Events and Email

- [APPEVENT](#)
- [DESTROYEVENT](#)
- [DISABLEEVENT](#)
- [EMAILEVENT](#)
- [ENABLEEVENT](#)
- [SENDMAIL \(Variant 1\)](#)

Search Engines

- [INDEXADD](#)
- [INDEXCREATE](#)
- [INDEXDESTROY](#)
- [INDEXEXISTS](#)
- [INDEXREMOVE](#)
- [INDEXREPLACE](#)
- [SEARCH](#)
- [SEARCHDATENATIVE](#)

Revision Tracking

- [COMMIT](#)
- [DELETEREVISION](#)
- [HISTORY](#)
- [LOCK](#)
- [RELEASE](#)

- RETRIEVEREVISION
- ROLLBACK
- SETVERSIONS
- TRACKTABLE
- UNLOCKRECORD
- UNTRACKTABLE

Conditions

- ICS.ARGUMENT
- IsCatalog
- IsColumn
- IsCounter
- IsError
- IsList
- ISINLIST
- IsObject
- IsSessionVariable
- IsTracked
- IsTree
- IsVariable

Built-in Functions

- CSVAR
- CS.Browser
- CS.CacheFile
- CS.CatalogDir
- CS.Date
- CS.DDate
- CS.Day
- CS.Hour
- CS.Hour24
- CS.IOError
- CS.Min
- CS.Mon
- CS.Property
- CS.SQLDate
- CS.SystemAssetsRoot

- [CS.UniqueID](#)
- [CS.Year](#)

Miscellaneous Tags

- [DISABLECACHE](#)
- [USERISMEMBER](#)
- [INDEXOF](#)
- [LOGMSG](#)
- [SETDATE](#)

ACL Tags

- [ACL.ARGUMENT](#)
- [ACL.CREATE](#)
- [ACL.DELETE](#)
- [ACL.GATHER](#)
- [ACL.GET](#)
- [ACL.LIST](#)
- [ACL.LOAD](#)
- [ACL.SAVE](#)
- [ACL.SCATTER](#)
- [ACL.SET](#)

Alphabetical List of XML Tags

The following section describes each Content Server XML tag beginning with [ACL.ARGUMENT](#).

ACL.ARGUMENT

Argument for other ACL tags.

Syntax

```
<ACL.ARGUMENT NAME="Some Name"
              VALUE="Some Value"/>
```

Parameters

NAME (required)
Name of the argument.

VALUE (required)
Value of the argument.

Description

The ACL.ARGUMENT tag declares a name/value pair that is available to the parent tag. The parent tag must be present in the ACL tag library.

errno

None.

Example

The following example creates an object called ACL1 in the object pool:

```
<ACL.CREATE NAME="ACL1">
  <ACL.ARGUMENT NAME="aclname" value="myacl"/>
  <ACL.ARGUMENT NAME="cataccess" value="1"/>
  <ACL.ARGUMENT NAME="description" value="test"/>
</ACL.CREATE>
```

See Also

[ACL.CREATE](#)
[ACL.LIST](#)

ACL.CREATE

Creates an ACL.

Syntax

```
<ACL.CREATE NAME="ACL Name">  
  <ACL.ARGUMENT NAME="Argument Name"  
    VALUE="Argument Value"/>  
</ACL.CREATE>
```

Parameter

NAME (required)

Name of the object in the object pool.

Description

Creates an unsaved ACL object in the object pool with the specified name. Use the ACL.ARGUMENT tag to specify names, descriptions and privileges.

Values for cataccess (the numeric value that determines the level of permissions) are as follows:

- read = 1
- write = 2
- create = 4
- destroy = 8
- retrieve = 10
- rtaudit = 20
- rtadmin = 40

The values for cataccess work similarly to UNIX permissions. For example, if you want a user to have read and write permission, assign 3 to cataccess.

errno

The possible values of errno include:

Value	Description
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10010	Invalid attribute value.

Example

The following example creates an object called ACL1 in the object pool:

```
<ACL.CREATE NAME="ACL1">
  <ACL.ARGUMENT NAME="aclname" value="myacl"/>
  <ACL.ARGUMENT NAME="cataccess" value="1"/>
  <ACL.ARGUMENT NAME="description" value="test"/>
</ACL.CREATE>
```

See Also

[ACL.ARGUMENT](#)
[ACL.DELETE](#)
[ACL.GATHER](#)
[ACL.GET](#)
[ACL.LIST](#)
[ACL.LOAD](#)
[ACL.SAVE](#)
[ACL.SCATTER](#)
[ACL.SET](#)

ACL.DELETE

Deletes an ACL.

Syntax

```
<ACL.DELETE NAME="Some ACL Name"/>
```

Parameter

NAME (required)

Name of the ACL object in the object pool.

Description

This tag deletes an ACL from the ACL pool. To delete an ACL, it must first be loaded using ACL.LOAD.

errno

The possible values of errno include:

Value	Description
-105	Database error
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

Example

The following example deletes ACL1:

```
<ACL.DELETE NAME="ACL1"/>
```

See Also

[ACL.CREATE](#)
[ACL.GATHER](#)
[ACL.GET](#)
[ACL.LIST](#)
[ACL.LOAD](#)
[ACL.SAVE](#)
[ACL.SCATTER](#)
[ACL.SET](#)

ACL.GATHER

Gathers fields into an ACL object.

Syntax

```
<ACL.GATHER NAME="ACL Name"
            PREFIX="Some Prefix"
            FIELDLIST="List of Field Names"/>
```

Parameters

NAME (required)

Name of the ACL.

PREFIX (required)

Specifies the name of a string to be used as the prefix for variable names, which may exist in the environment. When the ACL.GATHER tag looks for variables in the environment, a colon (:) is used to separate the prefix from the field name.

FIELDLIST (required)

Specifies a comma-separated list of field names that should be gathered. Any field in the list that does not exist in the environment is cleared. If this parameter is not specified, then the ACL.GATHER tag looks at all fields in the environment; if no field names are found, the existing value is maintained.

Description

This tag retrieves name/value pairs from the environment, and sets them into a loaded ACL object. The name/value pairs are grouped together using a prefix and a prefix separator (a colon).

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.

Value	Description
-10005	The requested object is not in the object pool (is not loaded into memory).

Example

The following example gathers the `aclname` and `description` fields from the environment into `ACL1` using the prefix `SAMPLE`:

```
<ACL.GATHER NAME="ACL1"
            PREFIX="SAMPLE"
            FIELDLIST="aclname,description"/>
```

See Also

[ACL.CREATE](#)
[ACL.GET](#)
[ACL.LIST](#)
[ACL.LOAD](#)
[ACL.SAVE](#)
[ACL.SCATTER](#)
[ACL.SET](#)

ACL.GET

Retrieves the value of a single field from a loaded asset object and stores it in a Content Server variable.

Syntax

```
<ACL.GET NAME="Some Object Name"
        FIELD="Some field"
        [OUTPUT="output variable"]/>
```

Parameters

NAME (required)

Name of the ACL.

FIELD (required)

Name of the field to retrieve.

OUTPUT (optional)

Specifies the name of the output variable, which should be created in the environment. If this parameter is not specified, the variable is given the same name as the field.

Description

This tag obtains the value (or contents) from the specified field of a loaded asset and stores it in the specified output variable.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

Example

The following example copies the `ac lname` column from a loaded ACL object into a variable called `NAME`:

```
<ACL.GET NAME="ACL1"
      FIELD="ac lname"
      OUTPUT="name"/>
```

See Also

[ACL.CREATE](#)
[ACL.GATHER](#)
[ACL.LIST](#)
[ACL.LOAD](#)
[ACL.SAVE](#)
[ACL.SCATTER](#)
[ACL.SET](#)

ACL.LIST

Queries the database and retrieves a list of ACLs that meet the specified criteria. It creates a list of ACLs of one type.

Syntax

```
<ACL.LIST LIST="Name of List"
      [ORDER="List of Field Names"]/>
<ACL.ARGUMENT NAME="ARGUMENT NAME" VALUE="ARGUMENT VALUE"/>
</ACL.LIST>
```

Parameters

LIST (required)

Specifies the name of the list in the list pool. This is the same list pool used by other Content Server tags that generate lists. If a list exists with the same name, it is replaced by the new list.

ORDER (optional)

Specifies a comma-separated list of field names that should be used to order the list of ACLs.

Description

This tag builds a list of objects that meet the specified criteria. The criteria are a list of name/value pairs, which must be present in the object.

errno

The possible values of `errno` include:

Value	Description
-101	No rows returned.
-105	Database error.
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing.
-10010	Invalid attribute value.

Example

The following example lists all ACLs with a `cataccess` column equal to 1:

```
<ACL.LIST LIST="List2"/>
  <ACL.ARGUMENT NAME="cataccess" VALUE="1"/>
</ACL.LIST/>
```

See Also

[ACL.ARGUMENT](#)

ACL.LOAD

Queries the database for the specified ACL and then loads an instance of the ACL into memory as an object.

Syntax

```
<ACL.LOAD NAME="Name of the ACL in Object Pool"
  ACL="Some ACL Name"/>
```

Parameters

NAME (required)
Name of the ACL in the object pool.

ACL (required)
Name of the ACL in the database.

Description

This tag executes a database query to retrieve an instance of the specified ACL and then saves the instance into memory, assigning it the name provided by the NAME parameter. Object names are global in scope.

You identify the ACL that you want to load with its ACL name.

errno

The possible values of `errno` include:

Value	Description
-101	No rows returned.
-105	Database error.
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing.
-10006	The object ID is not valid.
-10007	The version of the object is not valid.
-10008	The node ID of the object is not valid.
-10010	Invalid attribute value.

Example

The following example loads the Site God ACL into the object pool.

```
<ACL.LOAD NAME="ACL1"
        ACL="Site God"/>
```

See Also

[ACL.CREATE](#)
[ACL.GATHER](#)
[ACL.SAVE](#)
[ACL.SCATTER](#)
[ACL.SET](#)

ACL.SAVE

Saves an ACL.

Syntax

```
<ACL.SAVE NAME="ACL Name"
          [FLUSH="true or false"]/>
```

Parameters

NAME (required)

Name of the ACL to be saved.

FLUSH (optional)

Specifies that upon success, the ACL should be flushed from the object pool. A `true` value indicates the ACL should be flushed. The default value is `false`.

Description

The `ACL.SAVE` tag saves a loaded ACL into storage. The ACL must have been previously created or loaded into the object pool. Saving an ACL either creates a new ACL in storage or updates an existing ACL in storage.

errno

The possible values of `errno` include:

Value	Description
-105	Database error.
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

Example

The following example saves `ACL1` and removes it from memory:

```
<ACL.SAVE NAME="ACL1"
          FLUSH="true"/>
```

See Also

[ACL.CREATE](#)
[ACL.DELETE](#)
[ACL.GATHER](#)
[ACL.GET](#)
[ACL.LIST](#)
[ACL.LOAD](#)
[ACL.SCATTER](#)
[ACL.SET](#)

ACL.SCATTER

Retrieves and lists ("scatters") the values from all the fields of a loaded ACL object. (To obtain just one field value, use ACL.GET.)

Syntax

```
<ACL.SCATTER NAME="Some ACL Name"
              [PREFIX="Some Prefix"]/>
```

Parameters

NAME (required)

Name of the ACL.

PREFIX (optional)

Specifies the the name of a string for the variable names that is created in the environment. When the ACL.SCATTER tag creates a variable in the environment, a colon (:) is used to separate the prefix from the field name.

Description

This tag retrieves values from the fields of a previously loaded ACL and then writes them into memory as variables. The variables are constructed with the prefix that you supply with the PREFIX parameter. After you use this tag, you can display the information contained in the ACL's fields. This tag is the preferred alternative to using multiple ACL.GET tags.

errno

The possible values of errno include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

Example

The following example creates a variable in the environment with the prefix, sample:

```
<ACL.SCATTER NAME="ACL1"
              PREFIX="sample"/>
```

See Also

[ACL.CREATE](#)
[ACL.DELETE](#)
[ACL.GATHER](#)
[ACL.GET](#)
[ACL.LIST](#)
[ACL.LOAD](#)
[ACL.SET](#)

ACL.SET

Sets a field in an ACL.

Syntax

```
<ACL.SET NAME="Some ACL Name"
        FIELD="Some Field"
        VALUE="Some Value"/>
```

Parameters

NAME (required)
Name of the ACL.

FIELD (required)
Field to be set.

VALUE (required)
Value to be set in the field.

Description

This tag sets the value of a field in a previously loaded ACL.

errno

The possible values of errno include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

Example

The following example sets the value to Editor in the aclname field of ACL1:

```
<ACL.SET NAME="ACL1"
        FIELD="aclname"
        VALUE="Editor"/>
```

See Also

[ACL.GATHER](#)
[ACL.GET](#)
[ACL.LIST](#)
[ACL.LOAD](#)
[ACL.SCATTER](#)

APPEVENT

Creates an event that executes an application logic at a specified time.

Syntax

```
<APPEVENT NAME="APP_EVENT_NAME"
          GUID="GUID"
          TIMES="hh:mm:ss W/DD/MM">
  <ARGUMENT NAME="ARG_NAME" VALUE="ARG_VALUE"/>
</APPEVENT>
```

Parameters

NAME (required)

Name of the event to create.

GUID (required)

Name of the servlet to be called when the event is triggered.

TIMES (required)

The time at which the event is triggered. Use the following format:

hh:mm:ss W/DD/MM

where:

- *hh*: (hours) 0 - 23
- *mm*: (minutes) 0 - 59
- *ss*: (seconds) 0 - 59
- *W* (day of the week): 0-6 with 0 = Sunday.
- *DD* (day of the month): 1 - 31
- *MM* (month): 1 - 12

For each of these fields, you can use an asterisk (all legal values) or a list of elements separated by commas. A value can be one or more numbers separated by a minus sign to indicate an inclusive range. For example, 2, 5 - 7:0:0 5/*/* means the event is triggered at 2 a.m., 5 a.m., 6 a.m., and 7 a.m. every Friday.

Specify days by two fields: day of the month (DD) and day of the week (W). If you specify both, both take effect. For example, 1:0:0 1/15/* means the event is triggered at 1 a.m. every Monday, as well as on the fifteenth of each month. To specify days by only one field, set the other field to an asterisk (*).

Description

The APPEVENT tag creates an event that executes an application logic at a specified time. Values can be passed to the application logic when it is called by using the ARGUMENT tag.

errno

The possible values of `errno` include:

Value	Description
-200	Failed to register event.

Value	Description
-201	Exception in event. (May appear as -200.)

Example

The following example registers an event that is triggered every day at 1:10 in the afternoon. At that time, Content Server invokes the pagename API/Main:

```
<SETVAR NAME="errno" VALUE="0" />
<APPEVENT NAME="AppEvent"
    GUID="ContentServer"
    TIMES="13:10:00 */*/*">
    <ARGUMENT NAME="pagename" VALUE="API/Main"/>
</APPEVENT>
<IF COND="IsError.Variables.errno=true">
    <THEN>
        <!--Handle error-->
    </THEN>
</IF>
```

See Also

[ARGUMENT](#)

ARGUMENT

Declares a value that is available to the parent tag.

Syntax

```
<ARGUMENT NAME="ARG NAME"
    VALUE="ARG VALUE"/>
```

Parameters

NAME (required)

Name of the argument.

VALUE (required)

Value for this argument.

Description

The ARGUMENT tag declares a value that is available to the parent tag. It is used in the context of another tag, and causes a runtime error if it is not contained within a proper parent tag. Depending on the parent tag used, the following are variants of the ARGUMENT tag:

- CGIARGUMENT
- FILEARGUMENT
- TEXTARGUMENT

- FIELDARGUMENT
- INDEXARGUMENT

Note

Use `<CGIARGUMENT NAME="name"/>` to specify that certain CGI parameters be made available to the called module.

errno

The value of `errno` is unchanged.

Example

The following example uses four `ARGUMENT` tags to specify name/value pairs to a `<CATALOGMANAGER>` command:

```
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd"  VALUE="addrow"/>
  <ARGUMENT NAME="tablename" VALUE="HorrorBooks"/>
  <ARGUMENT NAME="title"  VALUE="The Shining"/>
  <ARGUMENT NAME="author" VALUE="Stephen King"/>
</CATALOGMANAGER>
```

BEGINS

Determines whether a string starts with the specified substring.

Syntax

```
<BEGINS STR="STRING_VALUE"
        WHAT="BEGINSTRING_VALUE"/>
```

Parameters

STR (required)

String to search.

WHAT (required)

Value of the string to search for within the value for `STRING_VALUE`.

Description

The `BEGINS` tag determines whether a string starts with the specified substring. The comparison is case-sensitive. For more information about variables, see the *CSEE Developer's Guide*.

errno

The possible values of `errno` include:

Value	Description
1	Comparison is true.

Value	Description
0	Comparison is false.

Example

This example determines whether the value of the variable `teststr` begins with the string "hello". It displays "yes" if the comparison is true and "no" if the comparison is false:

```
<SETVAR NAME="errno" VALUE="0"/>
<SETVAR NAME="teststr" VALUE="hello world"/>
<BEGINS STR="Variables.teststr" WHAT="hello"/>
<IF COND="Variables.errno=1">
  <THEN> yes </THEN>
  <ELSE> no </ELSE>
</IF>
```

See Also

[ENDS](#)

CALLAPPLOGIC (Deprecated)

As of Content Server version 4.0, replaced by [CATALOGMANAGER](#), [CONTENTSERVER \(Deprecated\)](#), or [TREEMANAGER](#).

This tag does not work properly in Content Server 4.0. (CALLAPPLOGIC requires Netscape Application Server 2.1, which Content Server 4.0 does not support.)

CALLELEMENT

Processes the content of an element.

Syntax

```
<CALLELEMENT NAME="ELEMENT NAME"
  [PACKEDARGS="PACKEDARG NAMES"]
  [SCOPED="SCOPING OPTION"/>]
  <ARGUMENT NAME="ARG NAME" VALUE="ARG VALUE"/>
</CALLELEMENT>
```

Parameters

NAME (required)

Name of the element to call.

PACKEDARGS (optional)

A name/value pair to pass to the called element. Pairs are in the format `x=y&a=b&c=d`. This is useful if a string in this form is retrieved from a database field. It is more efficient to pass individual parameters rather than constructing a string in this form to pass values to an element.

SCOPED (optional)

Valid options for this attribute are as follows:

- GLOBAL - the default behavior
- LOCAL - arguments passed in are restored (or destroyed) on return from the call.
- STACKED - the child only sees the explicit arguments passed in and can only return values in that same namespace.

Description

The CALLELEMENT tag processes the content of an element. The element must exist in the ElementCatalog.

Variables that exist prior to CALLELEMENT are available to tags within the called element. Variables set within the called element are available in the parent element after the CALLELEMENT tag. Variables can also be set using the ARGUMENT tag within the CALLELEMENT tag or by using the PACKEDARGS parameter.

Variables for the CALLELEMENT tag are subject to an order of precedence. The order of precedence is relative to the location of the variable as follows:

1. Variables set in arguments
2. Variables set in scripts
3. **resdetails1** and **resdetails2** of the ElementCatalog

The following are examples of how the order of precedence works:

- In the following element, the arg=3 variable is set in the **resdetails1** column in the ElementCatalog. The arg value with the highest order of precedence is value1.


```
<SETVAR NAME="arg" VALUE="value2"/>
...
<CALLELEMENT NAME="element name">
  <ARGUMENT NAME="arg" VALUE="value1"/>
</CALLELEMENT>
```
- In the following element, the arg=value3 is set in the **redetails1** column in the ElementCatalog. The arg value with the highest order of precedence is value2.


```
<SETVAR NAME="arg" VALUE="value2"/>
...
<CALLELEMENT NAME="element name">
</CALLELEMENT>
```
- In the following element, the arg=value3 is set in the **resdetails1** column of the ElementCatalog. The arg value with the highest order of precedence is value3; this is occurs when a variable is not defined in scope before the CALLELEMENT tag.


```
<CALLELEMENT NAME="element name">
</CALLELEMENT>
```

errno

The possible values of errno include:

Value	Description
-10	No element exists.

Value	Description
-12	Cannot evaluate element.

Example

This example calls the element `SimpleTest` with the arguments `Foo` and `Foo2`.

```
<SETVAR NAME="errno" VALUE="0"/>
<CALLELEMENT NAME="SimpleTest">
  <ARGUMENT NAME="Foo" VALUE="Bar"/>
  <ARGUMENT NAME="Foo2" VALUE="Bar2"/>
</CALLELEMENT>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>
```

You can specify `PACKEDARGS` in the following format:

```
<SETVAR NAME="errno" VALUE="0"/>
<SETVAR NAME="args" VALUE="Foo=Bar&Foo2=Bar2"/>
<CALLELEMENT NAME="SimpleTest"
  PACKEDARGS="Variables.args">
</CALLELEMENT>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>
```

The parameters `Foo` and `Foo2` with values `Bar` and `Bar2` are available to the element `SimpleTest`.

See Also

[ARGUMENT](#)

CALLJAVA

Calls a Java class.

Syntax

```
<CALLJAVA CLASS="JAVA_CLASS">
  <ARGUMENT NAME="ARG_NAME" VALUE="ARG_VALUE"/>
</CALLJAVA>
```

Parameter

CLASS (required)

Name of Java class to call.

Description

The CALLJAVA tag calls a Java class. The Java class must exist in the system's Java classpath and it must implement the `COM.FutureTense.XML.Template.Seed` interface. Use the ARGUMENT tag to pass arguments to the called Java class.

errno

The possible values of errno include:

Value	Description
-900	Java class not found.
-901	No access to class.
-902	No instance.
-903	Execute error.
-904	Class cast error.

Example

This example calls a compliant Java class, `timeexample`. The Java class sees `logindate` as an incoming parameter.

```
<SETVAR NAME="errno" VALUE="0"/>
<CALLJAVA CLASS="timeexample">
  <ARGUMENT NAME="logindate"
    VALUE="SessionVariables.logintime"/>
</CALLJAVA>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!-- Handle error -->
  </THEN>
</IF>
```

See Also

[ARGUMENT](#)

CALLSQL

Retrieves and executes an SQL query stored in the `SystemSQL` table.

Syntax

```
<CALLSQL
  QRYNAME="QRY_NAME"
  LIST="LIST"
  [LIMIT="MAX_NO_RESULTS"] />
```

Parameters

QRYNAME (required)

Name of query to retrieve and execute.

LIST (required)

Name of list to contain resultset.

LIMIT (optional)

Maximum number of rows in the resultset. Note that there is no limit if the parameters are missing or set to a negative number.

Description

The **CALLSQL** tag retrieves and executes an SQL query stored in the **SystemSQL** table. The resultset of the query is cached against the table name specified for the query in the **SystemSQL** table. The table name must exist in the database prior to calling **CALLSQL**.

Note

The **deftable** column in the **SystemSQL** table can specify a comma-separated list of table names.

errno

The possible values of **errno** include:

Value	Description
-15	No file found.
-101	No rows found.
-102	Empty column.
-103	No such table.
-104	No table definition.
-106	Bad parameter.

Example

This example executes a query called `gettables` and stores the resultset in `newrs`. The `tblname` and `defdir` for each row in `newrs` are displayed.

```
<SETVAR NAME="errno" VALUE="0"/>
<CALLSQL QRYNAME="gettables" LIST="newrs"/>
<LOOP LIST="news">
    Tablename: <CSVAR NAME="newrs.tblname"/>
    Directory: <CSVAR NAME="newrs.defdir"/>
</LOOP>
<IF COND="IsError.Variables.errno=true">
    <THEN>
        <!--Handle error-->
    </THEN>
</IF>
```

See Also

[EXECSQL](#)
[RENAMELIST](#)
[SELECTTO](#)
[ICS.LITERAL](#)

CATALOGDEF

Queries a table for its column definition and stores the results in a list.

Syntax

```
<CATALOGDEF LIST="LIST_NAME"
            FROM="CATALOG_NAME"/>
```

Parameters

LIST (required)
 List to contain column information for the table.

FROM (required)
 Name of the table.

Description

The `CATALOGDEF` tag queries a table for its column definition and stores the results in a list.

The list contains a row for each column in the table. The `KEY` and `FOREIGN` values are identical for each row in the resultset. The column names are as follows:

Column Name	Meaning
KEY	Name of table's primary key (unknown if table is foreign).
FOREIGN	If table is foreign, value is "true". Otherwise, "false".
COLNAME	Name of the column in the table.

Column Name	Meaning
COLTYPE	The values for COLTYPE can be: text, integer, double binary, date/time, upload, or unknown.
COLSIZE	Size of the column based on database information.

errno

The possible values of errno include:

Value	Description
-16	Empty List
-101	No rows found.
-102	Empty column.
-103	No such table.
-104	No table definition.

Example

This example displays the table definition of a table called Images:

```
<SETVAR NAME = "errno" VALUE = "0"/>
<CATALOGDEF LIST="table" FROM="movies"/>
  Key: <CSVAR NAME="table.KEY"/><br/>
  Foreign: <CSVAR NAME="table.FOREIGN"/><br/>
<RENAMELIST LIST="tbldef" FROM="table" />
<LOOP LIST="tbldef">
  Colname: <CSVAR NAME="tbldef.COLNAME"/>
  <br/>
  Coltype: <CSVAR NAME="tbldef.COLTYPE"/>
  <br/>
  Colsize: <CSVAR NAME="tbldef.COLSIZE"/>
  <br/>
</LOOP>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>
```

CATALOGMANAGER

Creates, modifies, or deletes rows and tables (known previously as a "catalogs" in earlier versions of the product).

Using LOCAL or STACKED for the SCOPED argument protects those variables from change, but disallow sharing of input space. Use <CGIARGUMENT NAME="name"/> to specify that specific CGI parameters be made available to the CatalogManager. Use <ARGUMENT> as always for specific string values.

By omitting SCOPED (or setting it to "local") you can cause the CatalogManager to share all the existing input variables without scoping. Note that this means specified arguments overwrite existing variables. Using SCOPED="shared" means that you need not specify any argument; CGIARGUMENTS, specifically, is ignored. Note that variables set since program execution began are reset to the original values passed on the URL line. Use with care.

Syntax

```
<CATALOGMANAGER>
  [<ARGUMENT NAME="ftcmd" VALUE="some value"/>]
  [<CGIARGUMENT NAME="Some CGI arg" VALUE="Some CGI value"/>]
</CATALOGMANAGER>
```

Parameters

ARGUMENT (optional)

Arguments passed to the CatalogManager as it is called.

CGIARGUMENT (optional)

CGI arguments passed to the CatalogManager.

SCOPED (optional)

Valid options for this attribute are as follows:

- GLOBAL - the default behavior. Does not protect input variables.
- LOCAL - arguments passed in are restored (or destroyed) on return from the call.
- STACKED - the child only sees the explicit arguments passed in and can only return values in that same namespace.

Note

HTTP headers are not preserved when the SCOPED parameter is set to STACKED. To preserve HTTP header variables, you must specify them within an argument.

For example:

```
<ARGUMENT NAME="HTTPS" VALUE="Variables.HTTPS"/>
```

and

```
<ARGUMENT NAME="HTTPS_HOST" VALUE="Variables.HTTPS_HOST"/>
```

Description

The <CATALOGMANAGER> tag invokes the CatalogManager module. The CATALOGMANAGER module creates, modifies, or deletes tables and rows. It is an alternative to SQL-based tags, such as <CALLSQL>.

The following is a list of the commands that you use with the CATALOGMANAGER tag.

addrow	addrows	addrowtext	addUser (Deprecated)
commit	createtable	deletelog	deleterevision
deleterow	deleterows	delUser (Deprecated)	deletetable
editrow (Deprecated)	editrows (Deprecated)	exportform	exportlog
flushcatalog	flushpage	lock	login
logout	mirrorabort	mirrorrows	release
replacerow	replacerows	rollback	selectrows
tracktable	unlockrecord	untracktable	updaterow
updaterows	updaterow2	updaterows2	

Using LOCAL or STACKED for the SCOPED argument protects those variables from change, but disallows sharing of input space. Use <CGIARGUMENT NAME="name"/> to specify that specific CGI parameters be made available to the CatalogManager. Use <ARGUMENT> as always for specific string values.

By omitting SCOPED (or setting it to "local") you can cause the CatalogManager to share all the existing input variables without scoping. Note that this means specified arguments overwrite existing variables. Using SCOPED="shared" means that you need not specify any argument; CGIARGUMENTS, specifically, is ignored. Note that variables set since program execution began are reset to the original values passed on the URL line. Use with care.

errno

The value of errno is unchanged.

Example

```
<SETVAR NAME="errno" VALUE="0"/>
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="deleterow"/>
  <ARGUMENT NAME="tablename" VALUE="Movies"/>
  <ARGUMENT NAME="contentname" VALUE="Ishtar"/>
  <!-- Be sure review upload files are deleted-->
  <ARGUMENT NAME="Delete uploaded file(s)" VALUE="yes"/>
</CATALOGMANAGER>
<!-- Check status -->
<ICS.ARGUMENT COND="Variables.errno=0">
```

```

<IF>
  <THEN>
    Deleted movie:
    <CSVAR NAME="Variables.contentname"/>
    <br/>
  </THEN>
</IF>

```

See Also

[CALLAPPLAGIC \(Deprecated\)](#)
[CONTENTSERVER \(Deprecated\)](#)
[DEBABLE](#)
[TREEMANAGER](#)

addrow

Adds a row to a table using the CATALOGMANAGER command.

Syntax

```

<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="addrow"/>
  <ARGUMENT NAME="tablename" VALUE="table"/>
  [<ARGUMENT NAME="urlcolumnname_folder" VALUE="folder name"/>]
  [<ARGUMENT NAME="columnname_file" VALUE="file name"/>]
  [<ARGUMENT NAME="columnname" VALUE="some column name"/>]
</CATALOGMANAGER>

```

Parameters

ftcmd (required)

Value must be set to addrow.

tablename (required)

The name of the table where you plan to add a row.

urlcolumnname_folder (optional)

A subfolder name to store the uploaded file. The file is then stored under the upload folder and the subfolder. The upload folder is specified in the SystemInfo table's defdir column. Note that you can have multiple upload columns.

Upload columns are designated by the prefix url. The column name in the table must begin with url (for example, urltext). The argument name must include the table's column name with _folder appended to the column name (for example, urltext_folder).

columnname_file (required for non-binary files)

The name of the file you want to upload. The column name in the table must begin with url (for example, urltext). The argument name must have _file appended to the table's column name (for example, urltext_file).

columnname (optional)

The value of each column in the row to be added. The columnname is a table column name.

Description

The `addrow` command adds a row to a table.

Note

When adding a row to a table of type `obj` (an object table), the system creates `Variables.newid`, which contains the system-assigned unique ID for the new record.

errno

The possible values of `errno` include:

Value	Description
-103	No such table.
-104	No table definition.
-105	Database error.

Example

The following example adds a new row to the table named `movies`.

```
<SETVAR NAME="errno" VALUE="0"/>
<CATALOGMANAGER SCOPED="GLOBAL">
  <ARGUMENT NAME="ftcmd" VALUE="addrow" />
  <ARGUMENT NAME="tablename" VALUE="movies" />
  <ARGUMENT NAME="Reviewer" VALUE="John Doe" />
</CATALOGMANAGER>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>
```

The next example uses a table called `CatalogExample`. The default upload column is `/export/home/temp`. The table has the following columns:

- `id` (table's primary key)
- `urlText`
- `description`
- `quantity`

The form uses the `CatalogManager` command. The row is added to the table when the user clicks the `addrow` button and the file, `AddrowExample.txt` is added to the `/export/home/temp/CatalogForm/`.

```

<FORM NAME="f1"
  ACTION="CatalogManager"
  METHOD="post"
  <INPUT TYPE="hidden"
    NAME="tablename"
    VALUE="CatalogExample"/>
  KEY: <INPUT TYPE="text" NAME="id" VALUE="2"/>
  <BR/>
  URL: <INPUT TYPE="text" NAME="urlText"
    VALUE="this text will be put in a file
called AddrowExample.txt"/>
  <BR/>
  DESC: <INPUT TYPE="text" NAME="description"
    VALUE="example record"/>

  <BR/>
  QUANTITY: <INPUT TYPE="text" NAME="quantity" VALUE="1"/><BR/>
  FILE: <INPUT TYPE="text" NAME="urlText_file"
    VALUE="AddrowExample.txt"/>

  <BR/>
  FOLDER: <INPUT TYPE="text" NAME="urlText_folder"
    VALUE="CatalogForm"/>

  <BR/>
  FTCMD: <INPUT TYPE="submit" NAME="ftcmd" VALUE="addrow"/>
  <BR/>
</FORM>

```

The next example uses the same table, CatalogExample. This example is similar to the previous example, except that the user browses and selects a file to upload. In addition, the columnname_file parameter is not included in this example because the file is uploaded in binary mode.

```

<FORM NAME="f1"
  ENCTYPE="multipart/form-data"
  ACTION="CatalogManager"
  METHOD="post"
  <INPUT TYPE="hidden" NAME="tablename" VALUE="CatalogExample"/>
  KEY: <INPUT TYPE="text" NAME="id" VALUE="3"/>
  <BR/>
  DESC: <INPUT TYPE="text" NAME="description"
    VALUE="second record"/>

  <BR/>
  QUANTITY: <INPUT TYPE="text" NAME="quantity" VALUE="1"/>
  <BR/>
  FILE: <INPUT TYPE="file" NAME="urlText" VALUE=""/>
  <BR/>
  FOLDER: <INPUT TYPE="text" NAME="urlText_folder"
    VALUE="CatalogForm"/>

  <BR/>
  <INPUT TYPE="hidden" NAME="ftcmd" VALUE="addrow"/>
  <INPUT TYPE="submit" VALUE="Add"/>
</FORM>

```

See Also

[addrow](#)
[addrowtext](#)

addrows

Adds multiple rows to a table using the CATLOGMANGER command.

Syntax

```
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="addrows"/>
  <ARGUMENT NAME="tablename" VALUE="table"/>
  [<ARGUMENT NAME="column_folderN" VALUE="folder name"/>]
  [<ARGUMENT NAME="columnnameN_file" VALUE="file name"/>]
  [<ARGUMENT NAME="columnnameN" VALUE="column name"/>]
</CATALOGMANAGER>
```

Parameters

ftcmd (required)

Value must be set to addrows.

tablename (required)

Table name to add rows.

columnname_folderN (optional)

A subfolder name to store the uploaded file. The file is then stored under the upload folder and the subfolder. The upload folder is specified in the SystemInfo table's `defdir` column. Note that you can have multiple upload columns.

Upload columns are designated by the prefix `url`. The column name in the table must begin with `url` (for example, `urltext`). The argument name must include the table's column name with `_folder` appended to the column name (for example, `urltext_folder`).

columnname_fileN (required for non-binary files)

The name of the file you want to upload. The column name in the table must begin with `url` (for example, `urltext`). The argument name must have `_file` appended to the table's column name (for example, `urltext_file`).

columnnameN (required for non-binary files)

The name of the file you want to upload. The `columnname` is a table column name. The column name in the table must begin with `url` (for example, `urltext`). The argument name must include the table's column name with `_folder` appended to the column name (for example, `urltext_folder`).

errno

The possible values of `errno` include:

Value	Description
-103	No such table.
-104	No table definition.
-105	Database error.

Example

The following example uses an HTML form to add two new rows to the table named movies.

```
<SETVAR NAME="errno" VALUE="0" />
<CATALOGMANAGER SCOPED="GLOBAL">
  <ARGUMENT NAME="ftcmd" VALUE="addrows" />
  <ARGUMENT NAME="tablename" VALUE="movies" />
  <ARGUMENT NAME="urlcolumnname_folder0" VALUE="urlreview" />
  <ARGUMENT NAME="columnname0" VALUE="Reviewer" />
  <ARGUMENT NAME="urlcolumnname_folder1" VALUE="urlrating" />
  <ARGUMENT NAME="columnname1" VALUE="Rating" />
</CATALOGMANAGER>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>
```

The next example modifies a table called CatalogExample. The default upload column is /export/home/temp. The CatalogExample table has the following columns:

- id (the table's primary key)
- urlText
- description
- quantity

The sample form uses a CatalogManager command to add rows to the table when the user clicks the addrow button. The text files are uploaded to the upload folder and specified subfolder on the host machine.

The first records text file, FirstAddrows.txt, is put in the /export/home/temp/CatalogForm directory; the second records text file is put in the /export/home/temp/CatalogForm/Part2 directory.

```
<SETVAR NAME="desc1" VALUE="first record added"/>
<FORM NAME="f1" ACTION="CatalogManager"
      METHOD="post"
  <INPUT TYPE="hidden" NAME="tablename" VALUE="CatalogExample"/>
  <INPUT TYPE="hidden" NAME="id0" VALUE="3"/> <br/>
  FIRST URL: <INPUT TYPE="text" NAME="urlText0"
              VALUE="this text will be put in a
file called FirstAddrows.txt"/>
  <BR/>
  FIRST DESC: <INPUT TYPE="text" NAME="description0"
                  VALUE="Variables.desc1"
                  REPLACEALL="Variables.desc1"/>
  <BR/>
  FIRST QUANTITY: <INPUT TYPE="text" NAME="quantity0" VALUE="1"/>
  <BR/>
  FIRST FILE: <INPUT TYPE="text" NAME="urlText0_file"
                  VALUE="FirstAddrows.txt"/> <BR/>
  FIRST FOLDER: <INPUT TYPE="text" NAME="urlText_folder0"
```



```

                                VALUE="CatalogForm"/> <BR/>
<INPUT TYPE="hidden" NAME="id1"
                                VALUE="4"/>
    <BR/>
    SECOND URL: <INPUT TYPE="text" NAME="urlText1"
                                VALUE="The second file will be
put in a file called SecondAddrows.txt"/>
    <BR/>
    SECOND DESC: <INPUT TYPE="text" NAME="description1"
                                VALUE="second record added"/>
    <BR/>
    SECOND QUANTITY: <INPUT TYPE="text" NAME="quantity1"
                                VALUE="12"/>
    <BR/>
    SECOND FILE: <INPUT TYPE="text" NAME="urlText1_file"
                                VALUE="SecondAddrows.txt"/>
    <BR/>
    SECOND FOLDER: <INPUT TYPE="text" NAME="urlText_folder1"
                                VALUE="CatalogForm/Part2"/>
    <BR/>
    <INPUT TYPE="submit" NAME="ftcmd"
                                VALUE="addrows"/>
    <BR/>
</FORM>

```

See Also

[addrow](#)
[addrowtext](#)

addrowtext

Adds a row to a table using the CATALOGMANAGER command.

Syntax

```

<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="addrowtext"/>
  <ARGUMENT NAME="tablename" VALUE="table"/>
  [<ARGUMENT NAME="urlcolumnname_folder" VALUE="folder name"/>]
  [<ARGUMENT NAME="columnname_file" VALUE="file name"/>]
  [<ARGUMENT NAME="columnname" VALUE="column name"/>]
</CATALOGMANAGER>

```

Parameters

ftcmd (required)

Value must be set to addrowtext.

tablename (required)

Name of the table to which to add text.

urlcolumnname_folder (optional)

For every file upload column (column name starts with url), you can use a parameter named `urlcolumnname_folder` to store the file in a subfolder under the default upload folder for the table.

columnname_file (optional)

Used to specify a name for a file that is created on the server. If not provided, the system generates a file name and gives it the extension `.txt`.

columnname (optional)

Value of each column in the row to be added. The `columnname` is a table column name.

Description

The `addrowtext` command adds a row to a table. If the table contains an upload column, instead of requiring a file upload, text can be uploaded and Content Server creates a file to store the uploaded text.

errno

The possible values of `errno` include:

Value	Description
-103	No such table.
-104	No table definition.
-105	Database error.

Example

This example adds a row to the `reviews` table.

```
<SETVAR NAME="errno" VALUE="0" />
<CATALOGMANAGER SCOPED="GLOBAL">
  <ARGUMENT NAME="ftcmd" VALUE="addrowtext" />
  <ARGUMENT NAME="tablename"
    VALUE="reviews" />
  <ARGUMENT NAME="urlcolumnname_folder"
    VALUE="urlreview" />
  <ARGUMENT NAME="columnname_file"
    VALUE="Review" />
  <ARGUMENT NAME="columnname" VALUE="Reviewer" />
</CATALOGMANAGER>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>
```

See Also

The following `CATALOGMANAGER` commands:

[addrow](#)
[addrowtext](#)

addUser (Deprecated)

As of Content Server version 4.0, replaced by [DIR.CREATE](#).

Syntax

```
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="addUser" />
  <ARGUMENT NAME="username" VALUE="name" />
  <ARGUMENT NAME="password" VALUE="password"/>
  <ARGUMENT NAME="useracl" VALUE="acl"/>
</CATALOGMANAGER>
```

Parameters

ftcmd (required)

Value must be set to addUser.

username (required)

Name of user to add.

password (required)

Password for the user. The password is stored in an encrypted format.

useracl (required)

Access Control List. Comma-separated list of groups.

errno

The possible values of errno include:

Value	Description
-105	Database error.

Example

This example adds the user Editor, whose password is edit1234, and gives that user ContentEditor privileges. These values can be modified when the form is displayed.

```
<SETVAR NAME="errno" VALUE="0"/>
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="addUser"/>
  <ARGUMENT NAME="username" VALUE="Editor" />
  <ARGUMENT NAME="password" VALUE="edit1234" />
  <ARGUMENT NAME="useracl" VALUE="ContentEditor" />
</CATALOGMANAGER>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>
```

See Also

The following CATALOGMANAGER commands:

[delUser \(Deprecated\)](#)

[modifyUser \(Deprecated\)](#)

commit

Commits a record from a tracked table into the RevisionTracking subsystem using the CATALOGMANAGER command.

Syntax

```
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="commit" />
  <ARGUMENT NAME="tablename" VALUE="table" />
  <ARGUMENT NAME="asset" VALUE="primarykey"/>
  [<ARGUMENT NAME="checkout" VALUE="true or false"/>]
  [<ARGUMENT NAME="annotation" VALUE="comment"/>]
</CATALOGMANAGER>
```

Parameters

ftcmd (required)

Value must be set to `commit`.

tablename (required)

Name of the table.

asset (required)

Value of the primary key for the row to commit.

checkout (optional)

Indicates lock status of record after commit. Set value to `true` to indicate the record are still locked by user after the commit.

annotation (optional)

Comment describing the new revision.

Description

The `commit` command commits a record from a tracked table into the RevisionTracking subsystem. The current user must have the record locked and have write access to the tracked table to perform this operation.

errno

The possible values of `errno` include:

Value	Description
-103	No such table.
-104	No table definition.
-105	Database error.

Example

This example commits the locked record whose primary key value is 95137.

```
<SETVAR NAME="errno" VALUE="0"/>
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="commit"/>
  <ARGUMENT NAME="tablename" VALUE="movies" />
  <ARGUMENT NAME="asset" VALUE="95137" />
  <ARGUMENT NAME="annotation" VALUE="fix typo"/>
</CATALOGMANAGER>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>
```

See Also

The following CATALOGMANAGER commands:

[lock](#)
[unlockrecord](#)

createtable

Creates a table using the CATALOGMANAGER command.

Syntax

```
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="createtable" />
  <ARGUMENT NAME="tablename" VALUE="table" />
  <ARGUMENT NAME="systable" VALUE="no/obj"/>
  <ARGUMENT NAME="uploadDir" VALUE="directory" />
  <ARGUMENT NAME="colnameN" VALUE="name" />
  <ARGUMENT NAME="colvalueN" VALUE="value"/>
</CATALOGMANAGER>
```

Parameters

ftcmd (required)

Value must be set to createtable.

tablename (required)

Name of the table to create.

systable (required)

Allows differentiation between ContentCatalog (systable=no) and ObjectCatalog (systable=obj). This defaults to ContentCatalog (systable=no).

uploadDir (required)

Server-side folder path to contain uploaded files.

aclList (optional)

Comma-separated list of ACLs. When security is enabled, only members of the ACLs can access content in the table.

colvalueN (required)

Database-specific field type for column. The value of N is 0 (zero) for the first column, 1 for the second column, and so on.

colnameN (required)

Column field name. Each colname parameter corresponds to a column in the table. The value of N is 0 (zero) for the first column, 1 for the second column, and so on.

Description

The `createtable` command creates a table. The table's primary key field name is defined in the Content Server properties file (`futuretense.ini`).

Note

The default primary key used by CatalogManager is defined in the property `cc.contentkey`. Unlike a custom property for primary keys, the `cc.contentkey` property does not restrict the name of the tables that you create. The Object table column, version, is reserved for future use.

errno

The possible values of `errno` include:

Value	Description
-22	Table already exists.
-105	Database error.

Example

The following example uses the `cc.contentkey` property when creating a new table called `ExampleTable`:

```
<SETVAR NAME="errno" VALUE="0" />
  <CATALOGMANAGER>
    <ARGUMENT NAME="ftcmd" VALUE="createtable" />
    <ARGUMENT NAME="tablename" VALUE="ExampleTable" />
    <ARGUMENT NAME="systable" VALUE="no" />
    <ARGUMENT NAME="uploadDir" VALUE="/export/home/public" />
    <ARGUMENT NAME="colname0" VALUE="CS.Property.cc.contentkey"/>
    <ARGUMENT NAME="colvalue0" VALUE="varchar(10) PRIMARY KEY NOT
NULL" />
    <ARGUMENT NAME="colname1" VALUE="description" />
    <ARGUMENT NAME="colvalue1" VALUE="varchar(24)" />
    <ARGUMENT NAME="colname2" VALUE="quantity" />
    <ARGUMENT NAME="colvalue2" VALUE="CS.Property.cc.integer"/>
  </CATALOGMANAGER>
```

To specify the primary key in the properties file (`futuretense.ini` file), you must use the following syntax:

tablenameKey

Note

The K in Key must be capitalized; for example:

`cc.TestKey = testname`

The tablename must also be named Test, otherwise the table name is not created.

The following example shows a custom primary key using `cc.TestKey`:

```
<SETVAR NAME="makekey"
      VALUE="CS.Property.CC.Primary"/>
<SETVAR NAME="keytype"
      VALUE="CS.Property.CC.bigint"/>
<SETVAR NAME="errno" VALUE="0" />
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="createtable" />
    <ARGUMENT NAME="tablename" VALUE="Test" />
    <ARGUMENT NAME="uploadDir"
      VALUE="CS.Property.ft.home" />
    <ARGUMENT NAME="colname0" VALUE="CS.Property.cc.TestKey" />
    <ARGUMENT NAME="colvalue0" VALUE="Variables.keytype
Variables.makekey" />
    <ARGUMENT NAME="colname1" VALUE="fruit" />
    <ARGUMENT NAME="colvalue1" VALUE="CS.Property.cc.varchar(55)" /
  >
    <ARGUMENT NAME="colname2" VALUE="quantity" />
    <ARGUMENT NAME="colvalue2"
      VALUE="CS.Property.cc.smallint" />
</CATALOGMANAGER>
<SETVAR NAME="table1" VALUE="errno:Variables.errno" />
error creating table <CSVAR NAME="Variables.table1" />
<BR/>
```

The following example creates the table named `moviereviews`.

```
<SETVAR NAME="errno" VALUE="0" />
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="createtable" />
  <ARGUMENT NAME="tablename" VALUE="moviereviews" />
  <ARGUMENT NAME="systable" VALUE="no" />
  <ARGUMENT NAME="uploadDir" VALUE="c:\moviereviews" />
  <ARGUMENT NAME="aclList" VALUE="Browser" />
  <ARGUMENT NAME="colname0" VALUE="Reviewer" />
  <ARGUMENT NAME="colvalue0"
    VALUE="varchar(40)" />
  <ARGUMENT NAME="colname1" VALUE="urlreview" />
  <ARGUMENT NAME="colvalue1" VALUE="varchar(255) NULL"/>
</CATALOGMANAGER>
<IF COND="IsError.Variables.errno=true">
  <THEN>
```

The following example creates the table named `moviereviews`.

```
<SETVAR NAME="errno" VALUE="0" />
```

```

<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="createtable" />
  <ARGUMENT NAME="tablename" VALUE="moviereviews" />
  <ARGUMENT NAME="systable" VALUE="no" />
  <ARGUMENT NAME="uploadDir" VALUE="c:\moviereviews" />
  <ARGUMENT NAME="aclList" VALUE="Browser" />
  <ARGUMENT NAME="colname0" VALUE="Reviewer" />
  <ARGUMENT NAME="colvalue0"
    VALUE="varchar(40)" />
  <ARGUMENT NAME="colname1" VALUE="urlreview" />
  <ARGUMENT NAME="colvalue1" VALUE="varchar(255) NULL"/>
</CATALOGMANAGER>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <
      ...
    >
  <
    ...
  >

```

See Also

The following CATALOGMANAGER command:

[deletetable](#)

deletelog

Deletes the log file using the CATALOGMANAGER command (futuretenseip_address.txt) if per-IP address logging is enabled.

Syntax

```

<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="deletelog" />
</CATALOGMANAGER>

```

Parameter

ftcmd (required)

Value must be set to deletelog.

Description

The deletelog command deletes the log file (futuretenseip_address.txt) if per-IP address logging is enabled. Enable per-IP address logging by setting the ft.dbl property, found in the Content Server properties file (futuretense.ini), to yes.

This command is useful for removing the existing log file so that you can create a new one when debugging. When the application server restarts and Content Server is called the first time, the ContentServer servlet attempts to delete all previous log files, regardless of whether per-IP address logging is enabled. This means that the Content Server log file is deleted as well as any other .TXT files in the directory where the logs are stored.

errno

The possible values of errno include:

Value	Description
-300	File not found.

Example

This example deletes the log via a URL.

```
<SETVAR NAME="makekey"
      VALUE="CS.Property.CC.Primary"/>
<SETVAR NAME="keytype"
      VALUE="CS.Property.CC.bigint"/>
<SETVAR NAME="errno" VALUE="0" />
<SETVAR NAME="errno" VALUE="0" />
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="deletelog" />
</CATALOGMANAGER>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>
```

See Also

The following CATALOGMANAGER command:

[exportlog](#)

deleterevision

Deletes a revision of a row from a tracked table using the CATALOGMANAGER command.

Syntax

```
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="deleterevision" />
  <ARGUMENT NAME="tablename" VALUE="table" />
  <ARGUMENT NAME="asset" VALUE="primarykey"/>
  [<ARGUMENT NAME="version" VALUE="first, last, #, or date"/>]
</CATALOGMANAGER>
```

Parameters

ftcmd (required)

Value must be set to deleterevision.

tablename (required)

Name of the tracked table.

asset (required)

Value of the primary key for the row to obtain history.

version (optional)

Possible values are:

- **first** - history of earliest revision of record.
- **last** - history of most recent revision of record.
- **#** - history of a specific revision number.
- **date** - history of the record to whatever revision was current at the specified date. The date must be in SQL format (*yyyy-mm-dd hh:mm:ss*) and is expected to be GMT (Greenwich Mean Time).

Description

The `deleterevision` command deletes a revision of a row from a tracked table. The current user must have `rtadmin` access to the tracked table to perform this operation.

errno

The possible values of `errno` include:

Value	Description
-103	No such table.
-104	No table definition.
-105	Database error.

Example

This example deletes version 5 of the row in `movies` whose primary key value is 95137.

```
<SETVAR NAME="errno" VALUE="0"/>
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="deleterevision"/>
  <ARGUMENT NAME="tablename" VALUE="movies" />
  <ARGUMENT NAME="asset" VALUE="95137" />
  <ARGUMENT NAME="version" VALUE="5"/>
</CATALOGMANAGER>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!-- Handle error -->
  </THEN>
</IF>
```

See Also

The following CATALOGMANAGER command:

[release](#)

deleterow

Deletes a row in a table using the CATALOGMANAGER command..

Syntax

```
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="deleterow" />
  <ARGUMENT NAME="tablename" VALUE="table" />
  [<ARGUMENT NAME="Delete uploaded file(s)" VALUE="yes|no"/>]
  [<ARGUMENT NAME="primarykey" VALUE="column name"/>]
  [<ARGUMENT NAME="tablekey" VALUE="column name"/>]
  [<ARGUMENT NAME="tablekeyvalue" VALUE="some value"/>]
</CATALOGMANAGER>
```

Parameters

ftcmd (required)

Value must be set to `deleterow`.

tablename (required)

Name of the table containing the row to delete.

Delete uploaded file(s) (optional)

Value should be set to `yes` if upload files associated with row should be deleted.
Default is `no`.

primarykey (optional)

`primarykey` is the column name of the primary key column. The value is the value of row's primary key.

tablekey (optional)

Use the `tablekey` parameter if you want to delete rows based on a value other than the primary key. The value of `tablekey` is the name of the column you use to perform the deletion. For example, if you want to delete a table based on the value in the `title` column, the `tablekey` parameter looks as follows:

```
inList.setValString("tablekey", "title");
```

tablekeyvalue (optional)

Value in the `tablekey` column of the row you want to delete. For example, if you want to delete a row with a value of `Jaws` in the `title` column, the `tablekeyvalue` parameter looks as follows:

```
inList.setValString("tablekeyvalue", "Jaws");
```

This parameter is required only if the `tablekey` parameter is used.

Description

The `deleterow` command deletes a row in a table. There are two options that you can use to delete a row with this command:

- To delete rows using the primary key, set the following parameters:
 - `ftcmd`
 - `tablename`
 - `primarykey`
 - If the `primarykey` is `id`, you must have a variable called `id set`.

- To delete one or more rows that match some "alternate" column value, set the following parameters:
 - `ftcmd`
 - `tablename`
 - `tablekey`
 - `tablekeyvalue`

Note

This is valid only if the primary key column does not have a variable set.

errno

The possible values of `errno` include:

Value	Description
-105	Database error.

Example

The following examples remove two rows from the `Movie` table.

id (primary key)	rating	title
1	G	Beauty and The Beast
2	PG	The Princess Bride
3	R	The Godfather

Example using the primary key column to delete a row:

```
<SETVAR NAME="errno" VALUE="0"/>
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="deleterow"/>
  <ARGUMENT NAME="tablename" VALUE="movie"/>
  <ARGUMENT NAME="id" VALUE="1"/>
</CATALOGMANAGER>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>
```

Example using a column other than the primary key to delete a row:

```
<SETVAR NAME="errno" VALUE="0"/>
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="deleterow"/>
  <ARGUMENT NAME="tablename" VALUE="movie"/>
  <ARGUMENT NAME="tablekey" VALUE="title"/>
```

```

        <ARGUMENT NAME="tablekeyvalue" VALUE="The Princess Bride"/>
    </CATALOGMANAGER>
    <IF COND="IsError.Variables.errno=true">
        <THEN>
            <!--Handle error-->
        </THEN>
    </IF>

```

After both rows have been removed, the table looks as follows:

id (primary key)	rating	title
3	R	The Godfather

See Also

[addrow](#)
[deleterows](#)

deleterows

Deletes multiple rows in a table using the CATALOGMANAGER command.

Syntax

```

<CATALOGMANAGER>
    <ARGUMENT NAME="ftcmd" VALUE="deleterows" />
    <ARGUMENT NAME="tablename" VALUE="table" />
    <ARGUMENT NAME="primarykeyN" VALUE="keyvalue"/>
    [<ARGUMENT NAME="Delete uploaded file(s)" VALUE="yes|no"/>]
</CATALOGMANAGER>

```

Parameters

ftcmd (required)

Values must be set to deleterows.

tablename (required)

Name of the table containing the rows to delete.

primarykeyN (required)

Value of row's primary key. The **primarykey** parameter is the primary key's column name. The value of N corresponds to the row number—in the resultset—that you want to delete.

Delete uploaded file(s)(optional)

Value should be set to yes if upload files associated with row should be deleted. Default is no.

Description

The deleterows command deletes rows in a table. You can also delete uploaded files.

errno

The possible values of errno include:

Value	Description
-105	Database error.

Example

The following example deletes two rows from the TopMovies table:

id	director	title
133	charlie chaplin	modern times
137	buster keaton	the general
145	clint eastwood	a perfect world
148	rob reiner	the princess bride

To delete two rows from this table, use the following code:

```

<SETVAR NAME="errno" VALUE="0"/>
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="deleterows"/>
  <ARGUMENT NAME="tablename" VALUE="TopMovies"/>
  <ARGUMENT NAME="id0" VALUE="137"/>
  <ARGUMENT NAME="id1" VALUE="145"/>
</CATALOGMANAGER>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>

```

See Also

[deleterow](#)

deletetable

Deletes a table using the CATALOGMANAGER command.

Syntax

```
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="deletetable" />
  <ARGUMENT NAME="tablename" VALUE="table" />
</CATALOGMANAGER>
```

Parameters

ftcmd (required)
Specify the literal value `deletetable`.

tablename (required)
Specify the name of the table to delete.

Description

The `deletetable` command deletes a table. This command does not delete the upload storage folder.

errno

The possible values of `errno` include:

Value	Description
-105	Database error.

Example

This example deletes the table named `moviereviews`.

```
<SETVAR NAME="errno" VALUE="0"/>
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="deletetable" />
  <ARGUMENT NAME="tablename" VALUE="moviereviews" />
</CATALOGMANAGER>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>
```

See Also

The following CATALOGMANAGER command:

[createtable](#)

delUser (Deprecated)

As of Content Server version 4.0, replaced by [DIR.DELETE](#).

Syntax

```
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="delUser" />
  <ARGUMENT NAME="username" VALUE="name" />
</CATALOGMANAGER>
```

Parameters

ftcmd (required)
Value must be set to delUser.

username (required)
Name of the user to delete.

errno

The possible values of errno include:

Value	Description
-105	Database error.

Example

The following example deletes the user joe.

```
<SETVAR NAME="errno" VALUE="0"/>
<CATALOGMANAGER>
  <ARGUMENT name="ftcmd" value="delUser"/>
  <ARGUMENT NAME="username" VALUE="joe" />
</CATALOGMANAGER>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>
```

See Also

The following CATALOGMANAGER command:

[addUser \(Deprecated\)](#)

editrow (Deprecated)

As of Content Server version 4.0, replaced by [updaterow](#).

Note

errno is not set. No error is generated if the value for a particular column is invalid (for example, enter a text character into an integer type field). Valid column edits succeed, invalid column edits do not. FatWire recommends that you check your input carefully.

Syntax

```
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="editrow" />
  <ARGUMENT NAME="tablename" VALUE="table" />
  <ARGUMENT NAME="primarykey" VALUE="key" />
  [<ARGUMENT NAME="urlcolumnname_folder" VALUE="folder name"/>]
  [<ARGUMENT NAME="columnname_file" VALUE="file name"/>]
  [<ARGUMENT NAME="columnname" VALUE="column name"/>]
  [<ARGUMENT NAME="tablekey" VALUE="column name"/>]
  [<ARGUMENT NAME="tablekeyvalue" VALUE="some value"/>]
</CATALOGMANAGER>
```

Parameters

ftcmd (required)

Value must be set to "editrow".

tablename (required)

Name of the table whose row is edited.

primarykey (required)

Value of a row's primary key. The primaryKey parameter is the primary key's column name.

urlcolumnname_folder (optional)

A subfolder name to store the uploaded file. The file is then stored under the upload folder and the subfolder. The upload folder is specified in the SystemInfo table's defdir column. Note that you can have multiple upload columns.

Upload columns are designated by the prefix url. The column name in the table must begin with url (for example, urltext). The argument name must include the table's column name with _folder appended to the column name (for example, urltext_folder).

columnname_file (required for non-binary files)

The name of the file you want to upload. The column name in the table must begin with url (for example, urltext). The argument name must have _file appended to the table's column name (for example, urltext_file).

columnname (optional)

The information in the field you are editing. The columnName parameter is the name of the column to be edited.

tablekey (optional)

Use the `tablekey` parameter if you want to update rows based on a value other than the primary key. The value of `tablekey` is the column name of the column you use to perform the update. For example, if you want to update a table based on the value in the `title` column, the `tablekey` parameter looks as follows:

```
inList.setValString("tablekey", "title");
```

tablekeyvalue (optional)

Value in the `tablekey` column of the row you want to update. For example, if you want to update a row with a value of `Jaws` in the `title` column, the `tablekeyvalue` parameter looks as follows:

```
inList.setValString("tablekeyvalue", "Jaws");
```

This parameter is required only if the `tablekey` parameter is used.

Description

The `editrow` command modifies the fields of a row in a table. The fields in the row are modified with the new values. If a URL field is edited, the content of the associated server-side upload file is modified to contain the new value. Thus, a user can pass text to update the back-end storage for a given URL column value.

errno

The possible values of `errno` include:

Value	Description
-103	No such table.
-104	No table definition.
-105	Database error.

Example

This example edits a movie description in the `movies` table.

```
<SETVAR NAME="errno" VALUE="0" />
<CATALOGMANAGER SCOPED="GLOBAL">
  <ARGUMENT NAME="ftcmd" VALUE="editrow" />
  <ARGUMENT NAME="tablename" VALUE="movies" />
  <ARGUMENT NAME="id" VALUE="1234" />
  <ARGUMENT NAME="title" VALUE="Godzilla" />
  <ARGUMENT NAME="comments" VALUE="miss it" />
</CATALOGMANAGER>
<IF COND="Variables.errno=0">
  <THEN>
    Edit successful.
  </THEN>
  <ELSE>
    <CSVAR NAME="Edit failed.See errno Variables.errno for more
information." />
  </ELSE>
</IF>
```

See Also

[editrow](#) (Deprecated)
[replacerow](#)
[updaterow](#)
[addrowtext](#)

editrows (Deprecated)

As of Content Server version 4.0, replaced by [updaterows](#).

Note

errno is not set. No error is generated if the value for a particular column is invalid (for example, enter a text character into an integer type field). Valid column edits succeed, invalid column edits do not. FatWire recommends that you check your input carefully.

Syntax

```
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="editrows" />
  <ARGUMENT NAME="tablename" VALUE="table" />
  <ARGUMENT NAME="primarykey" VALUE="key"/>
  [<ARGUMENT NAME="primarykeyN" VALUE="column name"/>]
  [<ARGUMENT NAME="columnnameN" VALUE="column name"/>]
  [<ARGUMENT NAME="urlcolumnname_folder" VALUE="folder name"/>]
  [<ARGUMENT NAME="columnname_file" VALUE="file name"/>]
</CATALOGMANAGER>
```

Parameters

ftcmd (required)

Value must be set to editrows.

tablename (required)

Name of the table containing rows to be edited.

primarykeynameN (optional)

The primarykeyname parameter is the is the name of the column you want to use as the primary key. The value of N corresponds to the row number to be edited. For example, if you want to edit the values in row 0 of the table, you use the following code:

```
inList.setValString("id0", "1");
```

columnnameN (optional)

The columnname parameter is the name of the column that you want to edit. The value of N corresponds to the row number—in the resultset—that you want to edit. For example, if you want to edit the value in the comments field of row 0 of the table to read “miss it”, you use the following code:

```
inList.setValString("comments0", "miss it");
```

urlcolumnname_folder (optional)

A subfolder name to store the uploaded file. The file is then stored under the upload folder and the subfolder. The upload folder is specified in the SystemInfo table's defdir column. Note that you can have multiple upload columns.

Upload columns are designated by the prefix url. The column name in the table must begin with url (for example, urltext). The argument name must include the table's column name with _folder appended to the column name (for example, urltext_folder).

columnname_file (required for non-binary files)

The name of the file you want to upload. The column name in the table must begin with url (for example, urltext). The argument name must have _file appended to the table's column name (for example, urltext_file).

Description

The editrows command modifies the fields for multiple rows in a table. The fields in the row are modified with the new values. If a url field is edited, the content of the associated server-side upload file is modified to contain the new value.

errno

The possible values of errno include:

Value	Description
-103	No such table.
-104	No table definition.
-105	Database error.

Example

This example edits the rows with godzilla and titanic in the movies table, and changes the values of the comments column to miss it and see it, respectively.

```
<SETVAR NAME="errno" VALUE="0" />
<CATALOGMANAGER SCOPED="GLOBAL">
  <ARGUMENT NAME="ftcmd" VALUE="editrows" />
  <ARGUMENT NAME="tablename" VALUE="movies" />
  <ARGUMENT NAME="id0" VALUE="1234" />
  <ARGUMENT NAME="title0" VALUE="Godzilla" />
  <ARGUMENT NAME="comments0" VALUE="miss it" />
  <ARGUMENT NAME="id1" VALUE="1235" />
  <ARGUMENT NAME="title1" VALUE="Titanic" />
  <ARGUMENT NAME="comments1" VALUE="see it" />
</CATALOGMANAGER>
<IF COND="Variables.errno=0">
  <THEN>
    Edit successful.
  </THEN>
  <ELSE>
    <CSVAR NAME="Edit failed.See errno Variables.errno for more
information."/>
```

```
</ELSE>
</IF>
```

See Also

[editrow \(Deprecated\)](#)
[replaceroes](#)
[updaterows](#)

exportform

Displays a default HTML form for adding, deleting, modifying, or selecting rows from a table using the CATALOGMANAGER command.

Syntax

```
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="exportform" />
  <ARGUMENT NAME="tablename" VALUE="table" />
  <ARGUMENT NAME="ftchoice" VALUE="command name"/>
  [<ARGUMENT NAME="tablekeyvalue" VALUE="primary key"/>]
</CATALOGMANAGER>
```

Parameters

ftcmd (required)

Value must be set to `exportform`.

tablename (required)

Name of the table that the form uses.

ftchoice (required)

Value must be set to one of the following CATALOGMANAGER commands: `addrow`, `deleterow`, `updaterow`, `selectrows`, or `replaceroes`.

tablekeyvalue (optional)

If **ftchoice** is `deleterow`, `editrow (Deprecated)`, or `updaterow`, the value for this input must be the primary key value for the record that you want to delete or update. No **tablekeyvalue** can be supplied when **ftchoice** is set to `addrow`.

errno

The possible values of **errno** include:

Value	Description
-105	Database error.

Example

The `updaterow`, `deleterow`, and `replaceroes` operations require that the specified row be selected by the value of its primary key. This example assumes that the column name `id` is the primary key for any **tablename** specified.

```
<SETVAR NAME="errno" VALUE="0"/>
<CATALOGMANAGER>
```

```

    <ARGUMENT NAME="ftcmd" VALUE="exportform" />
    <ARGUMENT NAME="tablename" VALUE="SystemInfo" />
    <ARGUMENT NAME="tablekeyvalue" VALUE="id"/>
    <ARGUMENT NAME="ftchoice" VALUE="updaterow"/>
</CATALOGMANAGER>
<IF COND="IsError.Variables.errno=true">
    <THEN>
        <!--Handle error-->
    </THEN>
</IF>

```

See Also

[addrow](#)
[deleterow](#)
[updaterow](#)
[selectrows](#)
[replacerow](#)

exportlog

Exports a per-IP address log using the CATALOGMANAGER command.

Syntax

```

<CATALOGMANAGER>
    <ARGUMENT NAME="ftcmd" VALUE="exportlog" />
</CATALOGMANAGER>

```

Parameter

ftcmd (required)
 Value must be set to `exportlog`.

Description

The `exportlog` command exports a per-IP address log. You can enable per-IP address logging by setting the `ft.dbl` property, located in the Content Server properties file (`futuretense.ini`), to `yes`.

errno

The possible values of `errno` include:

Value	Description
-300	File not found.

Example

Although `export log` could be called from inside an element, it is more commonly called as an argument to a URL. If your application server is WebLogic or WebSphere, the following URL exports the log:

```
http://host:port/servlet/CatalogManager?ftcmd=export log
```

If your application server is iPlanet, the following URL exports the log:

```
http://host:port/NASapp/cs/CatalogManager?ftcmd=export log
```

See Also

[deletelog](#)

flushcatalog

Flushes the internal memory cache for a table using the CATALOGMANAGER command.

Syntax

```
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="flushcatalog" />
  <ARGUMENT NAME="tablename" VALUE="table" />
</CATALOGMANAGER>
```

Parameters

`ftcmd` (required)

Value must be set to `flushcatalog`.

`tablename` (required)

Name of the table to flush.

Description

The `flushcatalog` command flushes the internal memory cache for a table. This command is useful to clear a table's cache after a table update using a non-Content Server mechanism.

`flushcatalog` can also force cache clearing when the SQL used to select against a table has performed a join with other tables; the system may hold the cached information for either table.

errno

The possible values of `errno` include:

Value	Description
-105	Database error.

Example

If your application server is WebLogic or WebSphere, the following URL flushes the cache for the SystemInfo table.

```
http://host:port/servlet/
CatalogManager?ftcmd=flushcatalog&tablename=SystemInfo
```

If your application server is iPlanet, the following URL flushes the cache for the SystemInfo table.

```
http://host:port/NASapp/cs/
CatalogManager?ftcmd=flushcatalog&tablename=SystemInfo
```

flushpage

Deletes a page's cache file using the CATALOGMANAGER command.

Syntax

```
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="flushpage" />
  <ARGUMENT NAME="pagename" VALUE="name" />
</CATALOGMANAGER>
```

Parameters

ftcmd (required)

Value must be set to flushpage.

pagename (required)

Name of page to be cleared from cache.

Notes

The flushpage tag does not return an error in either of the following situations:

- The specified pagename does not exist.
- A page with the supplied criteria does not exist.

As of CSEE 5.0., the params parameter can no longer be used.

Example

This example show how to call and flush the page named arts2 from the disk cache. Also notice that in this example the value custom1id and custom2id are contained on the resargs column in the SiteCatalog.

To call a page:

```
<SETVAR NAME="errno" VALUE="0"/>
<CONTENTSERVER>
  <ARGUMENT NAME="pagename" VALUE="arts2"/>
  <ARGUMENT NAME="custom1id" VALUE="pass1"/>
  <ARGUMENT NAME="custom2id" VALUE="show2"/>
</CONTENTSERVER>
```



```

<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>

To flush a page:
<SETVAR NAME="errno" VALUE="0"/>
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="flushpage"/>
  <ARGUMENT NAME="pagename" VALUE="arts2"/>
</CATALOGMANAGER>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>

```

history

Gets a revision history for rows in a revision-tracked table using the CATALOGMANAGER command.

Syntax

```

<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="history" />
  <ARGUMENT NAME="tablename" VALUE="table" />
  <ARGUMENT NAME="asset" VALUE="primarykey"/>
  [<ARGUMENT NAME="version" VALUE="version data"/>]
  [<ARGUMENT NAME="locker" VALUE="locker data"/>]
  [<ARGUMENT NAME="creator" VALUE="creator data"/>]
  [<ARGUMENT NAME="annotation" VALUE="row annotation"/>]
</CATALOGMANAGER>

```

Parameters

ftcmd (required)

Value must be set to history.

tablename (required)

Name of the tracked table.

asset (required)

Value of the primary key for the row to obtain history.

version (optional)

Possible values are:

first - history of earliest revision of record.

last - history of most recent revision of record.

- history of a specific revision number.

date - history of the record to whichever revision was current at the specified date.

The date must be in SQL format (yyyy-mm-dd hh:mm:ss) and is expected to be GMT.

locker (optional)

Filter rows returned based on a user who has rows locked. If the current user does not have `rtaudit` privileges, you must specify a value for `locker`, `creator`, or both. If you specify only `locker`, the value must be the current user's name; otherwise, no data is returned.

creator (optional)

Filter rows returned based on the user who created the row. If the current user does not have `rtaudit` privileges, you must specify a value for `locker`, `creator`, or both. If you specify only `creator`, the value must be the current user's name; otherwise, no data is returned.

annotation (optional)

Filter rows returned based on the annotation of a row.

Description

The `history` command gets a revision history for rows in a revision-tracked table.

To obtain history for all rows, the current user must have `rtaudit` privileges for the table. If the current user has read privileges on the table, only history for the revisions which the user has created or locked is returned. Specific versions can be specified by using the optional version parameter. Rows may be filtered using the optional parameters `locker` and `creator`. Use the optional parameter `annotation` to retrieve rows based on their annotation.

errno

The possible values of `errno` include:

Value	Description
-103	No such table.
-104	No table definition.
-105	Database error.

Example

This example gets the history for the row in `movies` whose primary key value is 95137.

```
<SETVAR NAME="errno" VALUE="0"/>
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="history"/>
  <ARGUMENT NAME="tablename" VALUE="movies" />
  <ARGUMENT NAME="asset" VALUE="95137" />
</CATALOGMANAGER>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>
```

lock

Locks a record for the current user in a tracked table using the `CATALOGMANAGER` command.

Syntax

```
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="lock" />
  <ARGUMENT NAME="tablename" VALUE="table" />
  <ARGUMENT NAME="asset" VALUE="primarykey"/>
</CATALOGMANAGER>
```

Parameters

`ftcmd` (required)

Value must be set to `lock`.

`tablename` (required)

Name of the tracked table.

`asset` (required)

Value of the primary key for the row to lock.

Description

The `lock` command locks a record for the current user in a tracked table. Once a row is locked, the current user can modify fields in the row. The current user must have write access to the table.

errno

The possible values of `errno` include:

Value	Description
-103	No such table.
-104	No table definition.
-105	Database error.

Example

This example locks the record whose primary key value is 95137 in movies.

```
<SETVAR NAME="errno" VALUE="0"/>
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="lock"/>
  <ARGUMENT NAME="tablename" VALUE="movies" />
  <ARGUMENT NAME="asset" VALUE="95137" />
</CATALOGMANAGER>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>
```

See Also

[release](#)

login

Authorizes a user within an application using the CATALOGMANAGER command.

Syntax

```
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="login" />
  <ARGUMENT NAME="username" VALUE="name" />
  <ARGUMENT NAME="password" VALUE="password"/>
</CATALOGMANAGER>
```

Parameters

ftcmd (required)
Value must be set to login.

username (required)
Name of existing user.

password (required)
Password of user.

errno

The possible values of errno include:

Value	Description
-105	Database error.

Example

This example logs in user Tom with password of oscar:

```
<SETVAR NAME="errno" VALUE="0"/>
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="login"/>
  <ARGUMENT NAME="username" VALUE="Tom"/>
  <ARGUMENT NAME="password" VALUE="oscar"/>
</CATALOGMANAGER>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>
```

See Also

[logout](#)

logout

Logs out a user from within an application using the CATALOGMANAGER command.

Syntax

```
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="logout"/>
  [<ARGUMENT NAME="killsession" VALUE="true or false"/>]
</CATALOGMANAGER>
```

Parameters

ftcmd (required)

Value must be set to `logout`.

killsession (optional)

If the value is set to `true`, it destroys the session maintained on the server. This should only be done at the end of processing a given HTTP request since you can't recover or re-create a session until the next HTTP connect issued by the client. This is useful for web users after they finish their session.

Description

This tag logs a user out from an application. Please note that as of Content Server version 4.0, the `CatalogManager logout` command is not valid when `CatalogManager security` is activated. Using the `CatalogManager logout` command with `CatalogManager security` is activated results in the session being destroyed and possible application errors.

errno

The possible values of `errno` include:

Value	Description
-105	Database error.

Example

This example logs out the logged in user.

```
<SETVAR NAME="errno" VALUE="0"/>
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="logout"/>
</CATALOGMANAGER>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>
```

See Also

[login](#)

mirrorabort

Aborts a mirror operation in progress using the CATALOGMANAGER command.

Note

errno is not set. No error is generated if the value for a particular column is invalid (for example, enter a text character into an integer type field). Valid column edits succeed, invalid column edits do not. FatWire recommends that you check your input carefully.

Syntax

```
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="mirrorabort" />
</CATALOGMANAGER>
```

Parameter

ftcmd (required)

Value must be set to mirrorabort.

Description

This tag aborts a mirror operation in progress. Note that when you use this command there can be a variety of errno values returned.

errno

The possible values of `errno` include:

Value	Description
0	Mirror operation succeeded.
-105	Database error.
-601	Returned from <code>mirrorabort</code> when a mirror operation is not currently in progress.
-602	Mirror operation in progress.
-603	Required variables are missing from the <code>mirrorrows CatalogManager</code> command.
-604	The data (file) associated with a <code>url</code> column could not be found on the source machine.
-605	The schema for a column on the source machine can not be determined.
-606	No rows to mirror from the target system.
-607	Current mirroring operation was aborted via the <code>mirrorabort</code> command.
-608	A connection could not be established to the mirror target server.
-609	The network disconnected communication between the source and target machines.
-610	An attempt was made to mirror from a machine where the database columns are mixed case to a system where the database columns are case-sensitive.
-611	Mirror commit operation failed.
-612	Target server returned an invalid configuration connection message.
-613	Obsolete; is not returned.
-614	Could not create a temporary table on the target server.
-615	Could not delete the temporary table.
-616	Server does not support this mirror operation.
-617	Table is not an Object table.
-618	No tree nodes found matching the <code>nid</code> parameter of <code>FTTTreeNode</code> .

Example

The following example aborts a mirror operation.

```
<SETVAR NAME="errno" VALUE="0"/>
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="mirrorabort" />
</CATALOGMANAGER>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>
```

See Also

[mirrorrows](#)

mirrorrows

Copies rows from a table in one database to a table in another database using the CATALOGMANAGER command.

Note

errno is not set. No error is generated if the value for a particular column is invalid (for example, enter a text character into an integer type field). Valid column edits succeed, invalid column edits do not. FatWire recommends that you check your input carefully.

Syntax

```
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="mirrorrows" />
  <ARGUMENT NAME="tablekey" VALUE="name" />
  <ARGUMENT NAME="tablekeyvalue" VALUE="value"/>
  [<ARGUMENT NAME="mirrorport" VALUE="target port"/>]
  <ARGUMENT NAME="mirrortarget" VALUE="target"/>
  [<ARGUMENT NAME="mirrorauthusername" VALUE="username"/>]
  [<ARGUMENT NAME="mirrorauthpassword" VALUE="password"/>]
  [<ARGUMENT NAME="useProxyServer" VALUE="true or false"/>]
  [<ARGUMENT NAME="cgiprefix" VALUE="some prefix"/>]
  [<ARGUMENT NAME="inimerge" VALUE="ini file"/>]
  [<ARGUMENT NAME="secureCommunication" VALUE="HTTPS"/>]
</CATALOGMANAGER>
```

Parameters

ftcmd (required)

Value must be set to mirrorrows.

tablename (required)

Name of the source table.

tablekey (required)

Name of the table column to perform a selection on. If the system does not find the key, it uses the primary key of the specified table as the selection column. Contentname is the default value. See the Content Server properties file, futuretense.ini to change this default.

tablekeyvalue (required)

Value of tablekey.

mirrorport (optional)

HTTP target port. Port 80 is used if a value is not specified.

mirrortarget (required)

Network name of the mirror target machine.

mirrorauthusername (optional)

Content Server user name to log in to the target machine. Value is required if security is enabled on the mirror target.

mirrorauthpassword (optional)

Password associated with the value of mirrorauthusername to log in to the mirror target machine. Value is required if security is enabled on the mirror target.

useProxyServer (optional)

If the value is true, transmission should be routed through the proxy server. The proxy server and port to use are specified using the cs.mirrorproxyserver and cs.mirrorproxyport properties.

cgiprefix (optional)

Specifies the application server type on the target system. For iPlanet Application Server, set it to:

/NASApp/CS/

For WebLogic and WebSphere, set it to:

/servlet

inimerge (optional)

Name of a property file on the target system that should be loaded before the mirroring operation proceeds. This parameter is used to define primary key values for tables on the target server.

secureCommunication (optional)

If true, uses secure protocol (HTTPS) to transmit the data to the target server.

errno

The possible values of errno include:

Value	Description
0	Mirror operation succeeded.
-105	Database error.
-601	Returned from mirrorabort when a mirror operation is not currently in progress.
-602	Mirror operation in progress.
-603	Required variables are missing from the mirrorrows CatalogManager command.

Value	Description
-604	The data (file) associated with a url column could not be found on the source machine.
-605	The schema for a column on the source machine cannot be determined.
-606	No rows to mirror from the target system.
-607	Current mirroring operation was aborted via the <code>mirrorabort</code> command.
-608	A connection could not be established to the mirror target server.
-609	The network disconnected communication between the source and target machines.
-610	An attempt was made to mirror from a machine where the database columns are mixed case to a system where the database columns are case-sensitive.
-611	Mirror commit operation failed.
-612	Target server returned an invalid configuration connection message.
-613	Obsolete; is not returned.
-614	Could not create a temporary table on the target server.
-615	Could not delete the temporary table.
-616	Server does not support this mirror operation.
-617	Table is not an Object table.
-618	No tree nodes found matching the <code>nid</code> parameter of <code>FTTreeNode</code> .

Example

This example passes mirror parameters that a user has entered in an HTML form to the `mirror` command.

```
<SETVAR NAME="errno" VALUE="0" />
<CATALOGMANAGER SCOPED="GLOBAL">
  <ARGUMENT NAME="ftcmd" VALUE="mirrorrows" />
  <ARGUMENT NAME="tablename" VALUE="movies" />
  <ARGUMENT NAME="tablekey" VALUE="id" />
  <ARGUMENT NAME="tablekeyvalue" VALUE="your_server" />
  <ARGUMENT NAME="mirrortarget" VALUE="" />
  <ARGUMENT NAME="mirrorauthusername" VALUE="ContentServer" />
  <ARGUMENT NAME="mirrorauthpassword" VALUE="FutureTense" />
</CATALOGMANAGER>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>
```

See Also

[mirrorabort](#)

modifyUser (Deprecated)

As of Content Server version 4.0, replaced by [DIR.REPLACEATTRS](#).

Syntax

```
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="modifyUser" />
  <ARGUMENT NAME="username" VALUE="name" />
  [<ARGUMENT NAME="password" VALUE="password"/>]
  [<ARGUMENT NAME="useracl" VALUE="user's acl"/>]
</CATALOGMANAGER>
```

Parameters

ftcmd (required)

Value must be set to `modifyUser`.

username (required)

Name of the user to modify.

password (optional)

Password for the user. The password is stored in an encrypted format.

useracl (optional)

Access Control List. Comma-separated list of groups.

errno

The possible values of `errno` include:

Value	Description
-105	Database error.

Example

The following example modifies the user Joe.

```
<SERVAR NAME="errno" VALUE="0"/>
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="modifyUser" />
  <ARGUMENT NAME="username" VALUE="Joe"/>
  <ARGUMENT NAME="password" VALUE="fred" />
  <ARGUMENT NAME="useracl" VALUE="ContentEditor" /><br/>
</CATALOGMANAGER>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>
```

See Also

[addUser \(Deprecated\)](#)

[delUser \(Deprecated\)](#)

release

Releases a lock on a row from a tracked table using the CATALOGMANAGER command.

Syntax

```
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="release" />
  <ARGUMENT NAME="tablename" VALUE="table" />
  <ARGUMENT NAME="asset" VALUE="primarykey"/>
</CATALOGMANAGER>
```

Parameters

ftcmd (required)

Value must be set to release.

tablename (required)

Name of the tracked table.

asset (required)

Value of the primary key for the row to lock.

Description

The release command releases a lock on a row from a tracked table. Only the user who has the row locked against the table can release it. This also reverts the row in the table to its last committed state, overwriting any changes the user made while the row was locked.

errno

The possible values of errno include:

Value	Description
-103	No such table.
-104	No table definition.
-105	Database error.

Example

This example releases the locked record whose primary key value is 95137 in the table movies.

```
<SETVAR NAME="errno" VALUE="0"/>
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="release"/>
  <ARGUMENT NAME="tablename" VALUE="movies"/>
  <ARGUMENT NAME="asset" VALUE="95137"/>
</CATALOGMANAGER>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>
```

See Also

[lock](#)

replacerow

Replaces an existing row in a table on a remote server by first deleting it, and then inserting the specified information into the row using the CATALOGMANAGER command.

Syntax

```
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="replacerow" />
  <ARGUMENT NAME="tablename" VALUE="table" />
  [<ARGUMENT NAME="columnname" VALUE="column name"/>]
  [<ARGUMENT NAME="urlcolumnname_folder" VALUE="folder name"/>]
  [<ARGUMENT NAME="columnname_file" VALUE="file name"/>]
  [<ARGUMENT NAME="tablekey" VALUE="column name"/>]
  [<ARGUMENT NAME="tablekeyvalue" VALUE="some value"/>]
</CATALOGMANAGER>
```

Parameters

ftcmd (required)

Value must be set to `replacerow`.

tablename (required)

Name of the table that contains the row to replace.

primarykey (optional)

Value of row's primary key. The `primarykey` parameter is the primary key's column name.

columnname (optional)

Each column in the row to be replaced. `columnname` is a table column name.

urlcolumnname_folder (optional)

A subfolder name to store the uploaded file. The file is then stored under the upload folder and the subfolder. The upload folder is specified in the SystemInfo table's defdir column. Note that you can have multiple upload columns.

Upload columns are designated by the prefix url. The column name in the table must begin with url (for example, urltext). The argument name must include the table's column name with _folder appended to the column name (for example, urltext_folder).

columnname_file (required for non-binary files)

The name of the file you want to upload. The column name in the table must begin with url (for example, urltext). The argument name must have _file appended to the table's column name (for example, urltext_file).

tablekey (optional)

Use the tablekey parameter if you want to update rows based on a value other than the primary key. The value of tablekey is the column name of the column you use to perform the update. For example, if you want to update a table based on the value in the title column, the tablekey parameter looks as follows:

```
inList.setValString("tablekey", "title");
```

tablekeyvalue (optional)

Value in the tablekey column of the row you want to update. For example, if you want to update a row with a value of Jaws in the title column, the tablekeyvalue parameter looks as follows:

```
inList.setValString("tablekeyvalue", "Jaws");
```

This parameter is required only if the tablekey parameter is used.

Description

The replacerow command replaces an existing row in a table on a remote server by first deleting it, and then inserting the specified information into the row. This command should be used to clear the value of a column in a row.

Note

If the row specified by the primarykey value does not exist, then the row is added to the table.

errno

The possible values of errno include:

Value	Description
-103	No such table.
-104	No table definition.
-105	Database error.

Example

The following example replaces the existing row information for `titanic` with information for `The Great Escape`.

133	modern times	g	see it
137	the general	g	see it
145	a perfect world	pg	see it
148	the princess bride	pg	see it
138	titanic	pg 13	miss it

```
<SETVAR NAME="errno" VALUE="0" />
  <CATALOGMANAGER SCOPED="GLOBAL">
    <ARGUMENT NAME="ftcmd" VALUE="replacrow" />
    <ARGUMENT NAME="tablename" VALUE="moviereviews" />
    <ARGUMENT NAME="id" VALUE="138" />
    <ARGUMENT NAME="title" VALUE="The Great Escape" />
    <ARGUMENT NAME="rating" VALUE="PG" />
    <ARGUMENT NAME="comments" VALUE="see it" />
  </CATALOGMANAGER>

<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>
```

After the row has been replaced, the table looks as follows:

133	modern times	g	see it
137	the general	g	see it
145	a perfect world	pg	see it
148	the princess bride	pg	see it
138	The Great Escape	PG	see it

See Also

[editrow \(Deprecated\)](#)

[updaterow](#)

[updaterow2](#)

replacerows

Replaces multiple rows in a table using the CATALOGMANAGER command.

Note

errno is not set. No error is generated if the value for a particular column is invalid (for example, enter a text character into an integer type field). Valid column edits succeed, invalid column edits do not. FatWire recommends that you check your input carefully.

Syntax

```
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="replacerows"/>
  <ARGUMENT NAME="tablename" VALUE="table"/>
  [<ARGUMENT NAME="primarykeyN" VALUE="some value"/>]
  [<ARGUMENT NAME="columnnameN" VALUE="column name"/>]
  [<ARGUMENT NAME="urlcolumnname_folderN" VALUE="folder name"/>]
  [<ARGUMENT NAME="columnname_fileN" VALUE="file name"/>]
  [<ARGUMENT NAME="tablekeyN" VALUE="column name"/>]
  [<ARGUMENT NAME="tablekeyvalueN" VALUE="some value"/>]
</CATALOGMANAGER>
```

Parameters

ftcmd (required)

Value must be set to replacerows.

tablename (required)

Name of the table that contains the rows to replace.

primarykeyN (optional)

Value of row's primary key. The primaryKey parameter is the primary key's column name. The value of N corresponds to the row number—in the resultset—that you want to replace.

columnnameN (optional)

Each column in the row to be replaced. columnName is a table column name. The value of N corresponds to the row number—in the resultset—that you want to replace.

urlcolumnname_folderN (optional)

A subfolder name to store the uploaded file. The file is then stored under the upload folder and the subfolder. The upload folder is specified in the SystemInfo table's defdir column. Note that you can have multiple upload columns.

Upload columns are designated by the prefix url. The column name in the table must begin with url (for example, urltext). The argument name must include the table's column name with _folder appended to the column name (for example, urltext_folder).

columnname_fileN (required for non-binary files)

The name of the file you want to upload. The column name in the table must begin with url (for example, urltext). The argument name must have _file appended to the table's column name (for example, urltext_file).

tablekeyN (optional)

Use the **tablekeyN** parameter if you want to update rows based on a value other than the primary key. The value of **tablekeyN** is the column name of the column you use to perform the update. The value of **N** corresponds to the row number—in the resultset—that you want to update. For example, if you want to update a table based on the value in the **title** column, the **tablekeyN** parameter looks as follows:

```
inList.setValString("tablekey0", "title");
```

tablekeyvalueN (optional)

Value in the **tablekey** column of the row you want to update. The value of **N** corresponds to the row number—in the resultset—that you want to update. For example, if you want to update a row with a value of **Jaws** in the **title** column, the **tablekeyvalueN** parameter looks as follows:

```
inList.setValString("tablekeyvalue0", "Jaws");
```

This parameter is required only if the **tablekey** parameter is used.

Description

The **replaceroes** command replaces multiple rows in a table. If a value is not specified for a column, the column value is cleared.

Note

If the row specified by the primary key value does not exist, then the row is added to the table.

errno

The possible values of **errno** include:

Value	Description
-22	Table already exists.
-103	No such table.
-104	No table definition.
-105	Database error.

Example

The following example updates a row and adds a new row to the table named movies:

24	Something About Mary	R
25	The Godfather	PG

```
<SETVAR NAME="errno" VALUE="0" />
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="replacerows" />
  <ARGUMENT NAME="tablename" VALUE="movies" />
  <ARGUMENT NAME="id0" VALUE="26" />
  <ARGUMENT NAME="title0" VALUE="American Pie" />
  <ARGUMENT NAME="rating0" VALUE="R"/>
  <ARGUMENT NAME="id1" VALUE="25" />
  <ARGUMENT NAME="title1" VALUE="The Godfather"/>
</CATALOGMANAGER>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>
```

The new row is displayed in the following table, and the rating row of the GodFather is changed to null:

24	Something About Mary	R
25	The Godfather	
26	American Pie	R

See Also

[editrows \(Deprecated\)](#)
[updaterows](#)

rollback

Reverts a row in a tracked table to a previous revision using the CATALOGMANAGER command.

Syntax

```
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="rollback" />
  <ARGUMENT NAME="tablename" VALUE="table" />
  <ARGUMENT NAME="asset" VALUE="primarykey"/>
  [<ARGUMENT NAME="version" VALUE="first, last, # or date"/>]
</CATALOGMANAGER>
```

Parameters

ftcmd (required)

Value must be set to rollback.

tablename (required)

Name of the tracked table.

asset (required)

Value of the primary key for the row to roll back.

version (optional)

Possible values are:

- **first** - roll back to earliest revision of record.
- **last** - roll back to most recent revision of record.
- **#** - roll back to a specific revision number.
- **date** - roll back the record to whatever revision was current at the specified date. The date must be in SQL format (yyyy-mm-dd hh:mm:ss) and is expected to be GMT.

Description

The rollback command reverts a row in a tracked table to a previous revision. The current user must have the row locked and have `rtaudit` privileges against the table to roll back the row. A row can be rolled back to its earliest revision, its most recent revision, a specific version number, or whichever revision was current at a specified date.

errno

The possible values of `errno` include:

Value	Description
-103	No such table.
-104	No table definition.
-105	Database error.

Example

This example performs a rollback on the record whose primary key value is 95137 to version 5.

```
<SETVAR NAME="errno" VALUE="0"/>
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="rollback"/>
  <ARGUMENT NAME="tablename" VALUE="movies"/>
  <ARGUMENT NAME="asset" VALUE="95137"/>
  <ARGUMENT NAME="version" VALUE="5"/>
</CATALOGMANAGER>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>
```

selectrows

Executes a query against a given table and displays records from a table using the CATALOGMANAGER command.

Note

errno is not set. No error is generated if the value for a particular column is invalid (for example, enter a text character into an integer type field). Valid column edits succeed, invalid column edits do not. FatWire recommends that you check your input carefully.

Syntax

```
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="selectrow(s)" />
  <ARGUMENT NAME="tablename" VALUE="tablename" />
  [<ARGUMENT NAME="selwhere" VALUE="keycolumnname" />]
  [<ARGUMENT NAME="selwhat" VALUE="keycolumnvalue" />]
</CATALOGMANAGER>
```

Parameters

ftcmd (required)

Value must be set to selectrows.

tablename (required)

Name of the table to query.

selwhere (optional)

Lists the where clause components. If the system does not find any where clause components, it attempts to build a where clause using variables that correspond to column names.

selwhat (optional)

Lists columns to select from the table. The default is *, which selects all columns.

Description

The `selectrow(s)` command executes a query against a given table and displays records from a table. The rows displayed matches the criteria specified by the value of the parameters. CatalogManager assigns the returned HTML to the `cshttp` variable.

Example

This example selects all the columns in the Reviews table.

id (primary key)	rating	title
1	R	Reservoir Dogs
2	PG	The Ice Pirates
3	PG	Kellys Heroes

```
<SETVAR NAME="errno" VALUE="0" />
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="selectrow(s)"/>
  <ARGUMENT NAME="tablename" VALUE="Reviews"/>
</CATALOGMANAGER>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
  <ELSE>
    <CSVAR NAME="Variables.cshttp"/>
  </ELSE>
</IF>
```

This example selects the `rating` and `title` columns in the reviews table.

```
<SETVAR NAME="errno" VALUE="0" />
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="selectrow(s)"/>
  <ARGUMENT NAME="tablename" VALUE="Reviews"/>
  <ARGUMENT NAME="selwhere" VALUE="id=1"/>
  <ARGUMENT NAME="selwhat" VALUE="rating, title"/>
</CATALOGMANAGER>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
  <ELSE>
    <CSVAR NAME="Variables.cshttp"/>
  </ELSE>
</IF>
```

setversions

Sets the number of revisions stored for each row in a tracked table using the CATALOGMANAGER command.

Syntax

```
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="setversions" />
  <ARGUMENT NAME="tablename" VALUE="table" />
  <ARGUMENT NAME="numversions" VALUE="number"/>
</CATALOGMANAGER>
```

Parameters

ftcmd (required)

Value must be set to setversions.

tablename (required)

Name of the tracked table.

numversions (required)

Number of revisions to be stored for each row in a tracked table.

Description

The setversions command sets the number of revisions stored for each row in a tracked table. The current user must have rtadmin privileges on the table.

errno

The possible values of errno include:

Value	Description
-103	No such table.
-104	No table definition.
-105	Database error.

Example

This example sets the maximum number of revisions for rows in the movies table to 8.

```
<SETVAR NAME="errno" VALUE="0" />
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="setversions"/>
  <ARGUMENT NAME="tablename" VALUE="movies" />
  <ARGUMENT NAME="numversions" VALUE="8" />
</CATALOGMANAGER>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>
```

See Also

[tracktable](#)

tracktable

Enables revision tracking operations on a table using the CATALOGMANAGER command.

Syntax

```
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="tracktable" />
  <ARGUMENT NAME="tablename" VALUE="table" />
  [<ARGUMENT NAME="numversions" VALUE="some number"/>]
  [<ARGUMENT NAME="storage" VALUE="storage location"/>]
</CATALOGMANAGER>
```

Parameters

ftcmd (required)

Value must be set to `tracktable`.

tablename (required)

Name of the table to track.

numversions (optional)

Maximum number of versions to store for each row. When the maximum is reached, the oldest version is removed. If the value of `numversions` isn't specified or is set to 0, there is no limit to the number of revisions.

storage (optional)

Location to store revisions associated with a row in the table.

Description

The `tracktable` command enables revision tracking operations on a table. The maximum number of revisions per row can be set, as well as the storage location for the revisions. Only users who have `rtadmin` privileges against the table can track the table.

errno

The possible values of `errno` include:

Value	Description
-103	No such table.
-104	No table definition.
-105	Database error.

Example

This example tracks the reviews table with a maximum of 5 versions for each row.

```
<SETVAR NAME="errno" VALUE="0"/>
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="tracktable"/>
  <ARGUMENT NAME="tablename" VALUE="reviews"/>
  <ARGUMENT NAME="numversions" VALUE="5"/>
  <ARGUMENT NAME="storage" VALUE="c:\storage"/>
</CATALOGMANAGER>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>
```

See Also

[untracktable](#)

untracktable

Stops tracking a table using the CATALOGMANAGER command.

Syntax

```
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="untracktable" />
  <ARGUMENT NAME="tablename" VALUE="table" />
</CATALOGMANAGER>
```

Parameters

ftcmd (required)

Value must be set to untracktable.

tablename (required)

Name of the tracked table to untrack.

Description

The untracktable command stops tracking a table. Revision tracking operations are no longer be available on the table. Only users who have `rtadmin` privileges against the table can untrack the table.

errno

The possible values of `errno` include:

Value	Description
-103	No such table.
-104	No table definition.
-105	Database error.

Example

This example stops tracking on the reviews table.

```
<SETVAR NAME="errno" VALUE="0"/>
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="untracktable"/>
  <ARGUMENT NAME="tablename" VALUE="reviews" />
</CATALOGMANAGER>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>
```

See Also

The following CATALOGMANAGER command:

[tracktable](#)

unlockrecord

Unlocks a locked row using the CATALOGMANAGER command.

Syntax

```
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="unlockrecord" />
  <ARGUMENT NAME="tablename" VALUE="table" />
  <ARGUMENT NAME="asset" VALUE="primarykey"/>
</CATALOGMANAGER>
```

Parameters

ftcmd (required)

Value must be set to unlockrecord.

tablename (required)

Name of the tracked table.

asset (required)

Value of the primary key for the row to unlock.

Description

The unlockrecord command unlocks a locked row. Only the user who has the row locked or a user who has `rtadmin` privileges against the table can unlock the row. UNLOCKRECORD does not restore the row to its most recent revision.

errno

The possible values of errno include:

Value	Description
-103	No such table.
-104	No table definition.
-105	Database error.

Example

This example unlocks the row in `movies` whose primary key value is 95137.

```

<SETVAR NAME="errno" VALUE="0"/>
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="unlockrecord"/>
  <ARGUMENT NAME="tablename" VALUE="movies"/>
  <ARGUMENT NAME="asset" VALUE="95137"/>
</CATALOGMANAGER>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>

```

See Also

[lock](#)

updaterow

Updates field values for a row in a table using the CATALOGMANAGER command.

Note

errno is not set. No error is generated if the value for a particular column is invalid (for example, enter a text character into an integer type field). Valid column edits succeed, invalid column edits do not. FatWire recommends that you check your input carefully.

Syntax

```

<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="updaterow" />
  <ARGUMENT NAME="tablename" VALUE="table" />
  [<ARGUMENT NAME="primarykey" VALUE="primary key value"/>]
  [<ARGUMENT NAME="columnname" VALUE="column name"/>]
  [<ARGUMENT NAME="urlcolumnname_folder" VALUE="folder name"/>]
  [<ARGUMENT NAME="columnname_file" VALUE="file name"/>]
  [<ARGUMENT NAME="tablekey" VALUE="column name"/>]

```

```
[<ARGUMENT NAME="tablekeyvalue" VALUE="some value"/>]
</CATALOGMANAGER>
```

Parameters

ftcmd (required)

Value must be set to `updaterow`.

tablename (required)

Name of the table containing the row to be updated.

primarykey (optional)

Value of row's primary key. The `primarykey` parameter is the primary key's column name.

columnname (optional)

Each column in the row to be modified. `columnname` is a table column name.

urlcolumnname_folder (optional)

A subfolder name to store the uploaded file. The file is then stored under the upload folder and the subfolder. The upload folder is specified in the `SystemInfo` table's `defdir` column. Note that you can have multiple upload columns.

Upload columns are designated by the prefix `url`. The column name in the table must begin with `url` (for example, `urltext`). The argument name must include the table's column name with `_folder` appended to the column name (for example, `urltext_folder`).

columnname_file (required for non-binary files)

The name of the file you want to upload. The column name in the table must begin with `url` (for example, `urltext`). The argument name must have `_file` appended to the table's column name (for example, `urltext_file`).

tablekey (optional)

Use the `tablekey` parameter if you want to update rows based on a value other than the primary key. The value of `tablekey` is the column name of the column you use to perform the update. For example, if you want to update a table based on the value in the `title` column, the `tablekey` parameter looks as follows:

```
inList.setValString("tablekey", "title");
```

tablekeyvalue (optional)

Value in the `tablekey` column of the row you want to update. For example, if you want to update a row with a value of `Jaws` in the `title` column, the `tablekeyvalue` parameter looks as follows:

Description

The `updaterow` command updates field values for a row in a table. For each column and specified value, the existing column value is replaced with the incoming data for that column. Only values specified are changed; you cannot clear a value in a table row or column by using `editrows` or `updaterows`. You must use `replacerow` or `updaterow2`.

errno

The possible values of errno include:

Value	Description
-103	No such table.
-104	No table definition.
-105	Database error.

Example

The following example updates the description in the movies table for a given movie:

```
<SETVAR NAME="errno" VALUE="0"/>
<CATALOGMANAGER SCOPED="GLOBAL">
  <ARGUMENT NAME="ftcmd" VALUE="updaterow" />
  <ARGUMENT NAME="tablename" VALUE="movies" />
  <ARGUMENT NAME="id" VALUE="1234" />
  <ARGUMENT NAME="title" VALUE="Godzilla" />
  <ARGUMENT NAME="comments" VALUE="miss it" />
</CATALOGMANAGER>
<IF COND="Variables.errno=0">
  <THEN>
    Update successful.
  </THEN>
  <ELSE>
    <CSVAR NAME="Update failed. See errno Variables.errno for
more information."/>
  </ELSE>
</IF>
```

See Also

[editrow \(Deprecated\)](#)
[replacerow](#)
[updaterow2](#)

updaterow2

Updates or clears values in columns for a row in a table using the CATALOGMANAGER command.

Note

errno is not set. No error is generated if the value for a particular column is invalid (for example, enter a text character into an integer type field). Valid column edits succeed, invalid column edits do not. FatWire recommends that you check your input carefully.

Syntax

```
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="updaterow2" />
  <ARGUMENT NAME="tablename" VALUE="table" />
  [<ARGUMENT NAME="primarykey" VALUE="primary key value"/>]
  [<ARGUMENT NAME="columnname" VALUE="columnname"/>]
  [<ARGUMENT NAME="urlcolumnname_folder" VALUE="folder name"/>]
  [<ARGUMENT NAME="columnname_file" VALUE="file name"/>]
  [<ARGUMENT NAME="tablekey" VALUE="key value"/>]
  [<ARGUMENT NAME="tablekeyvalue" VALUE="some value"/>]
  [<ARGUMENT NAME="updaterowNullColumns" VALUE="column1,
    column2"/>]
</CATALOGMANAGER>
```

Parameters

ftcmd (required)

Value must be set to updaterow2.

tablename (required)

Name of the table containing the row to be updated.

primarykey (optional)

Value of row's primary key. The primarykey parameter name is the primary key's column name.

columnname (optional)

Each column in the row to be modified. columnname is a table column name.

urlcolumnname_folder (optional)

A subfolder name to store the uploaded file. The file is then stored under the upload folder and the subfolder. The upload folder is specified in the SystemInfo table's defdir column. Note that you can have multiple upload columns.

Upload columns are designated by the prefix url. The column name in the table must begin with url (for example, urltext). The argument name must include the table's column name with _folder appended to the column name (for example, urltext_folder).

columnname_file (required for non-binary files)

The name of the file you want to upload. The column name in the table must begin with url (for example, urltext). The argument name must have _file appended to the table's column name (for example, urltext_file).

tablekey (optional)

Use the `tablekey` parameter if you want to update rows based on a value other than the primary key. The value of `tablekey` is the column name of the column you use to perform the update. For example, if you want to update a table based on the value in the `title` column, the `tablekey` parameter looks as follows:

```
inList.setValString("tablekey", "title");
```

tablekeyvalue (optional)

Value in the `tablekey` column of the row you want to update.

updaterowNullColumns (optional)

A comma-separated list of columns that are to be cleared if no data is supplied when the user submits a form.

Note that if this list is supplied, other columns are ignored and are not cleared.

Description

Like `updaterow`, `updaterow2` updates values in columns for a row in a table; however, in contrast to `updaterow`, which does not allow you to clear columns, `updaterow2` allows you to clear columns if there is no value for the specified column (for example, if there is no related field in the form, or if the field on the form was left blank).

errno

The possible values of `errno` include:

Value	Description
-103	No such table.
-104	No table definition.
-105	Database error.

Example

This example uses a form to update the description and rating fields in the `movies` table for a given movie.

```
<SETVAR NAME="errno" VALUE="0" />
<CATALOGMANAGER SCOPED="GLOBAL">
  <ARGUMENT NAME="ftcmd" VALUE="updaterow2" />
  <ARGUMENT NAME="tablename" VALUE="movies" />
  <ARGUMENT NAME="id" VALUE="1234" />
  <ARGUMENT NAME="title" VALUE="Godzilla" />
  <ARGUMENT NAME="comments" VALUE="see it" />
</CATALOGMANAGER>

<IF COND="Variables.errno=0">
  <THEN>
    Update successful.
  </THEN>
  <ELSE>
    <CSVAR NAME="Update failed. See errno Variables.errno for
more information."/>
```

```
</ELSE>
</IF>
```

See Also

[editrow \(Deprecated\)](#)
[replacerow](#)

updaterows

Modifies field values for multiple rows in a table using the CATALOGMANAGER command.

Syntax

```
<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="updaterows" />
  <ARGUMENT NAME="tablename" VALUE="table" />
  [<ARGUMENT NAME="primarykeyN" VALUE="primary key value"/>]
  [<ARGUMENT NAME="columnnameN" VALUE="column name"/>]
  [<ARGUMENT NAME="urlcolumnname_folderN" VALUE="folder name"/>]
  [<ARGUMENT NAME="columnname_fileN" VALUE="file name"/>]
  [<ARGUMENT NAME="tablekeyN" VALUE="key value"/>]
  [<ARGUMENT NAME="tablekeyvalueN" VALUE="some value"/>]
</CATALOGMANAGER>
```

Parameters

ftcmd (required)

Value must be set to `updaterows`.

tablename (required)

Name of the table containing rows to be updated.

primarykeyN (optional)

Value of row's primary key. The `primarykey` parameter is the primary key's column name. The value of *N* corresponds to the row number—in the resultset—that you want to update.

columnnameN (optional)

Each column in the row to be updated. `columnname` is a table column name. The value of *N* corresponds to the row number—in the resultset—that you want to update.

urlcolumnname_folderN (optional)

A subfolder name to store the uploaded file. The file is then stored under the upload folder and the subfolder. The upload folder is specified in the `SystemInfo` table's `defdir` column. Note that you can have multiple upload columns.

Upload columns are designated by the prefix `url`. The column name in the table must begin with `url` (for example, `urltext`). The argument name must include the table's column name with `_folder` appended to the column name (for example, `urltext_folder`).

columnname_fileN (required for non-binary files)

The name of the file you want to upload. The column name in the table must begin with `url` (for example, `urltext`). The argument name must have `_file` appended to the table's column name (for example, `urltext_file`).

`tablekeyN` (optional)

Use the `tablekeyN` parameter if you want to update rows based on a value other than the primary key. The value of `tablekeyN` is the column name of the column you use to perform the update. The value of `N` corresponds to the row number—in the resultset—that you want to update.

`tablekeyvalueN` (optional)

Value in the `tablekey` column of the row you want to update. The value of `N` corresponds to the row number—in the resultset—that you want to update.

Description

The `updaterows` command modifies field values for multiple rows in a table. For fields other than url fields, the value of the field is modified. For url fields, `updaterows` deletes the file associated with the url field and modifies the url field to point to the newly uploaded file.

errno

The possible values of `errno` include:

Value	Description
-103	No such table.
-104	No table definition.
-105	Database error.

Example

This example updates the rows with `godzilla` and `titanic` in the `movies` table, and changes the values of the `comments` column to `miss it` and `see it`, respectively.

```
<SETVAR NAME="errno" VALUE="0"/>
<CATALOGMANAGER SCOPED="GLOBAL">
  <ARGUMENT NAME="ftcmd" VALUE="updaterows" />
  <ARGUMENT NAME="tablename" VALUE="movies" />
  <ARGUMENT NAME="id0" VALUE="1234" />
  <ARGUMENT NAME="title0" VALUE="godzilla" />
  <ARGUMENT NAME="comments0" VALUE="miss it" />
  <ARGUMENT NAME="id1" VALUE="1235" />
  <ARGUMENT NAME="title1" VALUE="titanic" />
  <ARGUMENT NAME="comments1" VALUE="see it" />
</CATALOGMANAGER>
<IF COND="Variables.errno=0">
  <THEN>
    Update successful.
  </THEN>
  <ELSE>
    <CSVAR NAME="Update failed. See errno Variables.errno for
more information."/>
  </ELSE>
</IF>
```


See Also

[addrows](#)
[editrows \(Deprecated\)](#)
[replacerows](#)
[updaterows](#)
[updaterows2](#)

updaterows2

Modifies field values or clears columns for multiple rows in a table using the CATALOGMANAGER command.

Syntax

```

<CATALOGMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="updaterows2" />
  <ARGUMENT NAME="tablename" VALUE="table" />
  <ARGUMENT NAME="primarykeyN" VALUE="key"/>
  <ARGUMENT NAME="ftcmd" VALUE="updaterows" />
  <ARGUMENT NAME="tablenameN" VALUE="table" />
  [<ARGUMENT NAME="primarykeyN" VALUE="primary key value"/>]
  [<ARGUMENT NAME="columnnameN" VALUE="columnname"/>]
  [<ARGUMENT NAME="urlcolumnname_folderN" VALUE="folder name"/>]
  [<ARGUMENT NAME="columnname_fileN" VALUE="columnname"/>]
  [<ARGUMENT NAME="tablekeyN" VALUE="key value"/>]
  [<ARGUMENT NAME="tablekeyvalueN" VALUE="some value"/>]
  [<ARGUMENT NAME="updaterowNullColumns" VALUE="column1,
    column2"/>]
</CATALOGMANAGER>

```

Parameters

ftcmd (required)

Value must be set to "updaterows2".

tablename (required)

Name of the table containing rows to be updated.

primarykeyN (required)

Value of row's primary key. The primarykey parameter is the primary key's column name. The value of N corresponds to the row number—in the resultset—that you want to update.

columnnameN (optional)

Each column in the row to be replaced. columnname is a table column name. The value of N corresponds to the row number—in the resultset—that you want to update.

urlcolumnname_folderN (optional)

A subfolder name to store the uploaded file. The file is then stored under the upload folder and the subfolder. The upload folder is specified in the SystemInfo table's defdir column. Note that you can have multiple upload columns.

Upload columns are designated by the prefix url. The column name in the table must begin with url (for example, urltext). The argument name must include the table's column name with _folder appended to the column name (for example, urltext_folder).

`columnname_fileN` (required for non-binary files)

The name of the file you want to upload. The column name in the table must begin with `url` (for example, `urltext`). The argument name must have `_file` appended to the table's column name (for example, `urltext_file`).

`tablekeyN` (optional)

Use the `tablekeyN` parameter if you want to update rows based on a value other than the primary key. The value of `tablekeyN` is the column name of the column you use to perform the update. The value of `N` corresponds to the row number—in the resultset—that you want to update.

`tablekeyvalueN` (optional)

Value in the `tablekey` column of the row you want to update. The value of `N` corresponds to the row number—in the resultset—that you want to update.

This parameter is required only if the `tablekey` parameter is used.

`updaterowNullColumns` (optional)

A comma-separated list of columns that are to be cleared if no data is supplied when the user submits a form. Note that If this list is supplied, other columns are ignored and are not cleared.

Description

Like `updaterows`, the `updaterow2` command modifies field values for multiple rows in a table; however, in contrast to `updaterows`, which does not allow you to clear columns, `updaterows2` allows you to clear columns if there is no value for the specified column (for example, if there is no related field in the form). For url fields, `updaterows2` deletes the file associated with the url field and modifies the url field to point to the newly uploaded file.

errno

The possible values of `errno` include:

Value	Description
-103	No such table.
-104	No table definition.
-105	Database error.

Example

This example updates the rows with `godzilla` and `titanic` in the `movies` table, and changes the values of the `comments` column to one star and three stars, respectively.

```
<SETVAR NAME="errno" VALUE="0" />
<CATALOGMANAGER SCOPED="GLOBAL">
  <ARGUMENT NAME="ftcmd" VALUE="updaterows2" />
  <ARGUMENT NAME="tablename" VALUE="movies" />
  <ARGUMENT NAME="id0" VALUE="1234" />
  <ARGUMENT NAME="title0" VALUE="godzilla" />
  <ARGUMENT NAME="comments0" VALUE="one star" />
  <ARGUMENT NAME="id1" VALUE="1235" />
```

```

    <ARGUMENT NAME="title1" VALUE="titanic" />
    <ARGUMENT NAME="comments1" VALUE="three stars" />
</CATALOGMANAGER>
<IF COND="Variables.errno=0">
    <THEN>
        Update successful.
    </THEN>
    <ELSE>
        <CSVAR NAME="Update failed. See errno Variables.errno for
more information."/>
    </ELSE>
</IF>

```

See Also

[addrows](#)
[editrows \(Deprecated\)](#)
[replacerows](#)
[updaterows](#)

COMMIT

Creates a new revision of a record from a tracked table.

Syntax

```

<COMMIT TABLE="TABLE_NAME"
    ASSET="PRIMARY_KEY"
    [KEEPLocked="KEEPLocked_VAL "]
    [ANNOTATION="COMMENT"/>]

```

Parameters

TABLE (required)

Name of the table

ASSET (required)

Value of the primary key for the record being committed.

KEEPLocked (optional)

Indicates lock status of record after commit. Set value to `true` to indicate the record is still locked by user after the commit.

ANNOTATION (optional)

Comment describing the new revision.

Description

The `commit` command creates a new revision of a record from a tracked table. The current user must have the record locked and have write access to the tracked table to perform this operation.

errno

The possible value of errno include:

Value	Description
-1000	Command failure. Check the <code>errdetail</code> variable for the specific error.

Example

The following example assumes all COMMIT parameters are exposed on an HTML form. It first checks whether the parameters exist, clears the value of `errno`, then commits the record.

```

<!-- Check whether required parameters exist-->
  <IF COND="IsVariable.tablename=false">
    <THEN>
      <SETVAR NAME="tablename" VALUE="" />
    </THEN>
  </IF>
  <IF COND="IsVariable.asset=false">
    <THEN>
      <SETVAR NAME="asset" VALUE="" />
    </THEN>
  </IF>
  <IF COND="IsVariable.annotation=false">
    <THEN>
      <SETVAR NAME="annotation" VALUE="" />
    </THEN>
  </IF>
  <IF COND="IsVariable.keeplocked=false">
    <THEN>
      <SETVAR NAME="keeplocked" VALUE="false" />
    </THEN>
  </IF>
<!-- clear out errno -->
<SETVAR NAME="errno" VALUE="0" />
<!-- do the commit -->
<COMMIT TABLE="Variables.tablename"
  ASSET="Variables.asset"
  KEELOCKED="Variables.keeplocked"
  ANNOTATION="Variables.annotation" />
<!-- see if the commit succeeded -->
  <IF COND="Variables.errno=0">
    <THEN>
      <!-- if succeeded, brag about it -->
      Commit succeeded for <CSVAR NAME="Variables.asset" />
    <BR />
    </THEN>
    <ELSE>
      <!-- otherwise, call an error reporting element -->
      Commit failed.<BR />
    </ELSE>
  </IF>

```

```
</ELSE>
</IF>
```

See Also

[DELETEREVISION](#)
[HISTORY](#)
[LOCK](#)
[RELEASE](#)
[RETRIEVEREVISION](#)
[ROLLBACK](#)
[SETVERSIONS](#)
[TRACKTABLE](#)
[UNLOCKRECORD](#)
[UNTRACKTABLE](#)

CONTENTSERVER (Deprecated)

As of Content Server version 4.0, replaced by [satellite.page](#).

By omitting SCOPED (or setting it to “local”) you can cause the ContentServer to share all the existing input variables without scoping. Note that this means specified arguments overwrite existing variables. Using SCOPED=”shared” means that you need not specify any argument; CGIARGUMENTS, specifically, are be ignored. Note that variables set since program execution began are reset to the original values passed on the URL line. Use with care.

Syntax

```
<CONTENTSERVER>
  <[ARGUMENT NAME="somearg" VALUE="somevalue"]/>
  <[CGIARGUMENT NAME="somecgiarg"]/>
</CONTENTSERVER>
```

Parameters

ARGUMENT (optional)

Arguments passed to the ContentServer as it is called.

CGIARGUMENT (optional)

CGI arguments passed to the ContentServer.

SCOPED

Valid options for this attribute are as follows:

- GLOBAL - the default behavior
- LOCAL - arguments passed in are restored (or destroyed) on return from the call.
- STACKED - the child only sees the explicit arguments passed in and can only return values in that same namespace.

Description

The CONTENTSERVER tag calls the ContentServer servlet. The CONTENTSERVER tag supports the SCOPED attribute with the default (that is, if omitted) like specifying GLOBAL, and does not protect input variables. Using LOCAL or STACKED for the SCOPED

argument protects those variables from change, but disallow sharing of input space. Use `<CGIARGUMENT NAME="name"/>` to specify that specific CGI parameters be made available to the CatalogManager. Use `<ARGUMENT>` as always for specific string values.

By omitting `SCOPED` (or setting it to "local") you can cause the CatalogManager to share all the existing input variables without scoping. Note that this means specified arguments overwrite existing variables. Using `SCOPED="shared"` means that you need not specify any argument; `CGIARGUMENTS`, specifically, are ignored. Note that variables set since program execution began are reset to the original values passed on the URL line. Use with care.

Use `CGIARGUMENT` to specify that specific CGI parameters be made available to ContentServer. Use `ARGUMENT` for specific string values.

Variables for the `CONTENTSERVER` tag are subject to an order of precedence. The order of precedence is relative to the location of the variable. The following describes the order of precedence:

1. Variables set in arguments
 2. Variables set in scripts
 3. Variables set in the **resargs1** and **resargs2** columns of the SiteCatalog
 4. Variables set in the **redetails1** and **redetails2** columns of the ElementCatalog
- In the following example, the page has the variable `arg=value2` set in the **resargs1** column of the SiteCatalog. The root element has the the variable `arg=value3` set in the **redetails1** column of the ElementCatalog. `value1` is the value with the highest order of precedence.

```
<CONTENTSERVER>
  <ARGUMENT NAME="pagename"VALUE="name"/>
  <ARGUMENT NAME="arg" VALUE="value1"/>
</CONTENTSERVER>
```

- In the following example, the page has the variable `arg=value2` in the **resargs1** column of the SiteCatalog, and the root element has the variable `arg=value3` set in the **redetails1** column of the ElementCatalog. `value2` is the value with the highest order of precedence.

```
<CONTENTSERVER>
  <ARGUMENT NAME="pagename"VALUE="name"/>
</CONTENTSERVER>
```

- In the following example, the page has nothing set in the **resargs1** column of the SiteCatalog. The root element has the variable `arg=value3` set in the **redetails1** column of the ElementCatalog. `value3` is the value with the highest order of precedence.

```
<CONTENTSERVER>
  <ARGUMENT NAME="pagename"VALUE="name"/>
</CONTENTSERVER>
```

errno

The value of `errno` is unchanged.

Example

```
<CONTENTSERVER>
  <ARGUMENT NAME="pagename" VALUE="demos/addMovie"/>
  <ARGUMENT NAME="arg" VALUE="value1"/>
</CONTENTSERVER>
```

See Also

[ARGUMENT](#)
[CATALOGMANAGER](#)
[DEBABBLE](#)
[TREEMANAGER](#)

COPYLIST

Creates a copy of the specified list.

Syntax

```
<COPYLIST LIST="NewList"
          FROM="SomeList"/>
```

Parameters

LIST (required)

The list created as a result of the copy.

FROM (required)

The source list.

Description

The COPYLIST tag creates a copy of the specified list. Note that this operation can affect performance, since all resultset rows are cached in RAM if the list is a query. Use `IsList`, specifying the name of the new list that you copied. Note that the new list is initialized to row 1 if it is not empty.

The column values of the current row can be obtained using the syntax, `List.colname` where `List` is the name of the list and `colname` is the name of the column in the list.

Note

When using `StringLists`, the values can be obtained by using `List.Item`.

You can use the following attributes to return information about the list using the syntax, *List.attname*, where *attname* can be one of the following attributes:

#numRows	Number of rows in this list.
#curRow	Current row in this list.
#moreRows	Boolean (true/false) to indicate whether there are more rows following the current row.
#numCols	Number of columns in this list.
@<colname>	Only used on lists that contain resultsets. Contents of the file in the specified column if the colname begins with "url".

For example, `MyList.#numRows` returns the number of rows in the list `MyList`. `MyList.@urlheader` retrieves the contents of the file specified by the column `urlheader`.

The value of a list is maintained during the processing of a page request. If the list contains a resultset, the resultset is maintained independently of the list that uses it.

errno

The possible values of `errno` include:

Value	Description
-505	List copy error.

Example

The following example selects all rows from the `NewPortalArticle` table and stores them in `PortalArticles`. `PortalArticles` is copied to `Articles` and headline from each row in `Articles` is displayed.

```
<SELECTTO FROM="NewPortalArticle" WHAT="*" LIST="PortalArticles" />
<SETVAR NAME = "errno" VALUE = "0" />
<COPYLIST LIST = "Articles" FROM= "PortalArticles" />
<IF COND="Variables.errno=0">
  <THEN>
    <LOOP LIST="Articles">
      <CSVAR NAME="Articles.headline"/> <br/>
    </LOOP>
  </THEN>
</IF>
```


CSVAR

Displays the value of a variable, session variable, list item, built-in, or counter.

Syntax

```
<CSVAR NAME="VALUE"/>
```

Parameters

NAME (required)

Value of the variable, session variable, list item, built-in or counter to display.

Description

The CSVAR tag displays the value of a variable, session variable, list item, built-in, or counter. For more information about variables, see the *CSEE Developer's Guide*.

errno

The value of errno is unchanged.

Example

To display the value of the myvar variable:

```
<SETVAR NAME="myvar" VALUE="today is payday!" />
I am happy because <CSVAR NAME="Variables.myvar"/><BR/>
```

The following text appears in the browser:

"I am happy because today is payday!"

Note

If the variable "myvar" was not set at the time of CSVAR, then the browser displays, "Variables.myvar"

To display the value of the session variable tempvar:

```
<SETSSVAR NAME="tempvar" VALUE="soda" />
<CSVAR NAME="SessionVariables.tempvar"/><BR/>
```

The following text appears in the browser,

"soda"

To display the value of the list item "mylist.tblname":

```
<CSVAR NAME="mylist.tblname"/>
```

To display the value of the built-in "CS.Year":

```
The current year is <CSVAR NAME="CS.Year"/><BR/>
```

If the year is 2002, the following text appears in the browser:

The current year is 2002

To display the value of the counter mycount:

```
<CSVAR NAME="Counters.mycount"/>
```

See Also

[SETVAR](#)

DEBABBLE

Converts special characters such as %20 (a space) into their representations.

Syntax

```
<DEBABBLE VALUE="TRUE or FALSE"/>
```

Parameters

VALUE (required)
true or false.

Description

The DEBABBLE tag, often used in conjunction with Content Server variables, converts special characters such as %22 (quotation marks) into their representations. By default, DEBABBLE is set to `true`; however, you can set it to `false` to keep Content Server from converting these special characters. For more information about variables, see the *CSEE Developer's Guide*.

The following table describes the character conversions:

Character	Representation
%22	"
%20	(space)
%3c	<
%3e	>
%26	&
%09	\n (line feed)
%0d	\r (carriage return)
%09	\+ (tab)
%25	% (percent)

DEBABLE works with the following tag/argument pairs only:

ARGUMENT VALUE	BEGINS WHAT	CALLELEMENT PACKEDARGS	ENDS WHAT
INDEXOF WHAT	LOGMSG STR	SEARCH WHAT	SETSSVAR VALUE
SETVAR VALUE	SQLEXP VERB	STRINGLIST DELIM	SUBSTITUTE WHAT WITH

Example

In the following example, the first CSVAR displays the phrase "Hello World," the second displays "Hello%20World," and the last line sets DEBABLE back to its default value.

```
<!-- debabble is automatically on (set to true) -->
  <SETVAR NAME="greeting" VALUE="Hello%20World"/>
  <CSVAR NAME="Variables.greeting"/><BR/>

  <DEBABLE VALUE="false"/>
  <SETVAR NAME="hello" VALUE="Hello%20World"/>
  <CSVAR NAME="Variables.hello"/><BR/>
  <!--set debabble back to it's original state -->
  <DEBABLE VALUE="true"/>
```

DELETEFILE

Deletes a file from disk.

Syntax

```
<DELETEFILE FILE="FILE_NAME"/>
```

Parameters

FILE (required)

The complete path and name of the file to be deleted. You cannot use wildcards such as * when specifying FILE_NAME.

Description

The DELETEFILE tag deletes a file from disk. Syntax must be specific to the operating system.

errno

The possible values of `errno` include:

Value	Description
-300	File not found.
-302	File not deleted.
-303	Path not found.

Example

This example deletes a Windows NT file called `foo.bar` in the directory `temp`.

```

<SETVAR NAME="errno" VALUE="0"/>
<DELETEFILE FILE="c:\temp\foo.bar"/>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>

```

See Also

[WRITEFILE](#)

DELETEREVISION

Deletes a revision of a row from a tracked table.

Syntax

```

<DELETEREVISION TABLE="TABLE_NAME"
                  ASSET="PRIMARY_KEY"
                  VERSION="VERSION_NUM"/>

```

Parameters

TABLE (required)

Name of the table containing row whose revision is to be deleted.

ASSET (required)

Value of the primary key for the row whose revision is to be deleted.

VERSION (required)

Version of the row to delete. Values for *VERSION_NUM* are:

- "first"-delete the earliest revision of row.
- "last"-delete most recent revision of row.
- "#"-delete a specific revision number.
- "date"-delete the row revised on the specified date. The date must be in SQL format (yyyy-mm-dd hh:mm:ss) and is expected to be GMT.

Description

The `DELETEREVISION` tag deletes a revision of a row from a tracked table. The current user must have `rtadmin` access to the tracked table to perform this operation. If the most recent revision is locked, deleting the most recent revision also deletes the lock on it.

errno

The possible values of `errno` include:

Value	Description
-1000	Command failure. Check the <code>errdetail</code> variable for the specific variable.

Example

This example assumes all `DELETEREVISION` parameters are exposed on an HTML form. It first checks whether the parameters exist, clears any values from `errno`, then deletes the revision.

```
<!-- Make sure required parameters exist -->
<IF COND="IsVariable.tablename=false">
  <THEN>
    <SETVAR NAME="tablename" VALUE=""/>
  </THEN>
</IF>
<IF COND="IsVariable.asset=false">
  <THEN>
    <SETVAR NAME="asset" VALUE=""/>
  </THEN>
</IF>
<IF COND="IsVariable.version=false">
  <THEN>
    <SETVAR NAME="version" VALUE=""/>
  </THEN>
</IF>
<!-- clear out errno -->
<SETVAR NAME="errno" VALUE="0"/>
<!-- do the unlock -->
<DELETEREVISION TABLE="Variables.tablename"
  ASSET="Variables.asset"
  VERSION="Variables.version"/>
<!-- See if unlock succeeded -->
<IF COND="Variables.errno=0">
  <THEN>
    <!-- if succeeded, brag about it -->
    Delete revision succeeded for
    <CSVAR NAME="Variables.asset"/><BR/>
  </THEN>
<ELSE>
  <!-- otherwise, call an error reporting element -->
  Delete revision failed.<BR/>
```

```
</ELSE>  
</IF>
```

See Also

[RELEASE](#)

DESTROYEVENT

Deletes an event.

Syntax

```
<DESTROYEVENT NAME="EVENT_NAME"/>
```

Parameters

NAME (required)
Name of event to delete.

errno

The value of errno is unchanged.

Example

This example destroys the event named NightlyPurge.

```
<DESTROYEVENT NAME="NightlyPurge"/>
```

See Also

[APPEVENT](#)
[DISABLEEVENT](#)
[ENABLEEVENT](#)

DIR.ADDATTRS

Adds attributes to existing entries.

Syntax

```
<DIR.ADDATTRS NAME="ENTRY">  
  <DIR.ARGUMENT NAME="ATTRIBUTE_NAME">  
    <DIR.ARGUMENT VALUE="ATTRIBUTE_VALUE"/>  
</DIR.ADDATTRS>
```

Parameters

NAME (required)
Unique identifier for the name of the entry to which attributes are being added. The value is relative to the basedN property.

DIR.ARGUMENT NAME (required)
Name of the attribute to be added to this entry.

DIR.ARGUMENT VALUE (required)

Specifies the use of the `CS.Property.dir.typeUser` or `CS.Property.dir.typeGroup` properties. The property specified identifies the type of the object specified by the name parameter. This value is ignored if the arguments imply the distinguished name.

Description

The `DIR.ADDATTRS` tag adds attributes to existing entries. A single attribute may have multiple order dependent values. Each attribute must have at least one value.

errno

The possible values of `errno` include:

Value	Description
-15004	Implementation doesn't support this operation
-15005	Unexpected naming exceptions from JNDI provider.
-15006	A required property missing, or class name specified for factory classes is in error.
-15007	Unexpected schema violation errors from JNDI provider.
-15008	General error in the syntax of an IName.
-15009	Unexpected invalid attribute exceptions from JNDI provider
-15010	Unexpected invalid attribute id exceptions from JNDI provider.
-15011	Unexpected invalid attribute value exceptions from JNDI provider.
-15012	User does not have permission to perform the operation.
-15014	Unexpected JNDI read error.
-15015	Unexpected JNDI write error.
-15016	JNDI attribute already in use.
-15023	Unexpected JNDI connection errors.
-15024	Required attribute missing.
-15025	Unknown error.
-15032	Add attribute SQL failed.
-15036	Missing or invalid child argument tags for tag.

Example

```
<DIR.ADDATTRS NAME="Variables.modname" />
  <DIR.ARGUMENT NAME="Variables.phoneFieldName">
    <DIR.ARGUMENT VALUE="Variables.phone" />
  </DIR.ARGUMENT>
</DIR.ADDATTRS>
```

Release Introduced

This tag was introduced in Content Server release 4.0.

DIR.ADDGROUPMEMBER

Adds a member to a group.

Syntax

```
<DIR.ADDGROUPMEMBER
    NAME="group"
    MEMBER="member entry"/>
```

Parameters

NAME (required)

Unique identifier for the name of the entry to which members are being added. The value is relative to the baseDN property.

MEMBER (required)

Name of the entry to add as a member (usually, but not required to be, a user). The value is relative to the baseDN property, but is changed to the fully qualified distinguished name prior to adding as a value to the CS.Property.dir.uniquemember attribute.

errno

The possible values of errno include:

Value	Description
-15004	Implementation doesn't support this operation
-15005	Unexpected naming exceptions from JNDI provider.
-15006	A required property missing, or class name specified for factory classes is in error.
-15007	Unexpected schema violation errors from JNDI provider.
-15008	General error in the syntax of an IName.
-15009	Unexpected invalid attribute exceptions from JNDI provider
-15010	Unexpected invalid attribute id exceptions from JNDI provider.
-15011	Unexpected invalid attribute value exceptions from JNDI provider.
-15012	User does not have permission to perform the operation.
-15013	Node cannot be deleted because it has children.
-15014	Unexpected JNDI read error.
-15015	Unexpected JNDI write error.
-15016	JNDI attribute already in use.

Value	Description
-15017	User is already a member.
-15020	User is not valid.
-15023	Unexpected JNDI connection errors.
-15024	Required attribute missing.
-15025	Unknown error.
-15026	Missing required create attribute.
-15027	User create SQL failure.
-15028	User delete SQL failure.
-15029	Cannot delete user attributes.
-15030	SQL error.
-15031	Query failed.
-15032	Add attribute SQL failed.
-15035	Database error occurred while attempting to assign the user to a group.
-15036	Missing or invalid child argument tags for tag.

Description

Adds a member to a group. Note that this tag does not add a child to a group. The API implementation, not the client code, is responsible for checking that the member is a valid, existing distinguished name. If the specified entry is already a member, nothing is done.

In most SPIs, this entails adding the specified user's distinguished name to the set of values of the specified group's `uniquemember` attribute.

If the `CS.Property.dir.referentialIntegrity` property is turned on, the `CS.Property.dir.member` attribute (backlink) of the user is also updated, if implemented by the SPI (e.g., the value in the property file is non-empty).

Example

```
<DIR.ADDGROUPMEMBER NAME="Variables.groupName"
    MEMBER="Variables.userName"/>
```

Release Introduced

This tag was introduced in Content Server release 4.0.

DIR.CHILDREN

Retrieves the immediate child entries of the specified parent.

Syntax

```
<DIR.CHILDREN
  NAME="ENTRY"
  LIST="CHILDREN"/>
```

Parameters

NAME (required)

Unique identifier for the name of the entry to list. The value is relative to the `baseDN` property.

LIST (required)

List in which to return the distinguished names of all children of the specified entry. The names returned are relative to the `baseDN` property.

errno

The possible values of `errno` include:

Value	Description
-15004	Implementation doesn't support this operation
-15005	Unexpected naming exceptions from JNDI provider.
-15006	A required property missing, or class name specified for factory classes is in error.
-15007	Unexpected schema violation errors from JNDI provider.
-15008	General error in the syntax of an <code>IName</code> .
-15009	Unexpected invalid attribute exceptions from JNDI provider
-15010	Unexpected invalid attribute id exceptions from JNDI provider.
-15011	Unexpected invalid attribute value exceptions from JNDI provider.
-15012	User does not have permission to perform the operation.
-15014	Unexpected JNDI read error.
-15015	Unexpected JNDI write error.
-15016	JNDI attribute already in use.
-15023	Unexpected JNDI connection errors.
-15024	Required attribute missing.
-15025	Unknown error.
-15036	Missing or invalid child argument tags for tag.

Example

```
<DIR.CHILDREN NAME="Variables.engineering"
  LIST="engineeringDepartments"/>
```

Release Introduced

This tag was introduced in Content Server release 4.0.

DIR.CREATE

Creates new objects.

Syntax

```
<DIR.CREATE
  [<NAME="ENTRY_NAME"/>]
  [<DIR.ARGUMENT NAME="ARG_NAME"/>]
  <DIR.ARGUMENT VALUE="ARG_VALUE"/>
</DIR.CREATE>
```

Parameters

NAME (optional)

Unique identifier for the name of the entry to create.

DIR.ARGUMENT NAME (optional)

Name of the attribute to create for this entry.

DIR.ARGUMENT VALUE (required)

Value of the named attribute.

errno

The possible values of errno include:

Value	Description
-15002	Name being created already exists.
-15004	Implementation doesn't support this operation
-15005	Unexpected naming exceptions from JNDI provider.
-15006	A required property missing, or class name specified for factory classes is in error.
-15007	Unexpected schema violation errors from JNDI provider.
-15008	General error in the syntax of an IName.
-15009	Unexpected invalid attribute exceptions from JNDI provider
-15010	Unexpected invalid attribute id exceptions from JNDI provider.
-15011	Unexpected invalid attribute value exceptions from JNDI provider.
-15012	User does not have permission to perform the operation.

Value	Description
-15014	Unexpected JNDI read error.
-15015	Unexpected JNDI write error.
-15016	JNDI attribute already in use.
-15023	Unexpected JNDI connection errors.
-15024	Required attribute missing.
-15025	Unknown error.
-15026	Missing required create attribute.
-15027	User create SQL failure.
-15036	Missing or invalid child argument tags for tag.

Description

This tag creates a new object. The name of the object is implied from its attributes, specified as arguments.

Example

```
<DIR.CREATE NAME="Variables.newname"/>
  <DIR.ARGUMENT NAME="CS.Property.dir.username"
    VALUE="Variables.loginid"/>
  <DIR.ARGUMENT NAME="CS.Property.dir.password"
    VALUE="Variables.loginpassword"/>
  <DIR.ARGUMENT NAME="Variables.emailFieldName"
    VALUE="Variables.email"/>
</DIR.CREATE>
```

Release Introduced

This tag was introduced in CSEE release 4.0.

DIR.DELETE

Deletes a named object from the namespace.

Syntax

```
<DIR.DELETE
  NAME="ENTRY_NAME"/>
```

Parameters

NAME (required)

Unique identifier for the name of the entry to delete. The value is relative to the baseDN property.

Description

This tag deletes a named object from the namespace. It is the responsibility of the client code or the directory server to maintain referential integrity. Referential integrity, means that objects are first removed from any groups to which they belong before calling delete on that object.

errno

The possible values of errno include:

Value	Description
-15003	Name being deleted does not exist.
-15004	Implementation doesn't support this operation
-15005	Unexpected naming exceptions from JNDI provider.
-15006	A required property missing, or class name specified for factory classes is in error.
-15007	Unexpected schema violation errors from JNDI provider.
-15008	General error in the syntax of an IName.
-15009	Unexpected invalid attribute exceptions from JNDI provider
-15010	Unexpected invalid attribute id exceptions from JNDI provider.
-15011	Unexpected invalid attribute value exceptions from JNDI provider.
-15012	User does not have permission to perform the operation.
-15013	Node cannot be deleted because it has children.
-15014	Unexpected JNDI read error.
-15015	Unexpected JNDI write error.
-15016	JNDI attribute already in use.
-15017	User is already a member.

Value	Description
-15018	User is not a member.
-15020	User is not valid.
-15023	Unexpected JNDI connection errors.
-15024	Required attribute missing.
-15025	Unknown error.
-15026	Missing required create attribute.
-15028	User delete SQL failure.
-15029	Cannot delete user attributes.
-15030	SQL error.
-15031	Query failed.
-15034	Database error occurred while attempting to delete an attribute.
-15036	Missing or invalid child argument tags for tag.

Example

```
<DIR.DELETE NAME="Variables.deletename"/>
```

Release Introduced

This tag was introduced in CSEE release 4.0.

DIR.GETATTRS

Retrieves the attributes for a specified entry.

Syntax

```
<DIR.GETATTRS
  LIST="ATTRIBUTE_LIST"
  NAME="ENTRY"
  [ATTRS="ATTRIBUTE_IDS"]"/>
```

Parameters

LIST (required)

Two-column list in which the attribute values are returned. Columns are NAME and VALUE. Multi-valued attributes have as many rows in the list as values. The names returned are relative to the baseDN property.

NAME (required)

Unique identifier for the name of the entry from which attributes are being retrieved. The value is relative to the baseDN property

ATTRS (optional)

A StringList of the attributes to retrieve. Default: Retrieve all attributes. If the attribute is specified, but the StringList is empty, no attributes are retrieved.

errno

The possible values of errno include:

Value	Description
-15004	Implementation doesn't support this operation
-15005	Unexpected naming exceptions from JNDI provider.
-15006	A required property missing, or class name specified for factory classes is in error.
-15007	Unexpected schema violation errors from JNDI provider.
-15008	General error in the syntax of an IName.
-15009	Unexpected invalid attribute exceptions from JNDI provider
-15010	Unexpected invalid attribute id exceptions from JNDI provider.
-15011	Unexpected invalid attribute value exceptions from JNDI provider.
-15012	User does not have permission to perform the operation.
-15014	Unexpected JNDI read error.
-15015	Unexpected JNDI write error.
-15016	JNDI attribute already in use.
-15023	Unexpected JNDI connection errors.
-15024	Required attribute missing.
-15025	Unknown error.
-15036	Missing or invalid child argument tags for tag.

Example

```
<DIR.GETATTRS NAME="Variables.name" LIST="attrList"/>
```

Release Introduced

This tag was introduced in CSEE release 4.0.

DIR.GROUPMEMBERS

Lists all members of the specified group.

Syntax

```
<DIR.GROUPMEMBERS
  NAME="GROUP"
  LIST="MEMBERS"/>
```

Parameters

NAME (required)

Unique identifier for the name of the entry for which members are being listed. The value is relative to the baseDN property

LIST (required)

A list of distinguished names of all the entries belonging to this group. The list has a single column, NAME. The names returned are relative to the baseDN property.

errno

The possible values of errno include:

Value	Description
-15004	Implementation doesn't support this operation
-15005	Unexpected naming exceptions from JNDI provider.
-15006	A required property missing, or class name specified for factory classes is in error.
-15007	Unexpected schema violation errors from JNDI provider.
-15008	General error in the syntax of an IName.
-15009	Unexpected invalid attribute exceptions from JNDI provider
-15010	Unexpected invalid attribute id exceptions from JNDI provider.
-15011	Unexpected invalid attribute value exceptions from JNDI provider.
-15012	User does not have permission to perform the operation.
-15014	Unexpected JNDI read error.
-15015	Unexpected JNDI write error.
-15016	JNDI attribute already in use.
15017	User is already a member.
-15018	User is not a member.
-15020	User is not valid.
-15023	Unexpected JNDI connection errors.

Value	Description
-15024	Required attribute missing.
-15025	Unknown error.
-15030	SQL error.
-15031	Query failed.
-15036	Missing or invalid child argument tags for tag.

Example

```
<DIR.GROUPMEMBERS NAME="Variables.groupName" LIST="members"/>
```

Release Introduced

This tag was introduced in CSEE release 4.0.

DIR.GROUPMEMBERSHIPS

Lists all groups to which a specified entry belongs.

Syntax

```
<DIR.GROUPMEMBERSHIPS
  NAME="ENTRY"
  LIST="GROUPS"/>
```

Parameters

NAME (required)

Unique identifier for the name of the entry for which to find all memberships. The value is relative to the baseDN property.

LIST (required)

A list of distinguished names of all the groups to which this entry has membership. The list has a single column, NAME. The names returned are relative to the baseDN property.

errno

The possible values of errno include:

Value	Description
-15004	Implementation doesn't support this operation
-15005	Unexpected naming exceptions from JNDI provider.
-15006	A required property missing, or class name specified for factory classes is in error.
-15007	Unexpected schema violation errors from JNDI provider.

Value	Description
-15008	General error in the syntax of an IName.
-15009	Unexpected invalid attribute exceptions from JNDI provider
-15010	Unexpected invalid attribute id exceptions from JNDI provider.
-15011	Unexpected invalid attribute value exceptions from JNDI provider.
-15012	User does not have permission to perform the operation.
-15014	Unexpected JNDI read error.
-15015	Unexpected JNDI write error.
-15016	JNDI attribute already in use.
15017	User is already a member.
-15018	User is not a member.
-15020	User is not valid.
-15023	Unexpected JNDI connection errors.
-15024	Required attribute missing.
-15025	Unknown error.
-15030	SQL error.
-15031	Query failed.
-15035	Database error occurred while attempting to assign the user to a group.
-15036	Missing or invalid child argument tags for tag.

Example

```
<DIR.GROUPMEMBERSHIPS NAME="Variables.name" LIST="groups"/>
```

Release Introduced

This tag was introduced in CSEE release 4.0.

DIR.LISTUSERS

Returns a list of all users.

Syntax

```
<DIR.LISTUSERS
  LIST="USER_LIST"
  [COMMONANCESTOR="USERS_PARENT"]/>
```

Parameters

LIST (required)

List in which to return the distinguished names of all descendants found. The names returned are relative to the basedN property.

COMMONANCESTOR (optional)

Specifies the root of the Content Server user tree. If this parameter is not specified the default is the value specified in the CS.Property.dir.people.Parent property.

errno

The possible values of errno include:

Value	Description
-15004	Implementation doesn't support this operation
-15005	Unexpected naming exceptions from JNDI provider.
-15006	A required property missing, or class name specified for factory classes is in error.
-15007	Unexpected schema violation errors from JNDI provider.
-15008	General error in the syntax of an IName.
-15009	Unexpected invalid attribute exceptions from JNDI provider
-15010	Unexpected invalid attribute id exceptions from JNDI provider.
-15011	Unexpected invalid attribute value exceptions from JNDI provider.
-15012	User does not have permission to perform the operation.
-15014	Unexpected JNDI read error.
-15015	Unexpected JNDI write error.
-15016	JNDI attribute already in use.
15017	User is already a member.
-15018	User is not a member.
-15020	User is not valid.
-15023	Unexpected JNDI connection errors.

Value	Description
-15024	Required attribute missing.
-15025	Unknown error.
-15030	SQL error.
-15031	Query failed.
-15034	Database error occurred while attempting to delete an attribute.
-15036	Missing or invalid child argument tags for tag.

Description

The parameter `commonancestor` may be specified to override the `CS.Property.dir.peopleParent` value. The same results may be achieved using the `dir.children` tag, the `dir.listusers` tag is provided as a convenience for user operations.

Example

```
<DIR.LISTUSERS LIST="allList"/>
```

Release Introduced

This tag was introduced in CSEE release 4.0.

DIR.REMOVEATTRS

Deletes the specified attribute from the specified entry.

Syntax

```
<DIR.REMOVEATTRS NAME="ENTRY">
  <DIR.ARGUMENT
    NAME="ATTRIBUTE_NAME"
    VALUE="ATTRIBUTE_VALUE"/>
</DIR.REMOVEATTRS>
```

Parameters

NAME (required)

Unique identifier for the name of the entry from which attributes are being removed.
The value is relative to the `baseDN` property.

DIR.ARGUMENT NAME (required)

Name of the attribute to be removed from this entry.

DIR.ARGUMENT VALUE (required)

Value of the named attribute.

errno

The possible values of `errno` include:

Value	Description
-15003	Name being deleted does not exist.
-15004	Implementation doesn't support this operation
-15005	Unexpected naming exceptions from JNDI provider.
-15006	A required property missing, or class name specified for factory classes is in error.
-15007	Unexpected schema violation errors from JNDI provider.
-15008	General error in the syntax of an <code>IName</code> .
-15009	Unexpected invalid attribute exceptions from JNDI provider
-15010	Unexpected invalid attribute id exceptions from JNDI provider.
-15011	Unexpected invalid attribute value exceptions from JNDI provider.
-15012	User does not have permission to perform the operation.
-15013	Node cannot be deleted because it has children.
-15014	Unexpected JNDI read error.
-15015	Unexpected JNDI write error.
-15016	JNDI attribute already in use.
-15018	User is not a member.
-15020	User is not valid.
-15023	Unexpected JNDI connection errors.
-15024	Required attribute missing.
-15025	Unknown error.
-15028	User delete SQL failure.
-15029	Cannot delete user attributes.
-15030	SQL error.
-15031	Query failed.
-15034	Database error occurred while attempting to delete an attribute.
-15036	Missing or invalid child argument tags for tag.

Example

```
<DIR.REMOVEATTRS NAME="Variables.modname">
  <DIR.ARGUMENT NAME="Variables.phoneFieldName"/>
</DIR.REMOVEATTRS>
```

Release Introduced

This tag was introduced in CSEE release 4.0.

DIR.REMOVEGROUPMEMBER

Removes a member from a group.

Syntax

```
<DIR.REMOVEGROUPMEMBER
  NAME="group"
  MEMBER="member entry"/>
```

Parameters

NAME (required)

Unique identifier for the name of the entry from which members are being removed. The value is relative to the baseDN property.

MEMBER (required)

Name of the entry to remove as a member (usually, but not required to be, a user). The value is relative to the baseDN property, but is changed into the fully-qualified distinguished name prior to searching for the matching CS.Property.dir.uniqueMember attribute value.

errno

The possible values of errno include:

Value	Description
-15003	Name being deleted does not exist.
-15004	Implementation doesn't support this operation
-15005	Unexpected naming exceptions from JNDI provider.
-15006	A required property missing, or class name specified for factory classes is in error.
-15007	Unexpected schema violation errors from JNDI provider.
-15008	General error in the syntax of an IName.
-15009	Unexpected invalid attribute exceptions from JNDI provider
-15010	Unexpected invalid attribute id exceptions from JNDI provider.
-15011	Unexpected invalid attribute value exceptions from JNDI provider.
-15012	User does not have permission to perform the operation.

Value	Description
-15013	Node cannot be deleted because it has children.
-15014	Unexpected JNDI read error.
-15015	Unexpected JNDI write error.
-15016	JNDI attribute already in use.
-15018	User is not a member.
-15020	User is not valid.
-15023	Unexpected JNDI connection errors.
-15024	Required attribute missing.
-15025	Unknown error.
-15026	Missing required create attribute.
-15028	User delete SQL failure.
-15029	Cannot delete user attributes.
-15030	SQL error.
-15031	Query failed.
-15034	Database error occurred while attempting to delete an attribute.
-15036	Missing or invalid child argument tags for tag.

Description

This tag removes a member from a group, note that this tag does not remove a child from a group. In most SPIs, this entails removing the specified user's distinguished name from the set of values of the specified group's `uniquemember` attribute. Not that this tag does not remove a child from a group hierarchy through a delete operation.

If `CS.Property.dir.referentialIntegrity` is turned on, the `CS.Property.dir.member` attribute (backlink) of the user is also updated, if implemented by the SPI (e.g., the value in the property file is not populated).

Example

```
<DIR.REMOVEGROUPMEMBER NAME="Variables.groupName"
MEMBER="Variables.userName"/>
```

Release Introduced

This tag was introduced in CSEE release 4.0.

DIR.REPLACEATTRS

Replaces the existing attribute values with the specified values.

Syntax

```
<DIR.REPLACEATTRS NAME="ENTRY">
  <DIR.ARGUMENT
    NAME="ATTRIBUTE_NAME"
    VALUE="ATTRIBUTE_VALUE"/>
</DIR.REPLACEATTRS
```

Parameters

NAME (required)

Unique identifier for the name of the entry to which attributes are being replaced. The value is relative to the baseDN property.

DIR.ARGUMENT NAME (required)

Name of the attribute to be replaced for this entry.

DIR.ARGUMENT VALUE (required)

Value of the named attribute.

errno

The possible values of errno include:

Value	Description
-15002	Name being created already exists.
-15004	Implementation doesn't support this operation
-15005	Unexpected naming exceptions from JNDI provider.
-15006	A required property missing, or class name specified for factory classes is in error.
-15007	Unexpected schema violation errors from JNDI provider.
-15008	General error in the syntax of an IName.
-15009	Unexpected invalid attribute exceptions from JNDI provider
-15010	Unexpected invalid attribute id exceptions from JNDI provider.
-15011	Unexpected invalid attribute value exceptions from JNDI provider.
-15012	User does not have permission to perform the operation.
-15014	Unexpected JNDI read error.
-15015	Unexpected JNDI write error.
-15016	JNDI attribute already in use.
15017	User is already a member.

Value	Description
-15018	User is not a member.
-15020	User is not valid.
-15023	Unexpected JNDI connection errors.
-15024	Required attribute missing.
-15025	Unknown error.
-15026	Missing required create attribute.
-15027	User create SQL failure.
-15029	Cannot delete user attributes.
-15030	SQL error.
-15031	Query failed.
-15032	Add attribute SQL failed.
-15033	Database error handling new attribute.
-15034	Database error occurred while attempting to delete an attribute.
-15036	Missing or invalid child argument tags for tag.

Example

```
<DIR.REPLACEATTRS NAME="Variables.modname"/> <DIR.ARGUMENT
NAME="Variables.phFieldName" VALUE="Variables.newph"/> </
DIR.REPLACEATTRS>
```

Release Introduced

This tag was introduced in CSEE release 4.0.

DIR.SEARCH

Searches for entries matching the specified attributes.

Syntax

```
<DIR.SEARCH
  LIST="matches"
  CONTEXT="common ancestor"
  [OPERATOR="and|or"]>
  <DIR.ARGUMENT NAME="ATTRIBUTE_NAME"
    VALUE="ATTRIBUTE_VALUE"/>
</DIR.SEARCH>
```

Parameters

LIST (required)

List in which the distinguished names of the matching entries are returned. The list contains a single column called NAME.

CONTEXT (required)

Overrides the value of the parent in the property file

CS.Property.dir.peopleParent for type=user. Specify this argument for searching under groups or under a branch of the user tree.

OPERATOR (optional)

Determines if the matching entries must contain all "and" or one or more "or" of the specified attribute/value pairs. The default is "and".

DIR.ARGUMENT NAME (required)

The name of the attribute to be matched.

DIR.ARGUMENT VALUE (required)

Value of the attribute to be matched.

errno

The possible values of errno include:

Value	Description
-15004	Implementation doesn't support this operation
-15005	Unexpected naming exceptions from JNDI provider.
-15006	A required property missing, or class name specified for factory classes is in error.
-15007	Unexpected schema violation errors from JNDI provider.
-15008	General error in the syntax of an IName.
-15009	Unexpected invalid attribute exceptions from JNDI provider
-15010	Unexpected invalid attribute id exceptions from JNDI provider.
-15011	Unexpected invalid attribute value exceptions from JNDI provider.
-15012	User does not have permission to perform the operation.
-15014	Unexpected JNDI read error.
-15015	Unexpected JNDI write error.
-15016	JNDI attribute already in use.
-15023	Unexpected JNDI connection errors.
-15024	Required attribute missing.
-15025	Unknown error.
-15030	SQL error.
-15031	Query failed.

Value	Description
-15036	Missing or invalid child argument tags for tag.

Example

```
<DIR.SEARCH LIST="matchList">
  <DIR.ARGUMENT NAME="CS.Property.dir.username"
                VALUE="Variables.loginid"/>
</DIR.SEARCH>
```

Release Introduced

This tag was introduced in CSEE release 4.0.

DISABLEEVENT

Disables an event.

Syntax

```
<DISABLEEVENT NAME="EVENT_NAME"/>
```

Parameters

NAME (required)
Name of event to disable

Description

The DISABLEEVENT tag disables an event. The event continues to exist; however, it is never triggered while disabled.

errno

The value of `errno` is unchanged.

Example

This example disables the event named `NightlyPurge`.

```
<DISABLEEVENT NAME="NightlyPurge"/>
```

See Also

[ENABLEEVENT](#)

DISABLECACHE

Forces the current evaluation to not be cached either by the EvalServer or the ContentServer servlets.

Syntax

```
<DISABLECACHE/>
```

Parameters

None.

errno

The value of `errno` is unchanged.

EMAILEVENT

Creates an event that sends SMTP mail to one or more recipients at a specific time.

Syntax

```
<EMAILEVENT NAME="EVENT_NAME"  
            TO="USER"  
            TIMES="SEND_TIMES"  
            FILE="FILE_NAME"/>
```

Parameters

NAME (required)

Name of the e-mail event.

TO (required)

A comma-separated list of the recipients to receive the message

TIMES (required)

The time at which the event is triggered. Use the format:

hh:mm:ss W/DD/MM

where:

- *hh*: 0 - 23
- *mm*: 0 - 59
- *ss*: 0 - 59
- *W* (day of the week): 0-6 with 0 = Sunday.
- *DD* (day of the month): 1 - 31
- *MM* (month): 1 - 12

For each of these fields, you can use an asterisk (all legal values) or a list of elements separated by commas. A value can be one or more numbers separated by a minus sign, indicating an inclusive range. For example, 2, 5 - 7:0:0 5/*/* means the event is triggered at 2 a.m., 5 a.m., 6 a.m. and 7 a.m. every Friday.

Specify days by two fields: day of the month (DD) and day of the week (W). If you specify both, both take effect. For example, 1:0:0 1/15/* means the event is triggered

at 1 a.m. every Monday, as well as on the fifteenth of each month. To specify days by only one field, set the other field to *.

FILE (required)

Complete file specification of the file whose content is used as the body of the mail message.

Description

The MAILEVENT tag creates an event that sends SMTP mail to one or more recipients at a specific time. To send mail, the Content Server properties file (`futuretense.ini`) must contain valid values for the `cs.emailhost` and `cs.emailreturnto` properties. To receive mail, the Content Server properties file must contain valid values for the `cs.emailhost`, `cs.emailaccount`, and `cs.emailpassword` properties.

errno

The possible values of `errno` include:

Value	Description
-200	Failed to register event.
-201	Exception in event.

Example

This example constructs an email event called `SomeEmail` which sends mail to user `joeuser` every day at 12:55 in the afternoon. The file `"readme.txt"` contains the text used for the body of the mail message.

```
<SETVAR NAME="errno" VALUE="0"/>
<EMAILEVENT NAME="SomeEmail"
  TO="joeuser@someaddress.com"
  TIMES="12:55:00 */*/*"
  FILE="c:\program files\xircom\cem56\readme.txt"/>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>
```

See Also

[SENDMAIL \(Variant 1\)](#)

ENABLEEVENT

Enables a disabled event.

Syntax

```
<ENABLEEVENT NAME="EVENT_NAME"/>
```

Parameters

NAME (required)
Name of event to enable.

Description

The ENABLEEVENT tag enables a disabled event. An event is enabled automatically when using APPEVENT.

errno

The value of errno is unchanged.

Example

This example enables the event named NightlyPurge.

```
<ENABLEEVENT NAME="NightlyPurge"/>
```

See Also

[APPEVENT](#)
[DESTROYEVENT](#)
[DISABLEEVENT](#)

ENDS

Determines whether a string ends with the specified substring.

Syntax

```
<ENDS STR="SEARCH_STRING"  
      WHAT="ENDSTRING_VALUE"/>
```

Parameters

STR (required)
String to search.
WHAT (required)
Value of the substring to search for in *SEARCH_STRING*.

Description

The ENDS tag determines whether a string ends with the specified substring. The comparison is case-sensitive. For more information about variables, see the *CSEE Developer's Guide*.

errno

The possible values of `errno` include:

Value	Description
1	Comparison is true.
0	Comparison is false.

See Also

[BEGINS](#)

EVALSERVER (Deprecated)

As of Content Server version 4.0. FatWire strongly recommends the use of CS-Satellite for your memory-caching needs.

This is a more efficient way to call the EvalServer application logic. The attribute `SCOPED` is supported, and the default (that is, if omitted) is like specifying “GLOBAL,” and does not protect input variables. Using `LOCAL` or `STACKED` for the `SCOPED` argument protects those variables from change, but disallow sharing of input space. Use `<CGIARGUMENT NAME="name"/>` to specify that specific CGI parameters be made available to the ContentServer servlet. Use `<ARGUMENT>` as always for specific string values.

By omitting `SCOPED` (or setting it to “local”) you can cause the ContentServer servlet to share all the existing input variables without scoping. Note that this means specified arguments overwrite existing variables. Using `SCOPED="shared"` means that you need not specify any argument; `CGIARGUMENTS`, specifically, is ignored. Note that variables set since program execution began are reset to the original values passed on the URL line. Use with care.

Syntax

```
<EVALSERVER>
  [<ARGUMENT NAME="somearg" VALUE="somevalue"/>]
  [<CGIARGUMENT NAME="somecgiarg"
    VALUE="somecgivalue"/>]
</EVALSERVER>
```

Parameters

ARGUMENT (optional)

Arguments passed to the ContentServer servlet as it is called.

CGIARGUMENT (optional)

CGI arguments passed to the ContentServer servlet.

SCOPED

Valid options for this attribute are as follows:

- GLOBAL - the default behavior
- LOCAL - arguments passed in are restored (or destroyed) on return from the call.
- STACKED - the child only sees the explicit arguments passed in and can only return values in that same namespace.

Description

The EVALSERVER tag calls the EvalServer servlet. The EVALSERVER tag supports the SCOPED attribute with the default (that is, if omitted) like specifying GLOBAL, and does not protect input variables. Using LOCAL or STACKED for the SCOPED argument protects those variables from change, but disallow sharing of input space. Use <CGIARGUMENT NAME="name"/> to specify that specific CGI parameters be made available to the CatalogManager. Use <ARGUMENT> as always for specific string values.

By omitting SCOPED (or setting it to "local") you can cause the CatalogManager to share all the existing input variables without scoping. Note that this means specified arguments overwrite existing variables. Using SCOPED="shared" means that you need not specify any argument; CGIARGUMENTS, specifically, are ignored. Note that variables set since program execution began are reset to the original values passed on the URL line. Use with care.

Use CGIARGUMENT to specify that specific CGI parameters be made available to EvalServer. Use ARGUMENT for specific string values.

To use EvalServer to retrieve pages from memory, set the `es.security` property to false. If `es.security` is set to true, Content Server retrieves current information about the page from the database; therefore, you are always retrieving the page anew and not from memory cache. For information on flushing the EvalServer cache, see the *CSEE Developer's Guide*.

errno

The value of `errno` is unchanged.

Example

```
<EVALSERVER>
  <ARGUMENT NAME="pagename" VALUE="demos/addMovie"/>
</EVALSERVER>
```

See Also

[CONTENTSERVER \(Deprecated\)](#)
[CATALOGMANAGER](#)
[TREEMANAGER](#)

EXECSQL

Executes an inline SQL statement.

Syntax

```
<EXECSQL          SQL="SQL_COMMAND"
                  LIST="LIST_NAME"
                  [LIMIT="MAX_NO_RESULTS"]
                  [TABLE="table name"] />
```

Parameters

SQL (required)

SQL commands to execute.

LIST (required)

Name of list to contain resultset.

LIMIT (optional)

Maximum number of rows to be retrieved. If LIMIT is omitted or if the LIMIT value is set to less than or equal to 0, all records from the SQL command are returned.

TABLE (optional)

A comma-separated list of table to cache or synchronize against.

Description

The EXECSQL tag executes an inline SQL statement. Because the SQL statement is inline and not from the SystemSQL table, you must set the variable `tablename` before the call; otherwise, an error can occur. Furthermore, `tablename` must exist in the database prior to calling EXECSQL. The resultset is cached against the table indicated by the value of `tablename`.

Note

If the limit is less than or equal to 0, then no limit exists and the entire recordset is returned.

This tag supports multiple table caching. You provide a comma-separated list of tables via the TABLE parameter. The first table is the “primary” table, that is, the resultset is cached based on that table’s caching rules.

errno

The possible values of `errno` include:

Value	Description
-101	No rows found.
-102	Empty column.
-103	No such table.

Value	Description
-104	No table definition.
-106	Bad parameter

Example

This example selects headline from the table FTX1EE17FWB:

```
<SETVAR NAME="tablename" VALUE="NewPortalArticle"/>
<EXECSQL LIST="articles" SQL="select headline from
Variables.tablename where id='FTX1EE17FWB'"/>
<CSVAR NAME="articles.headline"/>
```

See Also

[CALLSQL](#)
[SELECTTO](#)
[SQLEXP](#)

EXPORT.EXPORT

Generates one or more static pages from a Content Server page name.

Syntax

```
<EXPORT .EXPORT
  PAGE="page name"
  FOLDER="folder name"
  WEBFOLDER="folder name"
  LIST="variable name"/>
```

Parameters

PAGE

Name of the page to export.

FOLDER

Name of the folder where the pages are written.

WEBFOLDER

URL prefix for all exported pages. For example, in the URL
mysite.com/foofolder/index.html, foofolder is the prefix.

LIST

Name of the output list.

Example

This example is taken from the JumpStart demonstration.

```
<IF COND="Variables.operation=display">
  <THEN>
    <!-- Displaying page -->
    <export.display page="Variables.exportpage" />
  </THEN>
  <ELSE>
    <TIME.SET NAME="counter"/>
    <!-- Exporting page -->
    <export.export page="Variables.exportpage"
      folder="Variables.folder"
webfolder="Variables.webfolder"
      list = "myinventory"/>
    <IF COND="IsList.myinventory=true">
      <THEN>
        <TABLE BORDER="1">
          <TH>Status</TH>
          <TH>File Name</TH>
          <TH>Key</TH>
          <TH>Query String</TH>
          <LOOP LIST="myinventory">
            <TR>
              <TD>
                <CENTER>
                  <IF COND="myinventory.status=not processed">
                    <THEN>--</THEN>
                    <ELSE><CSVAR NAME="myinventory.status"/></ELSE>
                  </IF>
                </CENTER>
              </TD>
              <TD>
                <CSVAR NAME="myinventory.fname"/>
              </TD>
              <TD>
                <CSVAR NAME="myinventory.key"/>
              </TD>
              <TD>
                <CSVAR NAME="myinventory.qry"/>
              </TD>
            </TR>
          </LOOP>
        </TABLE>
      </THEN>
    </IF>
    <P>Elapsed time: <TIME.GET NAME="counter"/> ms.</P>
  </ELSE>
</IF>
```

FILELIST

Constructs a list of the file names found in a folder.

Syntax

```
<FILELIST NAME="LIST_NAME"  
  PATH="PATH_NAME"  
  [FILTER="FILTER_VAL"]  
  [RECURSIVE="RECURSIVE_VAL"]  
  [LIMIT="MAX_NO_FILES"]/>
```

Parameters

NAME (required)

List of files found.

PATH (required)

Folder to scan.

FILTER (optional)

File filter to use. Possible filters are *.* or *.

RECURSIVE (optional)

Whether to list subfolders. Possible values are true or false.

LIMIT (optional)

The maximum number of file names to return in the list.

Description

The FILELIST tag constructs a list of the file names found in a folder. The created list contains the following columns:

FULLNAME	Full path name of the file.
NAME	Name of the file.
SIZE	Size of the file.
DIR	Set to true if the row is a folder, or false if the row is a file.

The column values of the current row can be obtained using the syntax, *List.colname* where *List* is the name of the list and *colname* is the name of the column in the list.

Note

In the case of *StringLists*, the values can be obtained by using *List.Item*.

You can use the following attributes to return information about the list using the syntax, *List.attname*, where *attname* can be one of the following attributes:

#numRows	Number of rows in this list
#curRow	Current row in this list.
#moreRows	Boolean (true/false) to indicate whether there are more rows following the current row.
#numCols	Number of columns in this list.
@<colname>	Only used on lists that contain resultsets. Contents of the file in the specified column if the colname begins with "url".

For example, `MyList.#numRows` returns the number of rows in the list `MyList`. `MyList.@urlheader` retrieves the contents of the file specified by the column `urlheader`. The value of a list is maintained during the processing of a page request. If the list contains a resultset, the resultset is maintained independently of a list that uses it.

errno

The possible values of `errno` include:

Value	Description
-16	Empty list.

Example

This example gets a list of all *.html files found in the folder `mydir` and stores the results in `myfiles`.

```
<SETVAR NAME="errno" VALUE="0"/>
<FILELIST NAME="myfiles" FILTER="*.html" RECURSIVE="true"
  PATH="c:\mydir" />
<IF COND="Variables.errno=-16">
  <THEN>
    No files found.
  </THEN>
  <ELSE>
    <LOOP> LIST="myfiles">
      Full path:
      <CSVAR NAME="myfiles.FULLNAME"/> <br/>
      Filename:
      <CSVAR NAME="myfiles.NAME"/> <br/>
      Size:
      <CSVAR NAME="myfiles.SIZE"/> <br/>
      Dir is:
      <CSVAR NAME="myfiles.DIR"/> <br/>
    </LOOP>
  </ELSE>
```

</IF>

See Also

[RENAMELIST](#)

FLUSH

Flushes a list.

Syntax

```
<FLUSH LIST="LIST_NAME"/>
```

Parameters

LIST (required)
Name of the list to flush.

Description

The FLUSH tag flushes a list. If the list contains a database resultset, FLUSH clears the resultset from the cache. If the list contains a `filelist` or `stringlist`, FLUSH frees system resources.

errno

The possible values of `errno` include:

Value	Description
-100	General exception.

Example

The following example selects all rows from the `NewPortalArticle` table, displays the `headline` from each row, then flushes the list. Since the list is flushed, subsequent identical queries are rerun against the database. If the list is not flushed, the cached resultset is used for subsequent identical queries against the `NewPortalArticle` table.

```

<RENAMELIST LIST = "Articles" FROM = "PortalArticles" />
<SELECTTO FROM="NewPortalArticle" WHAT="*" LIST="Articles" />
<SETVAR NAME = "errno" VALUE = "0" />
<IF COND="Variables.errno=0">
  <THEN>
    <LOOP LIST="Articles">
      <CSVAR NAME="Articles.headline"/> <br/>
    </LOOP>
  </THEN>
</IF>
<FLUSH LIST = "Articles" />

```

See Also

[STRINGLIST](#)
[SELECTTO](#)

GOTOROW

Sets the current row in a list to a specific row.

Syntax

```
<GOTOROW LIST="LIST_NAME"
      ROWNUM="NUMBER" />
```

Parameters

LIST (required)

Name of list.

ROWNUM (required)

Row number to set as current row.

Description

The GOTOROW tag sets the current row in a list to a specific row. The system generates runtime errors if the value of ROWNUM is not an integer. Values beyond the range of the list also yield an error.

The column values of the current row can be obtained using the syntax, *List.colname* where *List* is the name of the list and *colname* is the name of the column in the list.

Note

In the case of *StringLists*, the values can be obtained by using *List.Item*.

You can use the following attributes to return information about the list using the syntax, *List.attname*, where *attname* can be one of the following attributes:

#numRows	Number of rows in this list
#curRow	Current row in this list.
#moreRows	Boolean (true/false) to indicate whether there are more rows following the current row.
#numCols	Number of columns in this list.
@<colname>	Only used on lists that contain resultsets. Contents of the file in the specified column if the colname begins with "url".

For example, *MyList.#numRows* returns the number of rows in the list *MyList*. *MyList.@urlheader* retrieves the contents of the file specified by the column *urlheader*.

The value of a list is maintained during the processing of a page request. If the list contains a resultset, the resultset is maintained independently of a list that uses it.

errno

The possible values of `errno` include:

Value	Description
2	No row increment.
-500	No such row.
-503	No such list.

Example

This example moves the list, `SysList`, to row 3. The value of `tblname` for row 3 is then displayed.

```
<SELECTTO FROM="SystemInfo"
      WHAT="*"
      LIST="SysList" />
<GOTOROW LIST="SysList" ROWNUM="3"/>
<CSVAR NAME="SysList.tblname"/>
```

See Also

[SETROW](#)

HISTORY

Gets a revision history for rows in a revision tracked table.

Syntax

```
<HISTORY TABLE="TABLE_NAME"
      LIST="QUERY_OBJECT"
      [ASSET="PRIMARY_KEY"]
      [VERSION="VERSION_NUM"]
      [LOCKER="LOCK_USER"]
      [CREATOR="CREATE_USER"]
      [ANNOTATION="COMMENT"] />
```

Parameters

TABLE (required)

Name of the tracked table.

LIST (required)

List containing the returned results.

ASSET (optional)

Value of the primary key for the row whose history to retrieve.

VERSION (optional)

Version number.

LOCKER (optional)

Filter rows returned based on a user who has rows locked. If the current user does not have `rtaudit` privileges, you must specify a value for `LOCKER`, `CREATOR`, or both. If you specify only `LOCKER`, the value must be the current users name; otherwise, no data is returned.

CREATOR (optional)

Filter rows returned based on the user who created the row. If the current user does not have `rtaudit` privileges, you must specify a value for `LOCKER`, `CREATOR`, or both. If you specify only `CREATOR`, the value must be the current users name; otherwise, no data is returned.

ANNOTATION (optional)

Filter rows returned based on the annotation of a row.

Description

The `HISTORY` tag gets a revision history for rows in a revision tracked table.

To obtain history for all rows, the current user must have `rtaudit` privileges on the table. If the current user has read privileges on the table, only history on the revisions which the user has created or locked is returned.

errno

The possible values of `errno` include:

Value	Description
-1000	Command failure. Check the <code>errdetail</code> variable for the specific variable.

Example

This example assumes all `HISTORY` parameters are exposed on an HTML form. It first checks whether the parameters exist, clears any values from `errno`, then checks the history.

```
!-- Make sure required parameters exist -->
<IF COND="IsVariable.tablename=false">
  <THEN>
    <SETVAR NAME="tablename" VALUE="" />
  </THEN>
</IF>
<IF COND="IsVariable.asset=false">
  <THEN>
    <SETVAR NAME="asset" VALUE="" />
  </THEN>
</IF>
<IF COND="IsVariable.version=false">
  <THEN>
    <SETVAR NAME="version" VALUE="" />
  </THEN>
</IF>
```

```

<IF COND="IsVariable.locker=false">
  <THEN>
    <SETVAR NAME="locker" VALUE=""/>
  </THEN>
</IF>
<IF COND="IsVariable.creator=false">
  <THEN>
    <SETVAR NAME="creator" VALUE=""/>
  </THEN>
</IF>
<IF COND="IsVariable.annotation=false">
  <THEN>
    <SETVAR NAME="annotation" VALUE=""/>
  </THEN>
</IF>
<!-- clear out errno -->
<SETVAR NAME="errno" VALUE="0"/>
<!-- get the history -->
<HISTORY TABLE="Variables.tablename"
  LIST="historyResult"
  ASSET="Variables.asset"
  VERSION="Variables.version"
  LOCKER="Variables.locker"
  CREATOR="Variables.creator"
  ANNOTATION="Variables.annotation"/>
<!-- See if history succeeded -->
<IF COND="Variables.errno=0">
  <THEN>
    <!-- If succeeded, report it -->
    <TABLE CELLPADDING="2" CELLSPACING="4">
      <TR>
        <TH>Asset</TH><TH>Version Number</TH>
        <TH>Version Date</TH><TH>Created by</TH>
        <TH>Locked by</TH><TH>Comment</TH>
      </TR>
      <LOOP LIST="historyResult"/>
      <TR>
        <TD><CSVAR NAME="historyResult.asset"/></TD>
        <TD><CSVAR NAME="historyResult.versionnum"/>
        </TD>
        <TD><CSVAR NAME="historyResult.versiondate"/>
        </TD>
        <TD><CSVAR NAME="historyResult.createdby"/>
        </TD>
        <TD><CSVAR NAME="historyResult.lockedby"/>
        </TD>
        <TD><CSVAR NAME="historyResult.annotation"/>
        </TD>
      </TR>
    </LOOP>
  </TABLE>
</THEN>

```

```

    <ELSE>
    <!-- otherwise, call an error reporting element -->
        History failed.<BR/>
    </ELSE>
</IF>

```

See Also

[ICS.ARGUMENT](#)
[RENAMELIST](#)

ICS.ARGUMENT

Declares a value that is available to the parent tag.

Syntax

```

<ICS.ARGUMENT
    NAME="argument name"
    VALUE="argument value"/>

```

Parameters

NAME (required)
 Name of the argument.
VALUE (required)
 Value for the argument.

Description

The ICS.ARGUMENT tag declares a value that is available to the parent tag. It is used in the context of another ICS tag, and causes a runtime error if it is not contained within a proper parent tag.

errno

The value of errno is unchanged.

Example

```

<SETVAR NAME="base" VALUE="CS.Property.ft.cgipath"/>
<ICS.ENCODER BASE="Variables.baseContentServer"
    SESSION="true"
    OUTPUT="URL">
    <ICS.ARGUMENT NAME="pagename" VALUE="testpage"/>
    <ICS.ARGUMENT NAME="k1" VALUE="v1"/>
    <ICS.ARGUMENT NAME="k2" VALUE="v2"/>
    <ICS.ARGUMENT NAME="k3" VALUE="v3"/>
    <ICS.ARGUMENT NAME="k4" VALUE="v4"/>
    <ICS.ARGUMENT NAME="k5" VALUE="v5"/>
    <ICS.ARGUMENT
</ICS.ENCODER>
<CSVAR NAME="Variables.URL"/>

```

The browser displays:

```
/servlet/  
ContentServer?pagename=testpage&k1=v1&k2=v2&k3=v3&k4=v4&k5=v5
```

ICS.CDATA

Includes a CDATA in the output of a rendered XML element.

Syntax

```
<ICS.CDATA>  
    ...data...  
</ICS.CDATA>
```

Parameters

None.

Description

This tag enables streaming a CDATA wrapper around some data from Content Server. Use this tag in cases where another XML parser reads the page; for example WAP or SOAP.

Example

In the following example, the variable body is set to “This is the body of my article.” The following code is used:

```
<ICS.CDATA>  
    <ICS.GETVAR NAME=”body”/>  
</ICS.CDATA>
```

The output is as follows:

```
<![CDATA[  
    This is the body of my article.  
]]>
```

Release Introduced

This tag was introduced in CSEE release 5.0.

ICS.CLEARERRNO

Clears the current error state.

Syntax

```
<ICS.CLEARERRNO/>
```

Description

The ICS.CLEARERRNO tag clears the current error state, setting the value of the error state to 0. Call this tag immediately before invoking a tag that may alter the `errno`.

errno

None.

Example

The following example clears the error state before executing the `ics:catalogmanager` command:

```
<ICS:CLEARERRNO/>
  <CATALOGMANAGER>
    <ICS.ARGUMENT NAME="FTCMD" VALUE="DELETETABLE" />
    <ICS.ARGUMENT NAME="TABLENAME" VALUE="MOVIES" />
  </CATALOGMANAGER>
```

Release Introduced

This tag was introduced in CSEE release 5.0.

ICS.ENCODE

Builds a URL string from a base and key/value pairs.

Syntax

```
<ICS.ENCODE BASE="SOME BASE"
  [SESSION="TRUE OR FALSE"]
  [OUTPUT="SOME OUTPUT"]/>
```

Parameters

BASE (required)

Base of the URL.

SESSION (optional)

Specifies to include the session ID in the URL when set to `true`. Set this parameter to `true` if the session ID is required by the client.

OUTPUT (optional)

Specifies an optional output variable. If this parameter is not specified the output is streamed to the browser.

Description

This tag builds a URL string from a base and key/value pairs. The keys and values are encoded as required and the session ID can be added optionally.

errno

None.

Example

```
<SETVAR NAME="base" VALUE="CS.Property.ft.cgipath"/>
<ICS.ENCODER BASE="Variables.base/ContentServer"
  SESSION="true"
  OUTPUT="url">
  <ICS.ARGUMENT NAME="PAGENAME" VALUE="TESTPAGE"/>
  <ICS.ARGUMENT NAME="k1" VALUE="v1"/>
  <ICS.ARGUMENT NAME="k2" VALUE="v2"/>
  <ICS.ARGUMENT NAME="k3" VALUE="v3"/>
  <ICS.ARGUMENT NAME="k4" VALUE="v4"/>
  <ICS.ARGUMENT NAME="k5" VALUE="v5"/>
</ICS.ENCODER>
<CSVAR NAME="Variables.URL"/>
```

The browser displays:

```
/servlet/
ContentServer?pagename=testpage&k1=v1&k2=v2&k3=v3&k4=v4&k5=v5
```

See Also

[ICS.ARGUMENT](#)

ICS.GETERRNO

Gets the current error code as an integer.

Syntax

```
<ICS.GETERRNO/>
```

Description

Retrieves the error code of an element as an integer.

errno

None.

Example

The following example issues a call to a tag, then uses ICS.GETERRNO to determine if the method reported an error.

```
<ICS.CLEARERRNO/>
<CALLSQL QRYNAME="GETTABLES" LIST="NEWS"/>
<ICS.GETERRNO/>
```

Release Introduced

This tag was introduced in CSEE release 5.0.

ICS.GETPROPERTY

Gets a property value from the loaded Content Server property files.

Syntax

```
<ICS.GETPROPERTY  
    NAME="PropertyName"  
    [FILE="Some Property File Name"]  
    [OUTPUT="Some Output"]  
</ICS.GETPROPERTY/>
```

Parameters

NAME (required)

The name of the property to retrieve.

FILE (optional)

A semicolon-separated list of property file names.

OUTPUT (optional)

Specifies an optional output variable. If this parameter is not specified, the output is streamed to the browser.

Description

Retrieves a property value from the loaded or specified Content Server Property file. The property may then be used in conjunction with other Content Server XML tags.

errno

None.

Example

The following example displays the value of a property from an alternative property file.

```
<ICS.GETPROPERTY NAME="myproperty"  
    FILE="myfile.ini"/>
```

ICS.GETVAR

Returns the value of a Content Server variable.

Syntax

```
<ICS.GETVAR  
  [NAME="varname"]  
  ENCODING="default"  
  [OUTPUT="some output"]/>
```

Parameters

NAME (optional)

The regular name of the variable.

ENCODING (required)

The encoding used to convert binary values to a string. The value of this parameter must be set to default.

OUTPUT (optional)

Specifies an optional output variable. If this parameter is not specified, the output is streamed to the browser.

Description

The `ics.getvar` tag returns the value of a variable. If the variable does not exist, `ics.getvar` returns an empty string.

errno

None.

Example

The following line retrieves the value of the Content Server variable `username`.

```
<ics.getvar name=username"  
  encoding="default"/>
```

ICS.LISTGET

Retrieves a list.

Syntax

```
<ICS.LISTGET LISTNAME="list name"  
             FIELDNAME="some field"  
             [OUTPUT="name"] />
```

Parameters

LISTNAME (required)

Name of the list to be retrieved.

FIELDNAME (required)

Any column in the retrieved list. The ICS.LISTGET tag retrieves that column's value for the current row. In addition, the following keywords may be specified for the field name:

- #curRow – Retrieve the current row number.
- #numCols – Retrieve the number of columns in the list.
- #numRows – Retrieve the number of rows in the list.

OUTPUT (optional)

If this parameter is specified, the value of the request field in the list is set as a variable using the name specified.

Description

This tag retrieves the value of the specified field in a specified list.

errno

None.

Release Introduced

This tag was introduced in CSEE release 5.0.

ICS.LITERAL

Formats literals for [CALLSQL](#) and other tasks beyond those for which [SQLEXP](#) is designed.

Syntax

```
<ICS.LITERAL  
    TABLE="table name"  
    COLUMN="column name"  
    STRING="string"  
    OUTPUT="variable name" />
```

Parameters

TABLE (required)

Name of table.

COLUMN (required)

Name of column in the table.

STRING (required)

String that needs to be formatted into a literal for the specified column.

OUTPUT (required)

Variable name where the literal is placed.

Description

Specifying the table name and column name, Content Server determines which type of literal to create. Content Server has database-specified properties to help it format both string (`cc.stringpicture`) and timestamp literals (`cc.datepicture`). For `cc.datetime` data types, Content Server uses the `cc.datepicture` property. For `cc.char`, `cc.varchar`, and `cc.bigtext`, it uses the `cc.stringpicture` property.

errno

The possible values of `errno` include:

Value	Description
-100	General exception error.
-103	Invalid table name.
-106	Invalid column name.
-10004	Missing required parameter.

Example

The following query, named myquery, exists in the SystemSQL table.

```
SELECT * FROM Name WHERE firstname = Variables.firstliteral
```

The query is called in an element as follows:

```
<ICS.LITERAL
  TABLE="Name"
  COLUMN="firstname"
  STRING="Bubba"
  OUTPUT="firstliteral"/>
```

```
<ics.clearerrno/>
```

```
<CALLSQL QRYNAME="myquery" LIST="mylist"/>
```

In this example, Bubba is the string to be converted to a literal. The string is formatted according to the properties set for the database.

Release Introduced

This tag was introduced in CSEE release 5.0.

See Also

[CALLSQL](#)

ICS.QUERYEVENTS

Performs a query for a list of registered events.

Syntax

```
<ICS.QUERYEVENTS
  LIST="list name"
  [NAME="event name pattern"]
  [TYPE="event type"]
  [ENABLED="is enabled"]/>
```

Parameters

LIST (required)

Name of the output list.

NAME (optional)

Name of the event.

TYPE (optional)

Specifies if the event is a servlet event or an email event. Legal values are 1 to specify a servlet event and 2 to specify an email event.

ENABLED (optional)

Specifies whether the query runs on enabled events or disabled events. Legal values are true to query only enabled events or false to query only disabled events.

Description

This tag queries Content Server for a list of events that have been registered with using the [APPEVENT](#) and [EMAILEVENT](#) tags. The `ICS.QUERYEVENT` tag returns a list with the following columns:

- Name
- Type
- Enabled
- Time
- Target (for application events)
- Recipient (for email events)
- Params (for application events)
- Filename (for email events)

errno

The possible values of `errno` include:

Value	Description
-203	Event error
-10004	A required parameter is missing.
-10010	Invalid parameter value

Example

The following example lists all events:

```
<ICS.QUERYEVENTS LIST="events" ENABLED="true"/>
<LOOP LIST="events">
  CSVAR NAME="events.name"/> <BR/>
</LOOP>
```

Release Introduced

This tag was introduced in Content Server release 4.0.

See Also

[ICS.READEVENT](#)

ICS.READEVENT

Reads an event from storage.

Syntax

```
<ICS.READEVENT
  LIST="list name"
  NAME="event name"/>
```

Parameters

LIST (required)
Name of the output list.

NAME (required)
Name of the event.

Description

This tag reads an event from storage. In order for the event to be read it must have been created using the [APPEVENT](#) or [EMAILEVENT](#) tags. The ICS.READEVENT tag returns a list with the following columns:

- Name
- Type
- Enabled
- Time
- Target (for application events)
- Recipient (for email events)
- Params (for application events)
- Filename (for email events)

errno

The possible values of errno include:

Value	Description
-203	Event error
-10004	A required parameter is missing.
-10010	Invalid parameter value

Example

The following example reads the email event `confirmation_email` from storage and places the results in a list called `events`.

```
<ICS.READEVENT LIST="events" NAME="confirmation_email"/>
```

Release Introduced

This tag was introduced in Content Server release 4.0.

See Also

[ICS.QUERYEVENTS](#)

ICS.RESOLVEVARIABLES

Resolves Content Server multiple variables.

Syntax

```
<ICS.RESOLVEVARIABLES
  NAME="variable name"
  [OUTPUT="variable name"]
  [DELIMITED="true|false"] />
```

Parameters

NAME (required)

String with embedded variable references.

OUTPUT (optional)

Output variable name. If this parameter is not specified, the output is streamed.

DELIMITED (optional)

true indicates new syntax for resolving variables (see the following Description section); false indicates the old syntax.

Description

The new syntax for resolving variables uses a \$(suffix and a) prefix. For example, Variables.pagename becomes \$(Variables.pagename). This allows you to resolve variables that may be a prefix of another variable name. For example, Variables.listname.column becomes \$(\$(Variables.listname).column) to indicate that the listname is in the variable.

errno

The possible values of errno are:

Value	Description
-106	Invalid variable references.
-10004	A required parameter is missing.

Example

The following example resolves a variable called `Variables.Year` and streams the result to the browser.

```
<ICS.RESOLVEVARIABLES NAME="Variables.Year"/>
```

This example resolves the variables `Variables.lastname` and `Variables.firstname` then streams the result to the browser.

```
<ICS.RESOLVEVARIABLES  
NAME="Variables.lastname,Variables.firstname"/>
```

Release Introduced

This tag was introduced in CSEE release 5.0.

ICS.SQL

Executes an inline SQL statement.

Syntax

```
<ICS.SQL SQL="sql commands"  
LISTNAME="list name"  
TABLE="name of table"  
[LIMIT="max number of results"]/>
```

Parameters

SQL (required)

SQL commands to execute.

LISTNAME (required)

Name of list to contain the resultset.

TABLE (required)

Name of the Content Server table to query. The table must be registered in the SystemInfo table. This table verifies security and caches result sets when the `cc.cacheResults` property is enabled.

LIMIT (optional)

Maximum number of rows to be retrieved. If limit is omitted or if the limit value is set to less than or equal to 0, all records from the SQL command are returned.

Description

The `ICS.SQL` tag executes an inline SQL statement. It is the equivalent of the `EXECSQL` XML tag. The table specified by the `TABLE` must exist in the database prior to calling `ICS.SQL`. The resultset is cached against the table indicated by the value of the `TABLE` variable.

Note

If the limit is less than or equal to 0, then no limit exists and the entire recordset is returned.

errno

The possible values of `errno` include:

Value	Description
-101	No rows found.
-102	Empty column.
-103	No such table.
-104	No table definition.

ICS.SQLEXP

Allows building of the components of a SQL where clause without direct knowledge of the data type of a column and the database-specific rules for specifying a literal for that column.

Syntax

```
<ICS.SQLEXP
  COLUMN="column name"
  OUTPUT="variable name"
  STRING="string"
  TABLE="table name"
  [TYPE="logical operator"]
  VERB="comparison operator"
  [EXPRESSION="expression"]/>
```

Parameters

COLUMN (required)

Name of the column in the table.

OUTPUT (required)

Name of the output variable.

STRING (required)

Value or values to compare against.

TABLE (required)

Name of the table.

TYPE (optional)

Logical operator to use between. Default is AND.

VERB (required)

Comparison operator.

EXPRESSION (optional)

Column expression to be used instead of the column name.

errno

The possible values of `errno` include:

Value	Description
-100	Exception error
-103	Invalid table name
-106	Bad parameter value
-406	Bad parameter value
-10004	Missing required attribute

Release Introduced

This tag was introduced in CSEE release 5.0.

ICS.STREAMHEADER

Streams data from other tags.

Syntax

```
<ICS.STREAMHEADER NAME="Header Name"
                  VALUE="Header Value"/>
```

Parameters

NAME (required)
Name of the header.

VALUE (required)
Value of the header.

Description

This tag must be used to stream data from root elements or other tags that stream data.

errno

None.

Example

```
<ICS.STREAMHEADER NAME="Content-Type"
                  VALUE="text/html; charset=iso-8859-1"/>
```

See Also

[REMOVECOOKIE](#)
[SETCOOKIE](#)

IF

Processes other tags or displays text based on a logical condition.

Syntax

```
<IF COND="LOGICAL_EXPRESSION">
  <THEN>
    tags and/or text
  </THEN>
  <ELSE>
    tags and/or text
  </ELSE>
</IF>
```

Parameters

COND (required)

String comparison of two string values. The operators supported are "=" (values are equivalent) or "!=" (values are not equivalent)".

The following are conditions that Content Server provides for use with the IF tag; all these subconditions return either `true` or `false`:

IsCatalog	Tests whether this is a table.
IsColumn	Tests for the existence of a column in a list.
IsCounter	Tests for the existence of a counter.
IsElement	Tests for the existence of an element.
IsError	Tests whether the <code>errno</code> is actually an error.
IsList	Tests for the existence of a list.
IsObject	Tests whether the given name is an object table.
IsSessionVariable	Tests for the existence of a session variable.
IsTracked	Determine whether Content Server is revision tracking a specified table.
IsTree	Tests whether the given name is a tree table.
IsVariable	Tests for the existence of a variable.

Description

The IF tag processes other tags or displays text based on a logical condition. If the condition is `true`, the content of the THEN tag is processed. If the condition is `false`, the content of the ELSE tag is processed. The THEN tag is required as part of an IF tag. The ELSE tag is not required.

errno

The value of `errno` is unchanged.

Example

The following example prints the message `Color is red.` if the value of `Variables.color` is red.

```
<IF COND="Variables.color=red">
  <THEN>
    Color is red.<br/>
  </THEN>
  <ELSE>
    Color is not red.<br/>
  </ELSE>
</IF>
```

This example tests for the existence of the variable, `color`. If a variable named `color` has been set prior to this test, The `variable color exists.` is displayed.

```
<IF COND="IsVariable.color=true">
  <THEN>
    The variable color exists.<br/>
  </THEN>
</IF>
```

INCCOUNTER

Increments or decrements a counter.

Syntax

```
<INCCOUNTER NAME="COUNTER_NAME"
  VALUE="INCREMENTAL_VALUE"/>
```

Parameters

NAME (required)

Name of the counter to increment

VALUE (required)

Integer value to increment/decrement counter. A negative value decrements the counter.

errno

The possible values of `errno` include:

Value	Description
-501	Bad counter value.

Example

This example creates a counter "fred" and performs two different counter operations on fred.

```
<SETCOUNTER NAME="fred" VALUE="100"/>
<INCCOUNTER NAME="fred" VALUE="200"/>
  The value of Fred is <CSVAR NAME="Counters.fred"/>
  <!-- The value of Fred is 300 -->
<INCCOUNTER NAME="fred" VALUE="-50"/> <br/>
  The value of Fred is <CSVAR NAME="Counters.fred"/>
  <!-- The value of Fred is 250 -->
<INCCOUNTER NAME="fred" VALUE="-300"/>
  The value of Fred is <CSVAR NAME="Counters.fred"/>
  <!-- The value of Fred is -50 -->
```

See Also

[SETCOOKIE](#)

[REMOVECOUNTER](#)

INDEXADD

Adds an entry to an existing search index.

Syntax

```
<INDEXADD
  ENTRY="ENTRY_NAME"
  [ INDEX="INDEX_NAME" ]
  [ DETAIL="ENTRY_DETAIL " ]
  [ WORDLIST="WORDS" ]
  [ WORDLISTDELIMITERS="DELIMITERS" ]
  [ FILELIST="FILES" ]
  [ DATE="DATE_STRING" ]
  [ CHARACTERSET="CHAR_SET" ]
  [ SEARCHENGINE="NAME" ]
  <FILEARGUMENT NAME="FILEARG_NAME"
    VALUE="FILEARG_VALUE"/>
  <DATEARGUMENT NAME="DATEARG_NAME"
    VALUE="DATEARG_VALUE"/>
  <TEXTARGUMENT NAME="TEXTARG_NAME"
    VALUE="TEXTARG_VALUE"/>
</INDEXADD>
```

Parameters

ENTRY (required)

Name of the index entry to add. This value is returned in the search results. The ENTRY value must be unique within an index. Choose ENTRY values carefully so that the caller of the search can reference the associated article(s).

INDEX (optional)

Name of the index to which the ENTRY should be added. If this parameter is not specified, the value of the Content Server property `av.defaultindex` is used.

DETAIL (optional)

Detail text returned in search results.

WORDLIST (optional)

Words to add to the index. These are whitespace-separated by default but you can use the `WORDLISTDELIMITERS` parameter to specify something else. Use this parameter to add text that is not associated with a field of the index.

WORDLISTDELIMITERS (optional)

String containing delimiter characters to use to separate words in the value of the `TEXTARGUMENT` tag and the `WORDLIST` parameter. White space is used as delimiter characters if `WORDLISTDELIMITERS` is not specified.

FILELIST (optional)

For AltaVista, specify a comma-separated list of files to add. For Verity, specify a single file (not a list of files). Use this parameter to add text that is not associated with a field of the index.

DATE (optional)

Date string to add to the index entry. If no `DATE` is specified, the `INDEXADD` time is returned in the search results. Format is in Java SQL.

CHARACTERSET (optional)

Constant value representing the character set the index uses. For the AltaVista search engine this value may be 0, 1, or 2 (`ISO_LATIN1`, `ASCII8`, `UTF8`). If you do not specify `CHARACTERSET`, Content Server uses the values of `av.charset` from the Content Server the properties file.

For the Verity search engine, this value specifies the name of the subdirectory of the common directory where the locale is defined. If you do not specify `CHARACTERSET`, Content Server uses the value of `verity.charset` in the Content Server properties file.

SEARCHENGINE (optional)

Name of the search engine to use. If `SEARCHENGINE` is not specified, Content Server uses the value of `cs.searchengine` from the Content Server properties file.

Description

The `INDEXADD` tag adds an entry to an existing search index.

For both AltaVista and Verity, arguments are optional using `FILEARGUMENT`, `DATEARGUMENT` and `TEXTARGUMENT` tags. These arguments specify the index field (the `NAME` parameter) to which the data (the `VALUE` parameter) is to be added. When using the `FILEARGUMENT` value with AltaVista, specify a comma-separated list of files to add to the index with the given name. When using the `FILEARGUMENT` value with Verity, specify a single file (not a list of files). Use this parameter to add text that is not associated with a field of the index. The `DATEARGUMENT` is a date string specified in `JAVASQL` format. The `TEXTARGUMENT` value is a string delimited by white space. The white-space delimiter characters may be overridden by the `WORDLISTDELIMITERS` tag.

The search engine collects arguments with the same name into a concatenated single value for that field.

errno

The possible values of `errno` include:

Value	Description
-300	File not found.
-801	Cannot load search engine.
-804	Add index failed.
-805	No default index specified.
-806	Unknown search engine.
-811	Could not call native method.
-812	Index does not exist.
-815	Duplicate field values. (Verity only)
-816	Could not create temporary populate file. (Verity only)
-817	Could not create temporary data file. (Verity only)

Example

This example adds an entry (Article 083742132) to the Article index.

```
<SETVAR NAME="errno" VALUE="0"/>
<INDEXADD INDEX="c:/ArticleIdx"
    SEARCHENGINE="AV"
    ENTRY="Article083742132">
    <FILEARGUMENT NAME="body" VALUE="c:/articledata/
083742132.body"/>
    <FILEARGUMENT NAME="body"
        VALUE="c:/articledata/083742132/*.txt"/>
    <TEXTARGUMENT NAME="headline"
        VALUE="Clinton to vacation in Massachusetts."/>
</INDEXADD>
<IF COND="IsError.Variables.errno=true">
    <THEN>
        <!--Handle error-->
    </THEN>
</IF>
```

See Also

[INDEXCREATE](#)
[INDEXREMOVE](#)

INDEXCREATE

Creates a search index.

Syntax

```
<INDEXCREATE
  INDEX="INDEX_NAME"
  [FIELDARGUMENT="VERITY_SUBFIELD"]
  [CHARACTERSET="CHAR_SET"]
  [SEARCHENGINE="NAME"]/>
```

Parameters

INDEX (required)

The full path name of the index to create is required in the tag unless the properties `av.defaultindex` or `verity.defaultindex` have been specified in the `futuretense.ini` file.

FIELDARGUMENT (optional, used by Verity only)

Defines subfields of a Verity search index. Index subfields must be defined when `INDEXCREATE` is called. Valid `FIELDARGUMENT` types are `DATE` and `TEXT`.

CHARACTERSET (optional)

Constant value representing the character set the index uses. For the AltaVista search engine this value may be 0, 1, or 2 (`ISO_LATIN1`, `UTF8`, `ASCII8`). If you do not specify `CHARACTERSET`, Content Server uses the value of `av.charset` in the Content Server properties file.

Because Java Strings are transformed to UTF8, as they pass through the JNI interface to AltaVista, the 0 value only works for 7-bit ASCII. For all other cases, use the value for UTF8.

For the Verity search engine, this value specifies the name of the subdirectory of the common directory where the locale is defined. Note that case-sensitivity is important with directory names. If you do not specify `CHARACTERSET`, Content Server uses the value of `verity.charset` in the Content Server properties file.

SEARCHENGINE (optional)

Name of the search engine to use. If `SEARCHENGINE` is not specified, Content Server uses the value of `cs.searchengine` from the Content Server properties file.

Description

The `INDEXCREATE` tag creates a search index. For AltaVista, this creates the directory where the index is stored. For Verity, however, this creates a more complex base structure for the index in the directory specified by the `INDEX` parameter. In particular, it needs to create the structures that hold the field information for the index. Therefore, the fields that are used must be specified using `FIELDARGUMENT` tags when the index is created.

errno

The possible values of `errno` include:

Value	Description
-801	Cannot load search engine.
-802	Unsupported search function. Content Server 3.0 and earlier returned this error; however, Content Server 3.1 and higher no longer return this error.
-803	Create index failed.
-805	No default index specified.
-806	Unknown search engine.
-811	Could not call native method.
-815	Duplicate field values. (Verity only)
-819	Unrecognized field type. (Verity only)
-822	Index already exists.(Verity only)

Example

This AltaVista example creates the Article index used to save article information. The index is stored as `C:\ArticleIdx.avx`, but you don't specify the `.avx` extension in the tag.

```
<INDEXCREATE INDEX="C:\ArticleIdx" SEARCHENGINE="AV"/>
```

The following Verity example creates the Article index used to save article information. The index is stored as `C:\ArticleIdx.vix`, but you don't specify the `.vix` extension in the tag. Subfields are specified for headlines and article dates.

```
<SETVAR NAME="errno" VALUE="0"/>
<INDEXCREATE INDEX="C:\ArticleIdx" SEARCHENGINE="VERITY">
  <FIELDARGUMENT NAME="Headline" VALUE="TEXT"/>
  <FIELDARGUMENT NAME="ArticleDate" VALUE="DATE"/>
</INDEXCREATE>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>
```

See Also

[INDEXEXISTS](#)
[INDEXDESTROY](#)

INDEXDESTROY

Deletes an existing search index.

Syntax

```
<INDEXDESTROY INDEX="INDEX_NAME"
    [CHARACTERSET="CHAR_SET"]
    [SEARCHENGINE="NAME"] />
```

Parameters

INDEX (required)

Full path name of the index to delete. If you specify a nonexisting *INDEX_NAME*, INDEXDESTROY does not return an error.

CHARACTERSET (optional)

Constant value representing the character set the index uses.

For the AltaVista search engine this value may be 0, 1, or 2 (ISO_LATIN1, ASCII8, UTF8). If you do not specify CHARACTERSET, Content Server uses the value of `av.charset` in the Content Server properties file.

Because Java Strings are transformed to UTF8, as they pass through the JNI interface to AltaVista, the 0 value only works for 7-bit ASCII. For all other cases, use the value for UTF8.

For the Verity search engine, this value specifies the name of the subdirectory of the common directory where the locale is defined. If you do not specify CHARACTERSET, Content Server uses the value of `verity.charset` in the Content Server properties file.

SEARCHENGINE (optional)

Name of the search engine to use. If SEARCHENGINE is not specified, Content Server uses the value of `cs.searchengine` from the Content Server properties file.

Description

The INDEXDESTROY tag deletes an existing search index.

If you call INDEXEXISTS after deleting an index, INDEXEXISTS returns -812 (indicating that the index no longer exists).

For AltaVista, after calling INDEXDESTROY to delete an existing index, the base directory specified in the INDEX parameter usually still exists and contains an empty lock file. In this case, INDEXEXISTS still reports, as expected, that the index does not exist.

For Verity, after calling INDEXDESTROY to delete an existing index, the entire directory structure is left.

Note

The INDEXDESTROY tag deletes indexes only. Directories that contain the indexes must be deleted manually for both AltaVista and Verity .

errno

The possible values of `errno` include:

Value	Description
-801	Cannot load search engine.
-804	Delete index failed.
-805	No default index specified.
-806	Unknown search engine.

Example

This example deletes the Article index.

```
<SETVAR NAME="errno" VALUE="0"/>
<INDEXDESTROY INDEX="C:\ArticleIdx"
    SEARCHENGINE="AV"/>
<IF COND="IsError.Variables.errno=true">
    <THEN>
        <!--Handle error-->
    </THEN>
</IF>
```

See Also

[INDEXCREATE](#)

INDEXEXISTS

Use this tag to check for the existence of an index.

Syntax

```
<INDEXEXISTS INDEX="INDEX_NAME"
    [CHARACTERSET="CHAR_SET"]
    [SEARCHENGINE="NAME"]/>
```

Parameters

INDEX (required)

Full path name of the index to check.

CHARACTERSET (optional)

Constant value representing the character set the index uses.

For the AltaVista search engine this value may be 0, 1, or 2 (ISO_LATIN1, ASCII8, UTF8). If you do not specify CHARACTERSET, Content Server uses the value of `av.charset` in the Content Server properties file.

Because Java Strings are transformed to UTF8, as they pass through the JNI interface to AltaVista, the 0 value only works for 7-bit ASCII. For all other cases, use the value for UTF8.

For the Verity search engine, this value specifies the name of the subdirectory of the common directory where the locale is defined. If you do not specify CHARACTERSET, Content Server uses the value of `verity.charset` in the Content Server properties file.

SEARCHENGINE (optional)

Name of the search engine to use. If SEARCHENGINE is not specified, Content Server uses the value of `cs.searchengine` from the Content Server properties file.

errno

The possible values of errno include:

Value	Description
9	Index exists.
-806	Unknown search engine.
-812	Index does not exist.

Example

The following example checks for the existence of `c:\ArticleIdx` and creates it if it doesn't exist.

```
<SETVAR NAME="errno" VALUE="0"/>
<INDEXEXISTS INDEX="c:/ArticleIdx"
              SEARCHENGINE="AV"/>
<IF COND="Variables.errno!=9">
  <THEN>
    <INDEXCREATE INDEX="c:/ArticleIdx"
                  SEARCHENGINE="AV"/>
  </THEN>
</IF>
```

See Also

[INDEXCREATE](#)

INDEXOF

Finds the index of a string within a string.

Syntax

```
<INDEXOF STR="STRING_VALUE"  
  WHAT="SEARCH_STRING"  
  OUTSTR="VAR_NAME"  
  INDEX="OFFSET"/>
```

Parameters

STR (required)

String to search.

WHAT (required)

Value of the string to search for within *STRING_VALUE*.

OUTSTR (required)

Name of the variable to store the offset into *STRING_VALUE* of the *SEARCH_STRING*.

INDEX (required)

Offset in the source string to begin the search. The default for this parameter is 0.

Description

The INDEXOF tag finds the index of a specified string within another string. For more information about variables, see the *CSEE Developer's Guide*.

errno

The possible values of errno include:

Value	Description
1	Substring was found.
0	Substring was not found.

Example

This example looks for the index of test within the string, "This is a test".

```
<INDEXOF STR="This is a test" WHAT="test" OUTSTR="index"  
  INDEX="0"/>  
  <IF COND="Variables.errno=1">  
    <THEN>  
      <CSVAR NAME="Variables.index"/>  
    </THEN>  
  </IF>
```

The browser displays:

10

The following example sets the outstr variable to -1 if the string isn't found:

```

<SETVAR NAME="errno" VALUE="0"/>
<INDEXOF STR="a lost dog found the way home to Boston from Alaska"
    WHAT="Florida"
    OUTSTR="myindex1"
    INDEX="0"/>
<SETVAR NAME="myindex1"
    VALUE="errno:Variables.errno, outstr:Variables.myindex1"/>

```

The value of myindex1 is:

```
"errno:0, outstr: -1"
```

See Also

[BEGINS](#), [ENDS](#)

INDEXREMOVE

Removes an entry from an existing search index.

Syntax

```

<INDEXREMOVE
    ENTRY="ENTRY_NAME"
    [INDEX="INDEX_NAME"]
    [CHARACTERSET="CHAR_SET"]
    [SEARCHENGINE="NAME"]/>

```

Parameters

ENTRY (required)

Name of the index entry to remove.

INDEX (optional)

Full path name of the index from which the `ENTRY_NAME` should be removed. If this parameter is not specified, the value of the Content Server property `av.defaultindex` is used.

CHARACTERSET (optional)

Constant value representing the character set the index uses.

For the AltaVista search engine this value may be 0, 1, or 2 (ISO_LATIN1, ASCII8, UTF8). If you do not specify `CHARACTERSET`, Content Server uses the value of `av.charset` in the Content Server properties file.

Because Java Strings are transformed to UTF8, as they pass through the JNI interface to AltaVista, the 0 value only works for 7-bit ASCII. For all other cases, use the value for UTF8.

For the Verity search engine, this value specifies the name of the subdirectory of the common directory where the locale is defined. If you do not specify `CHARACTERSET`, Content Server uses the value of `verity.charset` in the Content Server properties file.

SEARCHENGINE (optional)

Name of the search engine to use. If `SEARCHENGINE` is not specified, Content Server uses the value of `cs.searchengine` from the Content Server properties file.

errno

The possible values of `errno` include:

Value	Description
-801	Cannot load search engine.
-804	Remove index failed.
-805	No default index specified.
-806	Unknown search engine.
-811	Could not call native method.
-812	Index does not exist.

Example

This example removes Article 083742132 from the index, `c:/ArticleIdx`:

```
<SETVAR NAME="errno" VALUE="0"/>
<INDEXREMOVE INDEX="c:/ArticleIdx"
    SEARCHENGINE="AV"
    ENTRY=" Article083742132"/>
<IF COND="IsError.Variables.errno=true">
    <THEN>
        <!--Handle error-->
    </THEN>
</IF>
```

See Also

[INDEXADD](#)
[INDEXCREATE](#)
[INDEXDESTROY](#)

INDEXREPLACE

Replaces an existing entry in a search index.

Syntax

```
<INDEXREPLACE
  ENTRY="ENTRY_NAME"
  INDEX="INDEX_NAME"
  DETAIL="ENTRY_DETAIL "
  [WORDLIST="WORDS"]
  [WORDLISTDELIMITERS="DELIMITERS"]
  [FILELIST="FILES"]
  [DATE="DATE_STRING"]
  [CHARACTERSET="CHAR_SET"]
  [SEARCHENGINE="NAME">]
  <FILEARGUMENT NAME="FILEARG_NAME"
    VALUE="FILEARG_VALUE"/>
  <DATEARGUMENT NAME="DATEARG_NAME"
    VALUE="DATEARG_VALUE"/>
  <TEXTARGUMENT NAME="TEXTARG_NAME"
    VALUE="TEXTARG_VALUE"/>
</INDEXREPLACE>
```

Parameters

WORDLIST (optional)

Words to add to the index. These are whitespace-separated by default but you can use the WORDLISTDELIMITERS parameter to specify something else. Use this parameter to replace text that is not associated with a field of the index.

WORDLISTDELIMITERS (optional)

String containing delimiter characters to use to separate words in the value of the TEXTARGUMENT tag and the WORDLIST parameter. White space is used as delimiter characters if WORDLISTDELIMITERS is not specified.

FILELIST (optional)

For AltaVista, specify a comma-separated list of files to add. For Verity, specify a single file (not a list of files). Use this parameter to replace text that is not associated with a field of the index.

DATE (optional)

Date string to replace to the index entry. If no DATE_STRING is specified, the INDEXADD time is returned in the search results. Format is in Java SQL.

CHARACTERSET (optional)

Constant value representing the character set the index uses. For the AltaVista search engine this value may be 0, 1, or 2 (ISO_LATIN1, ASCII8, UTF8). If you do not specify CHARACTERSET, Content Server uses the value of av.charset from the Content Server properties file. Because Java Strings are transformed to UTF8, as they pass through the JNI interface to AltaVista, the 0 value only works for 7-bit ASCII. For all other cases, use the value for UTF8. For the Verity search engine, this value specifies the name of the subdirectory of the common directory where the locale is defined. If you do not specify CHARACTERSET, Content Server uses the value of verity.charset in the Content Server properties file.

SEARCHENGINE (optional)

Name of the search engine to use. If SEARCHENGINE is not specified, Content Server uses the value of `cs.searchengine` in the Content Server properties file.

Description

The INDEXREPLACE tag replaces an existing entry in a search index.

Arguments are optional using FILEARGUMENT, DATEARGUMENT and TEXTARGUMENT tags. These arguments specify the index field (the NAME parameter) to which the data (the VALUE parameter) is to be added to. The FILEARGUMENT value is a comma separated list of files whose content is added to the index with the given name. The DATEARGUMENT is a date string specified in JAVASQL format. The TEXTARGUMENT value is a string delimited by white space. The white-space delimiter characters may be overridden by the WORDLISTDELIMITERS tag.

The search engine collects arguments with the same name into a concatenated single value for that field.

errno

The possible values of errno include:

Value	Description
-300	File not found.
-801	Cannot load search engine.
-804	Index remove failed.
-805	No default index specified.
-806	Unknown search engine.
-811	Could not call native method.
-812	Index does not exist.
-815	Duplicate field values. (Verity only)
-816	Could not create temporary populated file. (Verity only)
-817	Could not create temporary data file. (Verity only)

Example

This example replaces the search information for Article083742132.

```
<SETVAR NAME="errno" VALUE="0"/>
<INDEXREPLACE INDEX="c:/ArticleIdx"
  SEARCHENGINE="AV" ENTRY="Article083742132">
  <FILEARGUMENT NAME="body"
    VALUE="c:/articledata/083742132.body"/>
  <FILEARGUMENT NAME="body"
    VALUE="c:/articledata/083742132/*.txt"/>
  <TEXTARGUMENT NAME="headline"
    VALUE="Clinton to vacation in New Jersey."/>
```



```

</INDEXREPLACE>
<IF COND="IsError.Variables.errno=true">
    <THEN>
        <!--Handle error-->
    </THEN>
</IF>

```

See Also

[INDEXADD](#)

INSERT

Inserts the content of a file or URL into the current output stream or sets the content into a variable.

Syntax

```

<INSERT [FILE="FILE_NAME"]
        [URL="URL "]
        [NAME="VAR_NAME"] />

```

Parameters

FILE (optional)

Name of file to insert into the current output.

URL (optional)

URL to insert into the current output.

NAME (optional)

Name of the variable to set with the content of FILE or URL.

Description

The INSERT tag inserts the content of a file or URL into the current output stream or sets the content into a variable. In addition, it supports ARGUMENT constructs for compound URL INSERT tags. For example, when an INSERT tag specifies URL (as opposed to FILE), the arguments supplied are attached to the URL line as follows:

- A question mark (?) is appended to the URL value.
- The argument name/value is appended as name=value for each argument supplied
- All white space in the URL is replaced with %20 to make it appropriate for HTTP transport to a browser.

errno

The possible values of errno include:

Value	Description
-300	File not found.
-303	Path not found.

Example

This example inserts a file whose path is constructed by concatenating a property from the Content Server properties file and the value of the variable "prolog". The content of the file is inserted into the output.

```
<SETVAR NAME="prolog"
        Value="CS.Property.PrologDir"/>
<INSERT FILE="Variables.prolog/ContentServer"/>
```

This example sets the content of a relative URL to a variable, formbits, then displays the value of the variable.

```
<INSERT URL="/futuretense_cs/formbase.html"
        NAME="formbits"/>
<CSVAR NAME="Variables.formbits"/>
```

See Also

[CSVAR](#)

IsCatalog

Tests a list for the existence of a table (known previously as a "catalog" in earlier versions of the product).

Syntax

`IsCatalog.catalogName`

Returns

Returns true if the table exists; returns false if the table does not exist.

Example

The following code determines whether PotentialCatalog is indeed a table:

```
<IF COND="IsCatalog.PotentialCatalog=true">
  <THEN>
    table exists.<br/>
  </THEN>
  <ELSE>
    table does not exist.<br/>
  </ELSE>
</IF>
```

IsColumn

Tests for the existence of a column in a list.

Syntax

`IsColumn.listName.columnName`

Returns

Returns true if the column exists; returns false if the counter does not exist.

Example

```
<SELECTTO FROM="mytable" LIST="mylist" WHAT="*" />
<IF COND="IsColumn.mylist.title=true">
  <THEN>
    Column exists.<br/>
  </THEN>
  <ELSE>
    Column does not exist.<br/>
  </ELSE>
</IF>
```

IsCounter

Tests for the existence of a counter.

Syntax

`IsCounter.counterName`

Returns

Returns true if counter exists; returns false if the counter does not exist.

Example

```
<SETCOUNTER NAME="mycounter" VALUE="5" />
<IF COND="IsCounter.mycounter=true">
  <THEN>
    Counter exists.<br/>
  </THEN>
  <ELSE>
    Counter does not exist.<br/>
  </ELSE>
</IF>
```

IsElement

Tests for the existence of an element.

Syntax

`IsElement.elementName`

Returns

Returns `true` if the element exists; returns `false` if the element does not exist.

Example

```
<IF COND="IsElement.OpenMarket/Apps/main=true">
  <THEN>
    Element exists.<br/>
  </THEN>
  <ELSE>
    Element does not exist.<br/>
  </ELSE>
</IF>
```

IsError

Tests the `errno` variable for a negative value.

Syntax

`IsError.Variables.errno`

Returns

Returns `true` if the `errno` should be considered an error; returns `false` if the `errno` should not be considered an error.

Example

```
<SELECT TO FROM "mytable" LIST="mylist"/>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    Oops, theres an error.<br/>
  </THEN>
  <ELSE>
    Im okay, youre okay.<br/>
  </ELSE>
</IF>
```

ISINLIST

Determines whether an item is in a comma-separated string or string list.

Syntax

```
<ISINLIST ITEM="ITEM_NAME"
          STR="STR_VALUE"
          [LIST="LIST_NAME"]/>
```

Parameters

ITEM (required)

Specify the item to check for in `LIST_NAME`.

STR (required)

String to check for occurrence of `ITEM_NAME`. The string consists of a comma-separated list of items.

LIST (optional)

List to check for occurrence of `ITEM_NAME`. If `LIST_NAME` is not a list, an error is inserted into the output and processing may stop if `xmldebug=true` in the Content Server properties file.

Description

Determines whether an item is in a comma-separated string or string list. If `STR` or `LIST` is not specified or is empty, the item cannot be found and the value of `errno` is unchanged.

Note

This does not work on lists created from resultsets or search results.

The column values of the current row of a list can be obtained using the syntax, `List.colname` where *List* is the name of the list and *colname* is the name of the column in the list.

Note

In the case of *StringLists*, the values can be obtained by using *List.Item*.

You can use the following attributes to return information about the list using the syntax, `List.attname`, where *attname* can be one of the following attributes:

#numRows	Number of rows in this list
#curRow	Current row in this list.
#moreRows	Boolean (true/false) to indicate whether there are more rows following the current row.
#numCols	Number of columns in this list.

@<colname>	Only used on lists that contain resultsets. Contents of the file in the specified column if the colname begins with "url".

For example, `MyList.#numRows` returns the number of rows in the list `MyList`. `MyList.@urlheader` retrieves the contents of the file specified by the column `urlheader`.

The value of a list is maintained during the processing of a page request. If the list contains a resultset, the resultset is maintained independently of a list that uses it.

errno

The possible values of `errno` include:

Value	Description
1	Item was found.
0	Item was not found.

Example

The following example determines whether the string assigned to `MyItem` is one of the elements in `MyStr`:

```
<SETVAR NAME="MyItem" VALUE="dog"/>
<SETVAR NAME="MyStr" VALUE="horse,dog,monkey"/>

<SETVAR NAME="errno" VALUE="0"/>
<ISINLIST
  ITEM="Variables.MyItem"
  STR="Variables.MyStr" />
<IF COND="Variables.errno=1">
  <THEN>
    <CSVAR NAME="Variables.MyItem"/> is in string.
  </THEN>
  <ELSE>
    <CSVAR NAME="Variables.MyItem"/> is not in string.
  </ELSE>
</IF>
```

Running this code yields the following output:

```
dog is in string.
```

See Also

[STRINGLIST](#)

IsList

Tests for the existence of a specific list.

Syntax

`IsList.listName`

Returns

Returns `true` if the list exists; returns `false` if the list does not exist.

Example

```
<SELECTTO FROM="mytable" LIST="mylist"/>
<IF COND="IsList.myList=true">
  <THEN>
    The list exists.<br/>
  </THEN>
  <ELSE>
    The list does not exist.<br/>
  </ELSE>
</IF>
```

IsObject

Tests if the given name is an object table.

Syntax

`IsObject.objName`

Returns

Returns `true` if the name is an object table; returns `false` if the name is not an object table.

Example

```
<SETVAR NAME="tablename" VALUE="section"/>
<IF COND="IsObject.Variables.tablename=true">
  <THEN>
    This is an object table.<br/>
  </THEN>
  <ELSE>
    Sorry, not an object table.<br/>
  </ELSE>
</IF>
```

IsSessionVariable

Tests for the existence of a session variable.

Syntax

`IsSessionVariable.varName`

Returns

Returns `true` if the session variable exists; returns `false` if the session variable does not exist.

Example

```
<SETSSVAR NAME="profile" VALUE="whatever"/>
<IF COND="IsSessionVariable.profile=true">
  <THEN>
    Profile exists.<br/>
  </THEN>
  <ELSE>
    Profile does not exist.<br/>
  </ELSE>
</IF>
```

IsTracked

Determines whether a given table is revision tracked.

Syntax

`IsTracked.catalogName`

Returns

Returns `true` if Content Server is tracking revisions on the specified table; returns `false` otherwise.

Example

The following code determines whether MyCatalog is being revision tracked:

```
<IF COND="IsTracked.MyCatalog=true">
  <THEN>
    Content Server is tracking revisions on this table.<br/>
  </THEN>
  <ELSE>
    Content Server is not tracking revisions on this table.<br/>
  </ELSE>
</IF>
```

IsTree

Tests if the given name is a tree table.

Syntax

`IsTree.treeName`

Returns

Returns `true` if the name is a tree table; returns `false` if the name is not a tree table.

IsVariable

Tests for the existence of a variable.

Syntax

`IsVariable.varName`

Returns

Returns `true` if the variable exists; returns `false` if the variable does not exist.

LOADPROPERTY

Loads a property file.

Syntax

`<LOADPROPERTY FILE="PROPERTY_FILE"/>`

Parameters

FILE (required)

Name of the property file(s) to load. Load multiple files by separating them with a semicolon, for example:
`futuretense.ini;mytest.ini;`

Note

If multiple property files are loaded with the same property in them, the last property that is loaded takes precedence.

Description

The `LOADPROPERTY` tag loads a property file. This tag can load multiple property files using a semicolon to separate the files. The values loaded from the property file are valid only during the processing of the page containing `LOADPROPERTY`. Use this command to access a table in a different database, or to load string properties. For example, you might load a property file to find application-specific values for the local system configuration.

The property file must be located in the same folder as the Content Server property file (`futuretense.ini`).

Note

Loading a property that does not contain the string `futuretense.ini` in the property file disables Content Server security by setting the `CS.Security` property to `false`. Content Server security remains disabled until the `RESTOREPROPERTY` XML tag is called.

errno

The possible values of `errno` include:

Value	Description
-106	Required property is missing.
-300	File not found.
-303	Path not found.

If one of these errors occurs, the values from the previously loaded property file (probably the default) are still in effect. There should be no need to use the `RESTOREPROPERTY` tag.

Example

This example loads a new property file to access a table in a different database. The property file must contain all the information required to connect to the new database (For example, dsn, user name/password set, and so on.).

```
<SETVAR NAME="errno" VALUE="0"/>
<LOADPROPERTY FILE="other.ini"/>
<SELECTTO LIST="numbers" WHAT="*"
  FROM="NumberData"/>
  after select errno is:
<CSVAR NAME="Variables.errno"/>
```

The other `.ini` property file must define at least the following properties:

- `ft.debug`
- `cs.readuser`
- `cs.password`
- `cs.privuser`
- `cs.privpassword`
- `cs.dsn`
- `cs.db`
- `cc.contentkey`
- `cc.primary`
- `cc.varchar255`

- `cc.bigtext`
- `cc.datetime`
- `cc.unique`
- `ft.cgipath`
- `cc.smallint`
- `cc.integer`
- `cc.bigint`
- `cc.double`
- `cc.driver`
- `cc.driverclass`

See Also

[RESTOREPROPERTY](#)

LOCALESTRING.ARGUMENT

Declares a name/value pair that is available to the parent tag, [LOCALESTRING.GETSTRING](#), as a variable. To have the variable value substituted in the string, use the syntax `$(Variables.name)` in the string definition.

Syntax

```
<LOCALESTRING.ARGUMENT NAME="name1"
    VALUE ="value1" />
<LOCALESTRING.ARGUMENT NAME="nameN"
    VALUE ="valueN" />
```

Parameters

NAME (required)
Name of the argument.

VALUE (required)
Value of the argument.

errno

None.

Release Introduced

This tag was introduced in CSEE release 5.0.

See Also

[LOCALESTRING.GETSTRING](#)

LOCALESTRING.GETSTRING

Returns a localized value for a string based on the LOCALE value.

Syntax

```
<LOCALESTRING.GETSTRING
  NAME="name of string"
  LOCALE="locale"
  [OUTPUT="variable name"]
  [RESOLVE="true"]>
<LOCALESTRING.ARGUMENT NAME="name1" VALUE ="value1" />
<LOCALESTRING.ARGUMENT NAME="nameN" VALUE ="valueN" />
</LOCALESTRING.GETSTRING>
```

Parameters

NAME (required)

Defines string name. FatWire recommends using a naming convention to make further localization easier.

LOCALE (required)

Five-character locale name, for example, EN_US.

OUTPUT (optional)

If specified, a variable is created by the requested name with the locale-specific value, otherwise, the value is streamed.

RESOLVE (optional)

If set to `true`, this parameter specifies that the localized value has variables that should be replaced using the rules for recursive Content Server variable replacement (`$(Variables.name1)` syntax).

errno

The possible values of `errno` include:

Value	Description
-103	Missing or invalid SystemLocaleString table
-10004	Missing required attribute

Release Introduced

This tag was introduced in CSEE release 5.0.

See Also

[LOCALESTRING.ARGUMENT](#)

LOCK

Locks a record for the current user in a tracked table.

Syntax

```
<LOCK TABLE="TABLE_NAME"
  ASSET="PRIMARY_KEY"/>
```

Parameters

TABLE (required)

Name of the tracked table.

ASSET (required)

Value of the primary key for the row to lock.

Description

The LOCK tag locks a record for the current user in a tracked table. Once a row is locked, the current user can modify fields in the row. The current user must have write access to the table.

errno

The possible values of errno include:

Value	Description
-1000	Command failure. Check the errdetail variable for the specific error.

Example

This example assumes all LOCK parameters are exposed on an HTML form. It first checks whether the parameters exist, clears any values from errno, and then locks the row.

```
<!-- Make sure required parameters exist -->
<IF COND="IsVariable.tablename=false">
  <THEN>
    <SETVAR NAME="tablename" VALUE=""/>
  </THEN>
</IF>
<IF COND="IsVariable.asset=false">
  <THEN>
    <SETVAR NAME="asset" VALUE=""/>
  </THEN>
</IF>
<!-- clear out errno -->
<SETVAR NAME="errno" VALUE="0"/>
<!-- do the lock -->
<LOCK TABLE="Variables.tablename"
  ASSET="Variables.asset"/>
<!-- See if lock succeeded -->
```

```
<IF COND="Variables.errno=0">
  <THEN>
    <!-- if succeeded, note it-->
    Lock succeeded for
    <CSVAR NAME="Variables.asset"/>
    <BR/>
  </THEN>
  <ELSE>
    <!-- otherwise, call an error reporting element -->
    Lock failed.<BR/>
  </ELSE>
</IF>
```

See Also

[UNLOCKRECORD](#)

LOGMSG

Writes strings to a Content Server log file for debugging or system monitoring.

Syntax

```
<LOGMSG STR="text"/>
```

Parameters

STR (required)

Text of the message to write to the log file.

errno

The value of errno is unchanged.

Example

The following example writes the error number to the log file:

```
<IF COND="Variables.errno!= 0" >
  <THEN>
    <LOGMSG STR="Error Variables.errno " />
  </THEN>
</IF>
```

LOOP

Modifies the list's current row value.

Syntax

```
<LOOP [FROM="START"]
      [COUNT="LOOP_TIMES"]
      [LIST="LIST_NAME"]
      [UNTIL="END"]>
  ...
</LOOP>
```

Parameters

FROM (optional)

Row to begin the loop.

COUNT (optional)

Number of times to loop. If both **COUNT** and **FROM/UNTIL** are specified, the condition that is true first terminates the loop.

LIST (optional)

List to loop through.

UNTIL (optional)

Compare two values. The operators supported are "=" (values are equivalent) or "!=" (values are not equivalent).

Description

The **LOOP** tag modifies the list's current row value.

If the **UNTIL** condition never evaluates to true, the **UNTIL** condition is ignored and **LOOP** continues to the last item in the list.

The column values of the current row in a list can be obtained using the syntax, *List.colname* where *List* is the name of the list and *colname* is the name of the column in the list.

Note

In the case of *StringLists*, the values can be obtained by using *List.Item*.

You can use the following attributes to return information about the list using the syntax, *List.attrname*, where *attrname* can be one of the following attributes:

#numRows	Number of rows in this list
#curRow	Current row in this list.
#moreRows	Boolean (true/false) to indicate whether there are more rows following the current row.
#numCols	Number of columns in this list.

@<colname>	Only used on lists that contain resultsets. Contents of the file in the specified column if the colname begins with "url".

For example, `MyList.#numRows` returns the number of rows in the list `MyList`.
`MyList.@urlheader` retrieves the contents of the file specified by the column `urlheader`.

The value of a list is maintained during the processing of a page request. If the list contains a resultset, the resultset is maintained independently of a list that uses it.

errno

The value of `errno` is unchanged.

Example

```
<SETVAR NAME="errno" VALUE="0" />
<STRINGLIST NAME="NameList" STR="Alex Dan Dorothy Elliot Greg
Katie Julie Mark Tom" DELIM="%20" />
<IF COND="IsList.NameList=true" >
  <THEN>
    <BR/>EXAMPLE 1:
    <LOOP FROM="1" UNTIL="NameList.ITEM=Katie"
      LIST="NameList">
        <CSVAR NAME="NameList.ITEM" />
    </LOOP>
    <BR/>EXAMPLE 2:
    <LOOP FROM="3" COUNT="4" LIST="NameList">
      <CSVAR NAME="NameList.ITEM" />
    </LOOP>
    <BR/>EXAMPLE 3:
    <LOOP LIST="NameList">
      <CSVAR NAME="NameList.ITEM" />
    </LOOP>
  </THEN>
  <ELSE>
    Error processing request.
    Unable to create NameList.<BR/>
  </ELSE>
</IF>
```

The browser displays the following:

```
EXAMPLE 1: Alex Dan Dorothy Elliot Greg
EXAMPLE 2: Dorothy Elliot Greg Katie
EXAMPLE 3: Alex Dan Dorothy Elliot Greg Katie Julie Mark Tom
```

See Also

[GOTOROW](#)
[SETROW](#)

mail.send

Sends SMTP email messages to one or more recipients.

Syntax

```
<mail.send
  to="recipient, recipient"
  [from="your e-mail"]
  [subject="subject of message"]
  [body="message body"]
  [replyto="your alternate e-mail address"]
  [contenttype="content format"]
  [charset="character set"] />
```

Parameters

to (required)

Comma-separated list of recipients' e-mail addresses to receive the message

from (optional)

E-mail address of the sender. If a value is not specified, the default value in `mail.ini` is used.

subject (optional)

Subject of the message.

body (optional)

Body of the mail message.

replyto (optional)

Alternate e-mail address to which you wish replies to go. If a value is not specified, replies go to the sender's address.

contenttype (optional)

Type of content you are sending. The default is `plain/text`.

charset (optional)

Character set of your content. The default is the system default.

Description

A new **email** tab has been added the Content Server properties file. These properties provide the defaults for sending and receiving mail.

errno

The possible values of `errno` include:

Value	Description
-202	Cannot Send Mail

Release Introduced

This tag was introduced in CSEE release 5.0.

name.get

Retrieves the distinguished name of the user that is logged in.

Syntax

```
<name.get  
    output="output variable"/>
```

Parameters

output (required)

Variable name to in which to return the distinguished name.

errno

When using the name.get tag, errno must always be set to 0 (zero).

Example

```
<name.get output="curruser"/>
```

Release Introduced

This tag was introduced in CSEE release 4.0.

See Also

[name.makechild](#)

name.makechild

Builds a distinguished name.

Syntax

```
<name.makechild  
    [output="name"]  
    context="parent name">  
    <name.argument  
        name="naming attribute"  
        value="some value"/>  
</name.makechild/>
```

Parameters

output (optional)

Variable name in which the distinguished name is returned.

context (required)

Parent name to be used as a prefix for the newly constructed name.

name.argumentname (required)

Name of the naming attribute, such as cn or uid.

`name.argumentvalue` (required)

Value of the named attribute.

Description

This tag builds a distinguished name from the name of the parent and the least significant component of the child's name. The least significant component is specified using the `dir.argument` parameter. The name and value of the attributes are separated by the `syntax.separator.typeval.ini` property.

errno

The possible values of `errno` include:

Value	Description
-15024	Required parameter missing

Example

```
<name.makechild
output="newgroupname"context="CS.Properties.dir.groupParent"/>
  <dir.argument name="CS.Properties.dir.cn"
value="Variables.newGroupcn"/>
  <dir.argument name="CS.Properties.dir.cn"
value="Variables.parentGroupcn"/>
</name.makechild>
```

Release Introduced

This tag was introduced in CSEE release 4.0.

OBJECT.ARGUMENT

Argument for other object tags.

Syntax

```
<OBJECT.ARGUMENT NAME="some name"
VALUE="some value"/>
```

Parameters

`NAME` (required)

Name of the tag the argument is for.

`VALUE` (required)

Value for this argument.

Description

This tag can only be used as a child tag for other tags in the object tag library.

errno

None.

Example

The following example creates an object table with the additional columns name and description.

```
<OBJECT.INSTALL
CLASSNAME="com.openmarket.framework.objects.AbstractObject"
ARG1 ="sample">
    <OBJECT.ARGUMENT NAME="name" value="VARCHAR(32) NOT NULL"/>
    <OBJECT.ARGUMENT NAME="description" value="VARCHAR(128) NOT
NULL"/>
</OBJECT.INSTALL>
```

See Also

[OBJECT.CREATE](#)
[OBJECT.INSTALL](#)
[Release Introduced](#)

OBJECT.CHECKIN

Checks in an object.

Syntax

```
<OBJECT.CHECKIN NAME="object name"
    [ANNOTATION="some annotation"]
    [CHECKOUT="checkout info"]
    [FLUSH="true or false"]/>
```

Parameters

NAME (required)

Specifies the name of the object in the object pool. A previous load command is required to bring the object in to the object pool.

ANNOTATION (optional)

Specifies an annotation description that should be included with the checkin.

CHECKOUT (optional)

Specifies if the object should remain checked out. The default is `false`, and does not leave the object checked out.

FLUSH (optional)

Specifies that upon success, the object should be flushed from the object pool. A `true` value indicates that object should be flushed. The default is value if `false`.

Description

This tag checks an object in to storage. The object must have been checked out and loaded in to the object pool previously for this tag to function properly. Successful execution of this tag changes object's revision tracking state from `checked out` to `checked in`.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing
-10005	The requested object is not in the object pool (is not loaded into memory).
-10012	Object is not checked out
-10014	Object already checked out
-10016	Unable to check in specified object

Example

The following example checks in an object called `object1`. The example provides an annotation stating that the object's description has been changed. In addition to values that specify the object is no longer checked out, and to flush the object from the object pool.

```
<OBJECT.CHECKIN NAME="object1"
    ANNOTATION="changed description"
    CHECKOUT="false"
    FLUSH="true"/>
```

See Also

[OBJECT.CHECKIN](#)

OBJECT.CHECKOUT

Checks out an object.

Syntax

```
<OBJECT.CHECKOUT CLASS="some class"
    OBJECTTYPE="some object type"
    OBJECTID="some object id"
    [OBJECTVERSION="some object version"]/>
```

Parameters

- CLASS** (required)
Specifies the full name of the implementing class.
- OBJECTTYPE** (required)
Type of object to be checked out.
- OBJECTID** (required)
Content Server ID of the object to be checked out.
- OBJECTVERSION** (optional)
Version of the object to be checked out.

Description

This tag checks out an existing object in the revision tracking system.

errno

The possible values of `errno` include:

Value	Description
-10001	The implementing class is invalid.
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-10006	The object ID is not valid.
-10007	The version of the object is not valid.
-10011	Database table is not revision tracked.
-10013	Object is checked out by another user.
-10014	Object is already checked out
-10015	Unable to check out specified object.
-10019	Database table is not an object table.

Example

The following example checks out an object called `sample` from the revision tracking system.

```
<OBJECT.CHECKOUT  
    CLASS="com.openmarket.framework.objects.AbstractObject"  
    OBJECTTYPE"sample"  
    OBJECTID="Variables.id"/>
```

See Also

[OBJECT.CHECKIN](#)

OBJECT.CHILDREN

Retrieves children of a particular node.

Syntax

```
<OBJECT.CHILDREN NAME="name of node"  
                  LIST="some list"  
                  [OBJECTTYPE="some object type"]  
                  [OBJECID="some object id"]  
                  [OBJECTVERSION="some object version"]  
                  [CODE="selected code"]  
                  [ORDER="name1, name2, name3"]/>
```

Parameters

NAME (required)

Name of the parent node.

LIST (required)

Specifies the name of the list in the list pool. This is the same list pool used by other Content Server tags that generate lists. If a list exists with the same name, it is replaced by the new list.

OBJECTTYPE (optional)

Specifies a requested object type for the child nodes. It can be used in conjunction with the `objectid` and `objectversion` parameters to request a specific object, or it can be used independently to request all objects of a specific class.

OBJECTID (optional)

Specifies a requested object identifier for the child nodes. It should be used in conjunction with the `objecttype` parameter, to specify the class of object.

OBJECTVERSION (optional)

Specifies a requested object version for the child nodes. This parameter is typically used to uniquely identify an object, when used in conjunction with the `objecttype` and `objectid` parameters.

CODE (optional)

Allows for selection of child nodes based on the code associated with the node.

ORDER (optional)

Specifies a comma separated list of field names that should be used to order the list of objects.

Description

This tag builds a list of child node objects that meet specified criteria.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-10006	The object ID is not valid.
-10007	The version of the object is not valid.

Example

The following example retrieves only the child nodes from the `sample` object type from parent node1:

```
<OBJECTCHILDREN NAME="node1"
                LIST="list1"
                OBJECTTYPE="sample"/>
```

See Also

[OBJECT.ROOT](#)
[OBJECT.SETPARENT](#)

OBJECT.CREATE

Creates an object.

Syntax

```
<OBJECT.CREATE CLASSNAME=class name of object
                NAME="name of object"
                [ALLOCATE="true|false"]
                [ARG1="argument 1"]
                [ARG2="argument 2">]
                <[OBJECT.ARGUMENT NAME ="some name" VALUE="some value"]/>
</OBJECT.CREATE>
```

Parameters

CLASSNAME (required)

Classname of the object to be created.

NAME (required)

Name of the object to be created.

ALLOCATE (optional)

When set to true, the tag allocates the object, and makes the object id available before the object is saved to the database.

ARG1 (optional)

Argument 1.

ARG2 (optional)

Argument 2.

ARGUMENT (optional)

Name/value pairs to be set in the object.

Description

This tag creates a new object instance of the implementing class, and saves it in the object pool using the specified name. The implementing class must implement a create method in order for an instance of the implementing class to be constructed. The create method should have one of the following prototypes:

- `public static IAbstractBase Create(ICS ics)`
- `public static IAbstractBase Create(ICS ics, String sArg1)`
- `public static IAbstractBase Create(ICS ics, String sArg1, String sArg2)`

errno

The possible values of errno include:

Value	Description
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing.
-10010	Invalid attribute value

Example

```
<OBJECT.CREATE
  CLASSNAME="com.openmarket.framework.objects.AbstractObject"
  NAME="object1"
  ARG1="sample">
  <OBJECT.ARGUMENT Name="name" Value="John Doe"/>
</OBJECT.CREATE/>
```

See Also

[OBJECT.GATHER](#)

[OBJECT.SAVE](#)

[OBJECT.SET](#)

OBJECT.DEINSTALL

Deinstalls an object by deleting its supporting tables.

Syntax

```
<OBJECT.DEINSTALL CLASSNAME="some classname"  
    [ARG1="argument 1"]  
    [ARG2="argument 2"]/>
```

Parameters

CLASSNAME (required)

Classname of the object to be deinstalled.

ARG1 (optional)

Argument 1.

ARG2 (optional)

Argument 2.

Description

This tag deletes the tables that support the specified class with optional arguments.

errno

The possible values of errno include:

Value	Description
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing.
-10010	Invalid attribute value

Example

The following example deletes the object table called `sample`:

```
<OBJECT.DEINSTALL  
    CLASSNAME="com.openmarket.framework.objects.AbstractObject"  
    ARG1="sample"/>
```

See Also

[OBJECT.INSTALL](#)

OBJECT.DELETE

Deletes an object from storage and the object pool.

Syntax

```
<OBJECT.DELETE NAME="some object name"/>
```

Parameters

NAME (required)

Name of the object in the object pool to be deleted.

Description

This tag deletes a loaded object from storage and removes it from the object pool. The object to be deleted must exist in storage and the object pool.

errno

The possible values of `errno` include:

Value	Description
-105	Database error
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

Example

The following example deletes `object1` from the object pool.

```
<OBJECT.DELETE NAME="object1"/>
```

See Also

[OBJECT.LOAD](#)

OBJECT.GATHER

Gathers fields into an object.

Syntax

```
<OBJECT.GATHER NAME="object name"  
                [PREFIX="some prefix"]  
                [FIELDLIST="list of field names"]/>
```

Parameters

NAME (required)

Name of the object in the object pool.

PREFIX (optional)

Specifies the name of a string to be used as the prefix for variable names, which may exist in the environment. A colon (:) is used as a prefix separator, followed by a field name when the `object:gather` tag is looking for variables in the environment.

FIELDLIST (optional)

Specifies a comma-separated list of field names that should be gathered. Any field in the list that does not exist in the environment is cleared. If this parameter is not specified, then the `object:gather` tag looks at all fields in the environment. If they are not found the existing value is maintained.

Description

This tag retrieves name/value pairs from the environment, and sets them into a loaded object. The name/value pairs are grouped together using a prefix and a prefix separator.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

Example

The following example gathers the name and description fields from the environment into `object1`, using the prefix `sample`:

```
<OBJECT.GATHER NAME="object1"  
                PREFIX="sample"  
                FIELDLIST="name,description"/>
```

See Also

[OBJECT.GET](#)
[OBJECT.SCATTER](#)
[OBJECT.STREAM](#)

OBJECT.GET

Copies a field from an object.

Syntax

```
<OBJECT.GET NAME="some object name"
            FIELD="some field"
            [OUTPUT="some output"]/>
```

Parameters

NAME (required)

Name of the object.

FIELD (required)

Name or type of field to retrieve.

OUTPUT (optional)

Specifies the name of the output variable, which should be created in the environment. If this parameter is not specified, the variable is given the same name as the field.

Description

This tag copies a field value from a loaded object into the environment.

errno

The possible values of errno include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

Example

The following example copies the name column into a variable called name:

```
<OBJECT.GET NAME="object1"
            FIELD="name"
            OUTPUT="name"/>
```

See Also

[OBJECT.GATHER](#)
[OBJECT.SCATTER](#)
[OBJECT.STREAM](#)

OBJECT.INSTALL

Installs an object.

Syntax

```
<OBJECT.INSTALL CLASSNAME="some class name"
    [DIR="some directory"]
    [ACL="some acl"]
    [TRACK="revision tracking true or false"]
    [ARG1="argument 1"]
    [ARG2="argument 2"]>
  <OBJECT.ARGUMENT NAME="name"
    VALUE="value" />
</OBJECT.INSTALL>
```

Parameters

CLASSNAME (required)

Class name of the object.

DIR (optional)

Directory of the object.

ACL (optional)

ACL of the object.

TRACK (optional)

Indicates if the table associated with the implementing class should be revision tracked.

ARG1 (optional)

Argument 1.

ARG2 (optional)

Argument 2.

Description

This tag creates an installation of the implementing class. This typically involves creating the tables to allow for storage of instances of the class. The installation method should use one of the following prototypes:

- `public static int Install (ICS ics, Vector vector, String sUploadDir, String sAcList, boolean bTrack)`
- `public static int Install (ICS ics, Vector vector, String sUploadDir, String sAcList, boolean bTrack, String sArg1)`
- `public static int Install (ICS ics, Vector vector, String sUploadDir, String sAcList, boolean bTrack, String sArg1, String Arg2)`

The argument in the second prototype is a vector of a column elements, represented by the `ColumnElement` class. These typically represent column definitions using the `ColumnDefinition` class.

errno

The possible values of `errno` include:

Value	Description
-10001	Implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing
-10010	Invalid attribute value.

Example

The following example creates an object table with the additional columns `name` and `description`.

```
<OBJECT.INSTALL
CLASSNAME="com.openmarket.framework.objects.AbstractObject"
ARG1 ="sample">
    <OBJECT.ARGUMENT NAME="name" value="VARCHAR(32) NOT NULL"/>
    <OBJECT.ARGUMENT NAME="description" value="VARCHAR (128) NOT
NULL"/>
</OBJECT.INSTALL>
```

See Also

[OBJECT.CREATE](#)
[OBJECT.DEINSTALL](#)

OBJECT.ISEMPTY

Determines whether a field in an object is empty.

Syntax

```
<OBJECT.ISEMPTY
    NAME="name"
    FIELD="field or column name"
    OUTPUT="variable name" />
```

Parameters

NAME (required)

Name used when the object was created using [OBJECT.CREATE](#) or loaded using [OBJECT.LOAD](#).

FIELD (required)

Name of the field or column in the object.

OUTPUT (required)

Name of output variable.

Description

The result of this tag (held in the output variable) is either the string “true” or the string “false.” If the tag results in an error, no output variable is created and `errno` is set.

errno

The possible values of `errno` include:

Value	Description
-10004	Missing required attribute.
-10005	No object by specified name.

Release Introduced

This tag was introduced in CSEE release 5.0.

OBJECT.LIST

Retrieves a list of objects.

Syntax

```
<OBJECT.LIST CLASSNAME="some class name"
    LIST="name of list"
    [ORDER="list of field names"]
    [ARG1="argument 1"]
    [ARG2="argument 2"]>
    <OBJECT.ARGUMENT NAME="some name" VALUE="some value"/>
</OBJECT.LIST>
```

Parameters

CLASSNAME (required)

Classname of the object.

LIST (required)

Specifies the name of the list in the list pool. This is the same list pool used by other Content Server tags that generate lists. If a list exists with the same name, it is replaced by the new list.

ORDER (optional)

Specifies a comma separated list of names that should be used to order the list of objects.

ARG1 (optional)

Argument 1.

ARG2 (optional)

Argument 2.

ARGUMENT (optional)

Name/value pairs to set in the object.

Description

This tag builds a list of objects that meet the specified criteria. The criteria are a list of name/value pairs, which must be present in the object. The implementing class must implement a list method. The list method should have one of the following prototypes:

- `public static IList List (ICS ics, FTValList aArgument, String sOrder)`
- `public static IList List (ICS ics, FTValList aArgument, String sOrder, String sArg1)`

errno

The possible values of errno include:

Value	Description
-101	No rows returned.
-105	Database error.
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing.
-10010	Invalid attribute value.

Example

The following example lists all objects in the sample table.

```
<OBJECT.LIST
```

```
  CLASSNAME="com.openmarket.framework.objects.AbstractObject"
    LIST="list1"
    ARG1="sample"/>
```

OBJECT.LOAD

Loads an object.

Syntax

```
<OBJECT.LOAD CLASSNAME="some classname"  
    NAME="some object name"  
    [OBJECTID="some object id"]  
    [OBJECTVERSION="some object version"]  
    [CHECKOUT="checkout info"]  
    [NODEID="some node id"]  
    [FIELD="some field"]  
    [VALUE="some object value"]  
    [ARG1="argument 1"]  
    [ARG2="argument 2"]  
    [LIST="listname"]/>
```

Parameters

CLASSNAME (required)

Classname of the object.

NAME (required)

Name of the object.

OBJECTID (optional)

ID of the object.

OBJECTVERSION (optional)

Version of the object.

CHECKOUT (optional)

Checkout information for the object.

NODEID (optional)

Node ID for the object.

FIELD (optional)

Specifies the name of a field argument that can be used to uniquely identify an object. When using this parameter the, value parameter should also be specified to indicate the value of the specified field.

VALUE (optional)

Specifies the value of a field for unique identification. The value parameter must be used in conjunction with the field parameter.

ARG1 (optional)

Argument 1.

ARG2 (optional)

Argument 2.

LIST (optional)

Name of existing list. This parameter allows you to load an object from an existing list rather than from the database.

Description

This tag creates a new instance of the implementing class, loads a specific object into the object instance, and saves the object instance in the object pool using the specified name. The object can be requested by referencing the following information:

- object identifier
- node object identifier
- key field and key value

The implementing class must implement a load method. The load method should have one of the following prototypes:

- `public static MyClass Load(ICS ics, Long lObjectId, Integer iObjectVersion, boolean bLock)`
- `public static MyClass Load(ICS ics, Long lObjectId, Integer iObjectVersion, boolean bLock, String sArg1)`
- `public static MyClass Load(ICS ics, Long lNodeId)`
- `public static MyClass Load(ICS ics, Long lNodeId, String sArg1)`
- `public static MyClass Load(ICS ics, String sField, String sValue)`
- `public static MyClass Load(ICS ics, String sField, String sValue, String sArg1)`

errno

The possible values of errno include:

Value	Description
-101	No rows returned.
-105	Database error.
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing.
-10006	The object ID is not valid.
-10007	The version of the object is not valid.
-10008	The node ID of the object is not valid.
-10010	Invalid attribute value.
-10026	No list by specified name exists.

Example

The following example uses the optional LIST parameter.

```
<object.list
  classname="com.openmarket.framework.objects.AbstractObject"
  arg1="obj03"
  list="list1"/>
  <LOOP LIST="list1">
    <object.load

classname="com.openmarket.framework.objects.AbstractObject"
  arg1="obj03"
  list="list1"
  name="object1"/>
    <object.scatter name="object1"
  prefix="object1"/>
    id: <ics.getvar name="object1:id"/><br/>
  </LOOP>
```

OBJECT.ROLLBACK

Rolls back an object to a previous revision.

Syntax

```
<OBJECT.ROLLBACK CLASS="some class name"
  OBJECTTYPE="type of object"
  OBJECTID="id of object"
  [OBJECTVERSION="version of object"]
  REVISION="revision of object"/>
```

Parameters

CLASS (required)

Classname of the object.

OBJECTTYPE (required)

Type of object.

OBJECTID(required)

ID of the object.

OBJECTVERSION (optional)

Version of the object.

REVISION (required)

Specifies the target revision number of the object for the rollback.

Description

This tag rolls back an object to a specified revision. The implementing class must implement a rollback method. The rollback method should have the following prototype:

```
public static int RollBack (ICS ics, String sObjectCatalog, Long
10bjectid Interger iObjectVersion, int iRevision)
```

errno

The possible values of `errno` include:

Value	Description
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing.
-10006	The object ID is not valid.
-10007	The version of the object is not valid.
-10011	Table is not revision tracked
-10012	Object is not checked out
-10017	Unable to rollback specified object
-10019	Table is not an object table.

Example

The following example rolls backs the `sample` object to revision 1.

```
<OBJECT.ROLLBACK
  CLASS="com.openmarket.framework.objects.AbstractObject"
  OBJECTTYPE="sample"
  OBJECTID="Variables.id"/>
  REVISION="1"/>
```

See Also

[OBJECT.TRACK](#)
[OBJECT.SAVE](#)

OBJECT.ROOT

Retrieves a list of root nodes.

Syntax

```
<OBJECT.ROOT CLASS="some class name"
              LIST="some list"
              [OBJECTTYPE="some object type"]
              [OBJECTID="id of object"]
              [OBJECTVERSION="version of object"]
              [CODE="some code"]
              [ORDER="order of names"]
              [ARG1="argument 1"] />
```

Parameters

CLASS (required)

Class name of the object.

LIST (required)

Specifies the name of the list in the list pool.

OBJECTTYPE (optional)

Type of object.

OBJECTID (optional)

Id of the object.

OBJECTVERSION (optional)

Version of the object.

CODE (optional)

Allows selection of nodes based on their associated code.

ORDER (optional)

Specifies a comma separated list of field names that should be used to order the list of objects.

ARG1 (optional)

Argument one.

Description

This tag builds a list of node objects that have no parent, and are therefore considered root nodes.

errno

The possible values of `errno` include:

Value	Description
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.

Value	Description
-10006	The object ID is not valid.
-10007	The version of the object is not valid.

Example

The following example retrieves a list of root nodes from the `com.openmarket.framework.objects.AbstractObject` class.

```
<OBJECT.ROOT
  CLASS="com.openmarket.framework.objects.AbstractObject"
  LIST="list1"
  ARG1="Tree1"/>
```

See Also

[OBJECT.SETPARENT](#)

OBJECT.SAVE

Saves an object.

Syntax

```
<OBJECT.SAVE NAME="object name"
  [FLUSH="true or false"]/>
```

Parameters

NAME (required)

Name of the object to be saved.

FLUSH (optional)

Specifies that upon success, the object should be flushed from the object pool. A `true` value indicates the object should be flushed. The default value is `false`.

Description

Saves a loaded object into storage. The object must have been previously created or loaded into the object pool. Saving an object either creates a new object in storage or updates an existing object in storage.

errno

The possible values of `errno` include:

Value	Description
-105	Database error.
-10004	A required parameter is missing.

Value	Description
-10005	The requested object is not in the object pool (is not loaded into memory).

Example

The following example saves `object1` and removes it from memory:

```
<OBJECT.SAVE NAME="object1"
    FLUSH="true"/>
```

See Also

[OBJECT.DELETE](#)

OBJECT.SCATTER

Scatters a field from an object.

Syntax

```
<OBJECT.SCATTER NAME="some object name"
    [PREFIX="some prefix"]/>
```

Parameters

NAME (required)

Name of the object.

PREFIX (optional)

Specifies the the name of a string for the variable names that is created in the environment. The prefixes are separated from the field names with a colon (:) when creating variables in the environment.

Description

This command allows fields of a previously loaded object to be copied to the environment as variables with a specified prefix. This tag is the preferred alternative to using multiple `OBJECT.GET` tags.

errno

The possible values of `errno` include:

Value	Description
-105	Database error.
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

Example

The following example creates a variable in the environment with the prefix `sample`:

```
<OBJECT.SCATTER NAME="object1"
    PREFIX="sample"/>
```

See Also

[OBJECT.GATHER](#)
[OBJECT.GET](#)

OBJECT.SET

Sets a field in an object.

Syntax

```
<OBJECT.SET NAME="some object name"
    FIELD="some field"
    VALUE="some value"/>
```

Parameters

NAME (required)
 Name of the object.

FIELD (required)
 Field to be set.

VALUE (required)
 Value to be set in field.

Description

This tag sets the value of a field in a previously loaded object.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

Example

The following example sets the value to `red` in the `name` field of `object1`.

```
<OBJECT.SET NAME="object1"
    FIELD="name"
    VALUE="red"/>
```

See Also

[OBJECT.GET](#)

OBJECT.SETOBJECT

Sets an object in a node.

Syntax

```
<OBJECT.SETOBJECT NAME="some node name"
                  OBJECT="some object"/>
```

Parameters

NAME (required)

Name of the node in which the object is placed.

OBJECT (required)

Name of the object.

Description

This tag sets the referenced object of a node object. Node objects are used with tree tables in the Content Server system. The node object must implement the IAbstractNode interface and the referenced object must implement the IAbstractObject interface.

errno

The possible values of errno include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

Example

The following example maps node1 (residing in a tree) to object1 from an object table.

```
<OBJECT.SETOBJECT NAME="node1"
                  OBJECT="object1"/>
```

See Also

[OBJECT.ROOT](#)

[OBJECT.SETPARENT](#)

OBJECT.SETPARENT

Sets the parent of a node.

Syntax

```
<OBJECT.SETPARENT NAME="name of node"  
                  PARENT="name of parent node"/>
```

Parameters

NAME (required)

Name of the child node.

PARENT (required)

Name of the parent node.

Description

This tag sets the parent node of a node object. Node objects are used with tree tables in the Content Server system.

errno

The possible values of errno include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

Example

The following example sets parent1 as the parent node for the node object node1.

```
<OBJECT.SETPARENT NAME="node1"  
                  PARENT="parent1"/>
```

See Also

[OBJECT.ROOT](#)

OBJECT.STREAM

Streams the value of a field from an object.

Syntax

```
<OBJECT.STREAM NAME="some object name"
                FIELD="some field"
                [RESOLVE="true or false"]/>
```

Parameters

NAME (required)

Name of the object.

FIELD (required)

Field to be used for streaming.

RESOLVE (optional)

Indicates if variable references in the field should be resolved. The default value is false.

Description

This tag streams the field value of a loaded object.

errno

The possible values of errno include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

Example

The following example streams the name field from object1.

```
<OBJECT.STREAM NAME="object1"
                FIELD="name"
                RESOLVE="false"/>
```

See Also

[OBJECT.GET](#)

[OBJECT.SCATTER](#)

OBJECT.TRACK

Tracks an object.

Syntax

```
<OBJECT.TRACK CLASS="some class name"
               [DIR="some directory"]
               REVISIONS="revision information"
               [ARG1="argument 1"]/>
```

Parameters

CLASS (required)

Classname of the object.

DIR (optional)

Directory of the object.

REVISIONS (required)

Revision information for the object to be tracked.

ARG1 (optional)

Argument 1.

Description

This tag activates revision tracking for a previously untracked table. The track method should have one of the following prototypes:

- `public static int Track(ICS ics, string sUploadDir, int iRevisions)`
- `public static int Track(ICS ics, string sUploadDir, int iRevisions, String sArgs1)`

errno

The possible values of errno include:

Value	Description
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing.

Example

The following example activates tracking for the sample table:

```
<OBJECT.TRACK
  CLASS="com.openmarket.framework.objects.AbstractObject"
  DIR="C:\Content\sample"
  REVISIONS="10"
  ARG1="sample"/>
```

See Also

[UNLOCKRECORD](#)

OBJECT.UNDOCHECKOUT

Retracts the checkout of an object.

Syntax

```
<OBJECT.UNDOCHECKOUT NAME="some object name"
  [REVERT="true or false"]
  [FLUSH="true or false"]/>
```

Parameters

NAME (required)

Name of the object.

REVERT (optional)

Specifies that the object should revert back to its last committed state. The default value `true` causes the object to revert back to its last committed state. Specifying the value `false` removes the checkout lock without reverting the object back to the last committed state.

FLUSH (optional)

Flushes the object from the object pool. A `true` value indicates the object should be flushed. The default value is `false`.

Description

This tag is used for retracting a checkout made on an object in the Content Server system. The object can be reverted to its last committed state, or have the checkout lock removed.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-10011	Database table is not revision tracked.

Value	Description
-10012	Object is not checked out
-10018	Unable to undo check out specified object

Example

The following example retracts the checkout for `object1`, reverts it to its last committed state, and flushes it from memory.

```
<OBJECT.UNDOCHECKOUT NAME="object1"
                      REVERT="true"
                      FLUSH="true"/>
```

See Also

[OBJECT.CHECKIN](#)
[OBJECT.CHECKOUT](#)

OBJECT.UNTRACK

Removes tracking on an object.

Syntax

```
<OBJECT.UNTRACK CLASS="some class name"
                [ARG1="argument 1"]/>
```

Parameters

CLASS (required)
 Class name of the object.

ARG1 (optional)
 Argument 1.

Description

This tag deactivates revision tracking on for a previously tracked table. The untrack method should have one of the following prototypes:

- `public static int Untrack (ICS ics)`
- `public static int Untrack (ICS ics, String sArgs1)`

errno

The possible values of `errno` include:

Value	Description
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.

Value	Description
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing.
-10011	Database table is not revision tracked
-10022	Unable to untrack specified object
-10023	Invalid database table name
-10024	User not allowed access for request

Example

The following example deactivates revision tracking on the sample table.

```
<OBJECT.UNTRACK
  CLASS="com.openmarket.framework.objects.AbstractObject"
  ARG1="sample"/>
```

See Also

[OBJECT.TRACK](#)

portal.logdep

Logs an article as a dependency on an item.

Syntax

```
<portal.logdep
  id="string"
  date="JDBC-formatted date" />
```

Parameters

id (required)

A string that uniquely identifies the item you want to record. For example, Article-9876835 where 9876835 is a unique ID for an asset of type Article.

date (required)

SQL date format for the date and time when the item was last modified. If this parameter is left null, the current date and time are used.

Description

The name of the output variable created is `randval`.

errno

`errno` is not set.

Example

This example is taken from the New Portal site example included with Content Server.

```
<!-- get some articles from the database -->
<SELECTTO LIST="articlelist" LIMIT="8" WHERE="category"
FROM="NewPortalArticle"/>

<!-- display the list of articles -->
<IF COND="IsList.articlelist=true">
<THEN>
  <LOOP LIST="articlelist">
    <portal.logdep id="articlelist.id"/>
    <satellite.link pagename="OpenMarket/Samples/NewPortal/XML/
main"
      mode="article"
      articleid="articlelist.id" />
    <a REPLACEALL="Variables.referURL"
href="Variables.referURL">
      <div class="articleLinkBody"><CSVAR
NAME="articlelist.headline"/></div>
      </a>
      <div class="articleCopyright"><CSVAR
NAME="articlelist.byline,
articlelist.creditline"/></div>
    </LOOP>
  </THEN>
<ELSE>
  Error retrieving article headlines.  errno=<CSVAR
NAME="Variables.errno"/> </ELSE> </IF>
```

Release Introduced

This tag was introduced in CSEE release 5.0.

PORTAL.RANDOMVAL

Generates a random number between 1 and 10 to select an ad image for the banner.

Syntax

```
<PORTAL . RANDOMVAL
  MINVAL="integer"
  MAXVAL="integer"
  OUTPUT="variable name" />
```

Parameters

MINVAL (required)

Minimum integer value of the random number.

MAXVAL (required)

Maximum integer value of the random number.

OUTPUT (required)

Name of the variable in which to store the number generated.

errno

The possible values of errno include:

Value	Description
-2	MINVAL is greater than MAXVAL.

Release Introduced

This tag was introduced in CSEE release 5.0.

POST

Submits an HTTP request to a specific server.

Syntax

```
<POST
  URL="url string"
  [SECURED="secured value"]
  [USERNAME="user name"]
  [PASSWORD="password"]
  [COOKIE="cookie name"]
  [LOGOUT="logout value"]
  [HOST="hostname"]
  [PORT="port address"]
  [PROXY="proxyname"]
  [PROXYPORT="proxy port"]
  [OUTDATA="output variable"]
  [OUTERROR="error variable"]
  [OUTCOOKIE="outcookie value"]>
  <ARGUMENT NAME="somearg1" VALUE="somevalue1"/>
  <ARGUMENT NAME="somearg2" VALUE="somevalue2"/>
</POST>
```

Parameters

URL (required)

URL requests without protocol or host (for example, docs/index.html).

SECURED (optional)

Use HTTPS. Values are true and false. Default is false.

USERNAME (optional)

Username to use to log in to the host (default is current user).

PASSWORD (optional)

User password to use to log in to the host (default is current user).

COOKIE (optional)

Session cookie to use (defaults to cookie set from previous post).

LOGOUT (optional)

Log out the user session created on the target server. Values are `true` and `false`, and the default is `false`.

HOST (optional)

Target host name (default is current host).

PORT (optional)

Target host port address (default is the current host port).

PROXY (optional)

Proxy server host name (default is none).

PROXYPORT (optional)

Proxy server port address (default is none).

OUTDATA (optional)

Variable name whose value contains output of post (default is no variable).

OUTERROR (optional)

Variable name whose value contains errors reported by post (default is no variable).

OUTCOOKIE (optional)

Variable name whose value contains the security cookie set by the post (default is no variable).

Description

The POST tag submits an HTTP request to a specific server. This tag can use proxy information and can cause a login if connection is to a Content Server engine. This tag is useful for submitting a mirror request to a remote server, or inserting a page to get data from a disk cache, and the like.

errno

The value of `errno` is unchanged.

RELEASE

Releases a lock on a row from a tracked table.

Syntax

```
<RELEASE TABLE="TABLE_NAME"
      ASSET="PRIMARY_KEY" />
```

Parameters**TABLE (required)**

Name of the tracked table.

ASSET (required)

Value of the primary key for the row to release.

Description

The RELEASE tag releases a lock on a row from a tracked table. Only the user who has the row locked against the table can release it. This also reverts the row in the table to its last committed state, overwriting any changes the user made while the row was locked.

errno

The possible values of `errno` include:

Value	Description
-1000	Command failure. Check the <code>errdetail</code> variable for the specific error.

Example

This example assumes all `RELEASE` parameters are exposed on an HTML form. It first checks whether the parameters exist, clears any values from `errno`, and then releases the row.

```

<!-- Make sure required parameters exist -->
<ICS.ARGUMENT COND="IsVariable.tablename=false">
  <THEN>
    <SETVAR NAME="tablename" VALUE=""/>
  </THEN>
</IF>
<ICS.ARGUMENT COND="IsVariable.asset=false">
  <THEN>
    <SETVAR NAME="asset" VALUE=""/>
  </THEN>
</IF>
<!-- clear out errno -->
<SETVAR NAME="errno" VALUE="0"/>
<!-- do the release -->
<RELEASE TABLE="Variables.tablename"
ASSET="Variables.asset"/>
<!-- see if the release succeeded -->
<ICS.ARGUMENT COND="Variables.errno=0">
  <THEN>
    <!-- if succeeded, note it-->
    Release succeeded for
    <CSVAR NAME="Variables.asset"/>
    <BR/>
  </THEN>
  <ELSE>
    <!-- otherwise, call an error reporting element -->
    Release failed.<BR/>
  </ELSE>
</IF>

```

See Also

[UNLOCKRECORD](#)

REMOVECOOKIE

Deletes a cookie from the client browser.

Syntax

```
<REMOVECOOKIE NAME="COOKIE_NAME" />  
  SECURE="SECURE_VAL "  
  [URL="URL_PATH"]  
  [DOMAIN="DOMAIN_NAME"] />
```

Parameters

NAME (required)

Name of the cookie to delete.

SECURE (required)

Indicates whether security is set on this cookie. Possible values are true or false.

URL (optional)

Valid URL. This value restricts the deleting of the cookie from the client to this URL only.

DOMAIN (optional)

Valid domain name. This value restricts the deleting of the cookie from the client to addresses in this domain only.

Note

For REMOVECOOKIE to work correctly, you must supply the same values for these parameters as you did for the SETCOOKIE call that originally created the cookie.

Description

The REMOVECOOKIE tag deletes a cookie from the client browser. REMOVECOOKIE must appear before any tag that streams back data on the page or the cookie is not deleted. This is an HTTP requirement.

errno

The value of errno is unchanged.

Example

This example deletes the cookie mycookie from the client browser.

```
<REMOVECOOKIE NAME="mycookie" SECURE = "false" />
```

See Also

[SETCOOKIE](#)

REMOVECOUNTER

Deletes a counter.

Syntax

```
<REMOVECOUNTER NAME="COUNTER_NAME" />
```

Parameters

NAME (required)

Name of the counter to remove.

Description

The REMOVECOUNTER tag deletes a counter. After a counter is removed it is undefined and its value cannot be referenced.

errno

The value of errno is unchanged.

Example

This example first sets a counter, uses it, then removes it.

```
<SETCOUNTER NAME="fred" VALUE="100" />
<INCCOUNTER NAME="fred" VALUE="200" />
  Fred is <CSVAR NAME="Counters.fred" />
<REMOVECOUNTER NAME="fred" />
```

See Also

[SETCOUNTER](#)
[INCCOUNTER](#)

REMOVESSVAR

Deletes a session variable.

Syntax

```
<REMOVESSVAR NAME="SSVAR_NAME" />
```

Parameters

NAME (required)

Name of the session variable to remove.

Description

The REMOVESSVAR tag deletes a session variable. After a session variable is removed, it is no longer defined for the session and its value cannot be referenced.

errno

The value of `errno` is unchanged.

Example

This example removes the session variable `userpref`:

```
<REMOVESSVAR NAME="userpref"/>
```

See Also

[SETSSVAR](#)

REMOVEVAR

Deletes a variable.

Syntax

```
<REMOVEVAR NAME="VAR_NAME"/>
```

Parameters

NAME (required)

Name of the variable to remove.

Description

The `REMOVEVAR` tag deletes a variable. After a variable is removed it is no longer defined for the processing of the page and its value cannot be referenced. For more information about variables, see the *CSEE Developer's Guide*.

errno

The value of `errno` is unchanged.

Example

This example removes the variable `"myvar"`:

```
<REMOVEVAR NAME="myvar"/>
```

See Also

[SETVAR](#)

RENAMELIST

Removes a list object from the hash list and replaces it under the new name.

Syntax

```
<RENAMELIST LIST="NewList"
          FROM="SomeList"/>
```

Parameters

LIST (required)

The new list name.

FROM (required)

The current list name. (The name you want to change.)

Description

The RENAMELIST tag removes a list object from the hash list and replaces it under the new name. Does not reset any internal list state (for example, the row number).

The column values of the current row of a list can be obtained using the syntax, *List.colname* where *List* is the name of the list and *colname* is the name of the column in the list.

Note

In the case of *StringLists*, the values can be obtained by using *List.Item*.

You can use the following attributes to return information about the list using the syntax, *List.attname*, where *attname* can be one of the following attributes:

#numRows	Number of rows in this list
#curRow	Current row in this list.
#moreRows	Boolean (true/false) to indicate whether there are more rows following the current row.
#numCols	Number of columns in this list.
@<colname>	Only used on lists that contain resultsets. Contents of the file in the specified column if the colname begins with "url".

For example, *MyList.#numRows* returns the number of rows in the list *MyList*. *MyList.@urlheader* retrieves the contents of the file specified by the column *urlheader*.

The value of a list is maintained during the processing of a page request. If the list contains a resultset, the resultset is maintained independently of a list that uses it.

errno

The possible values of `errno` include:

Value	Description
-504	List rename error.

Example

The following example generates a list named `PortalArticles` and then changes the list's name from `PortalArticles` to `Articles`:

```
<SELECTTO FROM="NewPortalArticle" WHAT="*" LIST="PortalArticles" />
<SETVAR NAME="errno" VALUE="0" />
<RENAMELIST LIST="Articles" FROM="PortalArticles" />
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>
```

REPLACE

Substitutes the value of a variable, session variable, list item, built-in value, or counter within `REPLACE`.

Syntax

```
<REPLACE STR="STRING_VAL">
</REPLACE>
```

Parameters

STR (required)

Value of variable, session variable, list item, built-in value or counter to substitute.

Description

The `REPLACE` tag substitutes the value of a variable, session variable, list item, built-in value, or counter within the tag. Use the `REPLACE` tag selectively or you may slow performance, since `REPLACE` must scan all text in all tags contained within the parent `REPLACE`. For multiple replacements in non-tag data (for example, content or `CDATA`), use `REPLACEALL` for better performance.

For HTML tags, you can use the `REPLACEALL` attribute to replace values. For more information about variables, see the *CSEE Developer's Guide*.

errno

The value of `errno` is unchanged.

Example

This example substitutes the value of the variable `file` for `Variables.file` within `REPLACE`.

```
<SETVAR NAME="file" VALUE="somefile.html"/>
  <REPLACE STR="Variables.file">
    <a href="http://someserver/somefolder/Variables.file">
      See this story.</a>
    </REPLACE>
```

See Also

[REPLACEALL](#)

REPLACEALL

Substitutes the value of one or more variables, session variables, list items, built-in values or counters within `REPLACEALL`.

Syntax

```
<REPLACEALL
  LIST="LIST_VAL"/>
```

Parameters

LIST (required)

Comma-separated list of variables, session variables, built-in values, list items or counters to replace.

Description

The `REPLACEALL` tag substitutes the value of one or more variables, session variables, list items, built-in values or counters within `REPLACEALL`. Use the fewest number of tags and content possible within `REPLACEALL` for optimal performance.

For HTML tags, you can use the `REPLACEALL` attribute to replace values.

For more information about variables, see the *CSEE Developer's Guide*.

errno

The value of `errno` is unchanged.

Example

The following example replaces the variable `file` and the built-in value `CS.Date`.

```
<REPLACEALL LIST="Variables.file,CS.Date">
  <a href="http://someserver/somefolder/Variables.file">
    See this headline at CS.Date
  </a>
</REPLACEALL>
```

See Also

[REPLACE](#)

RESTOREPROPERTY

Sets the property file to the Content Server property file (`futuretense.ini`).

Syntax

```
<RESTOREPROPERTY
  CLOSE="CLOSE_VAL"/>
```

Parameters

CLOSE (required)

Whether to close the connection to a database opened using the current properties. Possible values are `true` or `false`.

If `CLOSE` is set to `true` any open database connection specified by the current properties is closed before restoring the defaults. This invalidates any resultsets cached against that database. If access to those resultsets is required after `RESTOREPROPERTY`, `CLOSE` should be set to `false`. This keeps the connection open and resultsets against the database available.

Note that if the Content Server property file `futuretense.ini` is in effect, using `RESTOREPROPERTY` with `CLOSE = "true"` closes the database connection and deletes all existing variables. One might be tempted to do this if a `LOADPROPERTY` fails, but it should not be necessary.

errno

The value of `errno` is unchanged.

Example

This example loads the properties from `other.ini` and queries a table in the database specified in `other.ini`. After using the resultset from the query, the original property file is restored and the connection to the database specified in `other.ini` is closed. The value of `"numbers"` is undefined after `RESTOREPROPERTY`.

```
<SETVAR NAME="errno" VALUE="0"/>
<LOADPROPERTY FILE="other.ini"/>
<SELECTTO LIST="numbers" WHAT="*"
  FROM="NumberData"/>
  after select errno is:
  <CSVAR NAME="Variables.errno"/>
<RESTOREPROPERTY CLOSE="true"/>
```

See Also

[LOADPROPERTY](#)

RETRIEVEREVISION

Retrieves a specific version of a tracked record.

Syntax

```
<RETRIEVEREVISION TABLE="TABLE_NAME"
                  ASSET="PRIMARY_KEY"
                  VERSION="VERSION_NUM"
                  LIST="LIST_VAL"/>
```

Parameters

TABLE (required)

Name of the table containing row to retrieve the version from.

ASSET (required)

The value of the primary key for the record.

VERSION (required)

The version number.

LIST (required)

The name of the Query object to create.

errno

The possible values of errno include:

Value	Description
-1000	Command failure. Check the <code>errdetail</code> variable for the specific error.

Example

```
<SETVAR NAME="errno" VALUE="0"/>
<RETRIEVEREVISION TABLE="movies"
                  ASSET="id"
                  VERSION="3"
                  LIST="rev5"/>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>
```

ROLLBACK

Reverts a row in a tracked table to a previous revision.

Syntax

```
<ROLLBACK TABLE="TABLE_NAME"
          ASSET="PRIMARY_KEY"
          [VERSION="VERSION_NUM"]/>
```

Parameters

TABLE (required)

Name of the table containing the row you want to revert to a previous version.

ASSET (required)

Value of the primary key for the row you want to revert to a previous version.

VERSION (optional)

Version to make current. Values for `VERSION_NUM` are:

- "first"- Roll back to earliest revision of record.
- "last"- Roll back to most recent revision of record.
- "#"- Roll back to a specific revision number.
- "date"- Roll back the record to whatever revision was current at the specified date. The date must be in SQL format (yyyy-mm-dd hh:mm:ss) and is expected to be GMT.

Description

The `ROLLBACK` tag reverts a row in a tracked table to a previous revision. The current user must have the row locked and have `rtaudit` privileges against the table to roll back the row. A row can be rolled back to its earliest revision, its most recent revision, a specific version number, or whatever revision was current at a specified date.

errno

The possible values of `errno` include:

Value	Description
-1000	Command failure. Check the <code>errdetail</code> variable for the specific error.

Example

This example assumes all `ROLLBACK` parameters are exposed on an HTML form. It first checks whether the parameters exist, clears any values from `errno`, and then performs a roll back on the row.

```
<!-- Make sure required parameters exist -->
<IF COND="IsVariable.tablename=false">
  <THEN>
    <SETVAR NAME="tablename" VALUE=""/>
  </THEN>
</IF>
```

```

<IF COND="IsVariable.asset=false">
  <THEN>
    <SETVAR NAME="asset" VALUE=""/>
  </THEN>
</IF>
<IF COND="IsVariable.version=false">
  <THEN>
    <SETVAR NAME="version" VALUE=""/>
  </THEN>
</IF>
<!-- clear out errno -->
<SETVAR NAME="errno" VALUE="0"/>
<!-- do the rollback -->
<ROLLBACK TABLE="Variables.tablename"
          ASSET="Variables.asset"
          VERSION="Variables.version"/>
<!-- see if the rollback succeeded -->
<IF COND="Variables.errno=0">
  <THEN>
    <!-- if succeeded, note it-->
    Rollback succeeded for
    <CSVAR NAME="Variables.asset"/>
    <BR/>
  </THEN>
<ELSE>
    <!-- otherwise, call an error reporting element -->
    Rollback failed.<BR/>
  </ELSE>
</IF>

```

SEARCH

Searches an index based on a query.

Syntax

```

<SEARCH [INDEX="INDEX_NAME"]
  WHAT="SEARCH_STRING"
  [LIST="OUTPUT_LIST"]
  QUERYPARSER="PARSER_NAME"
  RELEVANCE="RELEVANCE"
  [LIMIT="MAX_NO_RESULTS"]
  [CHARACTERSET="CHAR_SET"]
  [SEARCHENGINE="NAME"]/>

```

Parameters

INDEX (optional)

Name of the index to search. Default is specified in the Content Server properties file.

WHAT (required)

Query to submit. This should be in the format the given search engine understands and may also depend on the QUERYPARSER value.

LIST (required)

Name of the output list. Columns returned are:

- **ENTRY** - Name of the index entry that matches the search criteria.
- **DETAIL** - Details of the index entry that matches the search criteria.
- **DATE** - Date the entry was added or the date specified when the index was added. Format is in Java SQL.
- **RELEVANCE** - Relevance value associated with the search result. The closer the value is to 1, the more useful and relevant the search result is likely to be.

QUERYPARSER (optional for Verity)

The AltaVista search engine supports three search types—Simple, Advanced, and Combined. Advanced is the default, and was previously the only one supported.

The simple search syntax uses optional + or - signs on a word or phrase to indicate that it is required or prohibited in the search results. The Advanced syntax uses Boolean operators (such as and, or, not, near) to constrict complex queries. In both cases, the query is passed in the **WHAT** parameter. The **RELEVANCE** parameter is ignored.

The combined syntax uses both types of syntax. A Boolean query is specified by the **WHAT** parameter and a simple query is specified by the **RELEVANCE** parameter to determine the ranking order of the results.

The Verity search engine supports three query parsers—Simple, FreeText, and BoolPlus. The optional **QUERYPARSER** tag tells the Verity search engine the syntax of the **WHAT** clause. If this argument is omitted, then the Simple parser is used by default. The parser may be specified by the `verity.parser` property in the `futuretense.ini` file. The choice of parser determines the syntax and form of the query

RELEVANCE (optional for AltaVista)

Boolean term to be used for computing relevance ranking.

LIMIT (optional)

Maximum number of results to return to the list.

CHARACTERSET (optional)

Constant value representing the character set the index uses.

For the AltaVista search engine this value can be 0, 1, or 2 (ISO_LATIN1, ASCII8, UTF8). If you do not specify **CHARACTERSET**, Content Server uses the value of `av.charset` from the Content Server properties file.

Because Java Strings are transformed to UTF8, as they pass through the JNI interface to AltaVista, the 0 value only works for 7-bit ASCII. For all other cases, use the value for UTF8.

For the Verity search engine, this value specifies the name of the subdirectory of the common directory where the locale is defined. If you do not specify **CHARACTERSET**, Content Server uses the value of `verity.charset` in the Content Server properties file.

SEARCHENGINE (optional)

Name of the search engine to use. If **SEARCHENGINE** is not specified, Content Server uses the value of `cs.searchengine` from the Content Server properties file.

Description

The **SEARCH** tag searches an index based on a query. The resultset of the query is stored in a list.

errno

The possible values of `errno` include:

Value	Description
-101	No search results.
-800	Bad search type.
-801	Cannot load search engine.
-802	Unsupported search function.
-805	No default index specified.
-806	Unknown search engine.
-809	Search failed.
-810	Bad character set.
-811	Could not call native method.
-812	Index does not exist.

Example

This example searches the existing index in `c:\ArticleIndex` for headlines with the name `Clinton` in them. If any are found, then the search result list is looped over and the headline and the author are displayed. Since this information is not directly available from the index, the `ENTRY` field of the search result list is used to lookup the article entry in the appropriate database table and the required fields are obtained from the database record for the article.

```
<SETVAR NAME="errno" VALUE="0" />
<SEARCH INDEX="C:/ArticleIdx"
  LIST="headlist"
  WHAT="headline:Clinton"
  SEARCHENGINE="AV" />
<ICS.ARGUMENT COND="Variables.errno=0">
  <THEN>
    <RENAMELIST LIST="headlist">
      <SETVAR NAME="id" VALUE="headlist.ENTRY" />
      <SELECTTO FROM="Variables.tablename"
        WHAT="*"
        WHERE="Variables.id"
        LIST="thearticle" />
      <CSVAR NAME="thearticle.headline" /><BR/>
      <CSVAR NAME="--by thearticle.byline"/>
      <BR/>
    </LOOP>
  </THEN>
</IF>
```


Note

This example requires that the database be accessed for each article. This can be inefficient, especially in those cases where the search results are to be displayed only so that the one element required may be selected for further use. In this case, you should use the `DETAIL` field of the search result list. When entries are added to the index, you should set the `DETAIL` parameter to some useful text string. For this example, it could be set to the article title so that you can then display and use it to select the article of interest.

See Also

[INDEXCREATE](#)

SEARCHDATENATIVE

Converts the Java SQL formatted date to the date format native to the search engine.

Syntax

```
<SEARCHDATENATIVE DATE="JAVA_DATE"
                  NAME="VAR_NAME"
                  [CHARACTERSET="CHAR_SET"]
                  [SEARCHENGINE="NAME"] />
```

Parameters

DATE (required)

The Java SQL date to convert.

NAME (required)

The name of the variable to store the converted date string.

CHARACTERSET (optional)

Constant value representing the character set the index uses.

For the AltaVista search engine this value may be 0, 1, or 2 (ISO_LATIN1, ASCII8, UTF8). If you do not specify `CHARACTERSET`, Content Server uses the value of `av.charset` from the Content Server properties file.

Because Java Strings are transformed to UTF8, as they pass through the JNI interface to AltaVista, the 0 value only works for 7-bit ASCII. For all other cases, use the value for UTF8.

For the Verity search engine, this value specifies the name of the subdirectory of the common directory where the locale is defined. If you do not specify `CHARACTERSET`, Content Server uses the value of `verity.charset` in the Content Server properties file.

SEARCHENGINE (optional)

Name of the search engine to use. If `SEARCHENGINE` is not specified, Content Server uses the value of `cs.searchengine` from the Content Server properties file.

Description

The SEARCHDATENATIVE tag converts the Java SQL formatted date to the date format native to the search engine. Use in the WHAT parameter of a SEARCH tag. Dates in queries must be in a format that the search engine can parse.

It is important that the input to this tag (the DATE parameter) be a valid Java SQL date. Content Server does not detect certain errors in the date format.

The Verity search engine handles most date formats, including the Java SQL format. If you are using Verity, you do not need to call SEARCHDATENATIVE.

errno

The possible values of errno include:

Value	Description
-801	Cannot load search engine.

Example

This example converts the date in Java SQL format to the AltaVista date format and sets the results in the variable AVDate.

```
<SETVAR NAME="errno" VALUE="0"/>
<SEARCHDATENATIVE DATE="1998-05-22"
                  NAME="AVDate"
                  SEARCHENGINE="AV"/>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>
```

SELECTTO

Executes a simple query from a table.

Syntax

```
<SELECTTO FROM="CATALOG_NAME"
  [WHAT="WHAT_CRITERIA"]
  LIST="OUTPUT_LIST"
  [WHERE="WHERE_CRITERIA"]
  [ORDERBY="ORDERBY_EXPRESSION"]
  [LIMIT="MAX_NO_RESULTS"]/>
```

Parameters

FROM (required)

Name of the Content Server table to query. The table must be registered in the SystemInfo table. This table verifies security and caches result sets when the cc.cacheResults property is enabled.

WHAT (optional)

List of columns to return. The format of *WHAT_CRITERIA* is a comma-separated list of column names.

LIST (required)

Name of the list to contain the resultset.

WHERE (optional)

Comma-separated list of table columns to select against. For each item in the *WHERE* string, the value of the corresponding variable is used to construct a SQL *WHERE* clause. All the elements in the comma-separated list *must* be variables. When you specify a datetime field in the *WHERE_CRITERIA*, Content Server uses the *cc.datepicture* property to convert the supplied datetime to the proper format.

ORDERBY (optional)

Orders the rows returned by the query. The format of *ORDERBY_EXPRESSION* is a comma-separated list of column names. You can also override the default sort order by specifying ascending/descending with a (potentially) database specific command like:

```
ORDERBY="colname1 desc,colname2 asc"
```

LIMIT (optional)

Maximum number of rows to return.

Note that if the *LIMIT* parameter is omitted or is less than or equal to 0, then no limit exists and the entire recordset is returned.

Description

The *SELECTTO* tag executes a simple query from a table. The results are stored in a list.

errno

The possible values of *errno* include:

Value	Description
-101	No search results.
-106	One of the elements in the <i>WHERE</i> clause was not a variable. (All elements in the <i>WHERE</i> clause must be variables.)

Example

The following example selects all columns from the *SystemInfo* table and puts the results into a list called *RS* for later use.

```
<SELECTTO FROM="SystemInfo" WHAT="*" LIST="RS" />
```

The next example demonstrates a comma-separated list. It selects records in the *visitor* table where *firstname* = Fred and *lastname* = Smith. It then loops through the list if the list is created and has records.

```
<SETVAR NAME="firstname" VALUE="Fred"/>
<SETVAR NAME="lastname" VALUE="Smith"/>
<SETVAR NAME="tablename" VALUE="visitor"/>
<SELECTTO FROM="Variables.tablename" WHAT="*"
  WHERE="firstname, lastname" LIST="VisitorLst" />
<SETVAR NAME="CurErr" VALUE="Variables.errno"/>
<IF COND="IsList.Visitor=true">
```

```
<THEN>
  <IF COND="Variables.CurErr!=-101">
    <THEN>
      <LOOP LIST="VisitorLst">
        <CSVAR NAME="FOUND MATCH:
VisitorLst.firstname,VisitorLst.lastname" />
      </LOOP>
    </THEN>
    <ELSE>
      <CSVAR NAME="NO RECORD FOUND
matching,
Variables.firstname
Variables.lastname" />
    </ELSE>
  </IF>
</THEN>
<ELSE>
  Error creating list<BR/>
  <!-- put error handling code here -->
</ELSE>
</IF>
```

See Also

[CALLSQL](#)
[EXECSQL](#)
[SQLEXP](#)

SENDMAIL (Deprecated)

As of Content Server version 5.0, replaced by [mail.send](#).

The SENDMAIL tag sends SMTP email messages to one or more recipients.

There are two variants of the SENDMAIL tag:

- SENDMAIL (Variant 1)
- SENDMAIL (Variant 2)

SENDMAIL (Variant 1)

Sends an SMTP mail message to one or more recipients. The Content Server properties file (`futuretense.ini`) must contain valid mail server configuration information to use this tag.

Syntax

```
<SENDMAIL TO="USERS"
          SUBJECT="SUBJECT"
          BODY="MESSAGE"/>
```

Parameters

TO (required)

A comma-separated list of the recipients to receive the message

SUBJECT (required)

Subject of the message.

BODY (required)

Body of the mail message.

Description

The SendMail method sends an SMTP message to one or more recipients. To send mail, the Content Server properties file must contain valid values for the `cs.emailhost` and `cs.emailreturnto` properties. To receive mail, the Content Server properties file must contain valid values for the `cs.emailhost`, `cs.emailaccount`, and `cs.emailpassword` properties.

errno

The possible values of `errno` include:

Value	Description
-202	Cannot send mail.

Example

This example sends an e-mail message to `joeuser` and `janeuser`. The body of the message contains the current value of variable `FOO`. The syntax of all parameter values must be SMTP mail compliant.

```
<SETVAR NAME="FOO" VALUE="Congratulations on becoming a
member."/>
<SENDMAIL TO="joeuser,janeuser"
          SUBJECT="Test Message"
          BODY="Variables.FOO"/>
```

See Also

[EMAIL EVENT](#)

SENDMAIL (Variant 2)

Sends an SMTP mail message to one or more recipients.

Syntax

```
<SENDMAIL TO="USERS"  
    [REPLYTO = "ADDRESS"]  
    SUBJECT="SUBJECT"  
    BODY="MESSAGE"  
    [CONTENTTYPE="TYPE"] />
```

Parameters

TO (required)

A comma-separated list of the recipients to receive the message

REPLYTO (optional)

The email address you wish replies to go to. If a value is not specified, replies go to the sender's address.

SUBJECT (required)

Subject of the message.

BODY (required)

Body of the mail message.

CONTENTTYPE (optional)

The type of content you are sending. The default is `plain/text`.

Description

The SendMail method sends an SMTP message to one or more recipients. To send mail, the Content Server properties file must contain valid values for the `cs.emailhost` and `cs.emailreturnto` properties. To receive mail, the Content Server properties file must contain valid values for the `cs.emailhost`, `cs.emailaccount`, and `cs.emailpassword` properties.

errno

The possible values of `errno` include:

Value	Description
-202	Cannot send mail.

Example

This example sends an e-mail message to joeuser and janeuser. The body of the message contains the current value of variable FOO. The syntax of all parameter values must be SMTP mail compliant.

```
<SETVAR NAME="FOO" VALUE="Congratulations on becoming a
member."/>
<SENDMAIL TO="joeuser,janeuser"
SUBJECT="Test Message"
BODY="Variables.FOO"/>
```

See Also

[EMAILVENT](#)

SETCOOKIE

Sets a cookie on the client.

Syntax

```
<SETCOOKIE NAME="COOKIE_NAME"
VALUE="COOKIE_VALUE"
TIMEOUT="EXPIRATION"
SECURE="SECURE_VAL "
[URL="URL_PATH"]
[DOMAIN="DOMAIN_NAME"]/>
```

Parameters

NAME (required)

Name of the cookie to set.

VALUE (required)

Value of the cookie being set.

TIMEOUT (required)

Timeout value of cookie, in seconds. After the cookie has expired, it no longer exists.

Note

To produce a session-long cookie, set the timeout value to -1.

SECURE (required)

Indicates whether security is set on this cookie. Possible values are "true" or "false".

URL (optional)

Valid URL. This value restricts the sending of the cookie from the client to this URL only.

DOMAIN (optional)

Valid domain name. This value restricts the sending of the cookie from the client to addresses in this domain only.

Description

The SETCOOKIE tag sets a cookie on the client. SETCOOKIE must appear before any tag that streams back data on the page or the cookie is not set. This is an HTTP requirement.

Note

When setting cookies it is preferable to use CookieServer rather than the SETCOOKIE tag.

errno

The possible values of errno include:

Value	Description
-202	Cannot send mail.

Example

This example sets the cookie mycookie on the client with the value frank. The cookie expires 600 seconds from the time it is created.

```
<SETVAR NAME="errno" VALUE="0"/>
<SETCOOKIE NAME="mycookie" VALUE="frank"
  TIMEOUT="600" SECURE="false"/>
<HTML>
  <HEAD>
    <TITLE>Set mycookie</TITLE>
  </HEAD>
  <BODY>
    The cookie, mycookie, has been set.
  </BODY>
</HTML>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>
```

See Also

[REMOVECOOKIE](#)
[SETCOOKIE](#)

SETCOUNTER

Creates a counter.

Syntax

```
<SETCOUNTER NAME="COUNTER_NAME"
            VALUE="COUNTER_VALUE"/>
```

Parameters

NAME (required)

Name of the counter to set.

VALUE (required)

Integer value of counter.

errno

The possible values of errno include:

Value	Description
-501	Bad counter value.

Example

This example creates a counter frank, assigns that value to another counter, fred, and displays the value of fred.

```
<SETVAR NAME="errno" VALUE="0"/>
<SETCOUNTER NAME="frank" VALUE="10"/>
<SETCOUNTER NAME="fred" VALUE="Counters.frank"/>
<P>Fred is <CSVAR NAME="Counters.fred"/></P>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>
```

See Also

[INCCOUNTER](#)

[REMOVECOUNTER](#)

SETDATE

Sets or clears the Content Server date/time property.

Syntax

```
<SETDATE DATE="DATE"/>
```

Parameters

DATE (required)

Date to set the date property to. Use the format: yyyy-mm-dd hh:mm:ss. To restore the date/time property to use the current system date/time, set DATE to blank using an escaped value:

```
<SETDATE DATE="%20"/>
```

Description

The SETDATE tag sets or clears the Content Server date/time property. Date built-ins can then be used to display various date components. Clearing the date property restores the property to use the current system date/time.

errno

The possible values of errno include:

Value	Description
-106	Bad date syntax.

Example

This example sets the date to Feb 1 2001, displays the hour according to the date set, then restores the date property to use the current system date/time.

```
<SETVAR NAME="errno" VALUE="0"/>
<SETDATE DATE="2001-02-01 12:24:01"/>
<CSVAR NAME="CS.Hour"/>
<SETDATE DATE="%20"/>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>
```

See Also

[CS.Date](#)

SETRow

Sets the current row in a list to the first, last, next or previous row.

Syntax

```
<SETRow LIST="LIST_NAME"
      ACTION="ACTION_VAL"/>
```

Parameters

LIST (required)

Name of list.

ACTION (required)

Directive to set the current row. Possible values are:

- **NEXT** - Set current row to next row in list.
- **PREV** - Set current row to previous row in list.
- **FIRST** - Set current row to first row in list.
- **LAST** - Set current row to last row in list.

Description

The **SETRow** tag sets the current row in a list to the first, last, next, or previous row.

The column values of the current row of a list can be obtained using the syntax, *List.colname* where *List* is the name of the list and *colname* is the name of the column in the list.

Note

In the case of *StringLists*, the values can be obtained by using *List.Item*.

You can use the following attributes to return information about the list using the syntax, *List.attname*, where *attname* can be one of the following attributes:

#numRows	Number of rows in this list
#curRow	Current row in this list.
#moreRows	Boolean (true/false) to indicate whether there are more rows following the current row.
#numCols	Number of columns in this list.
@<colname>	Only used on lists that contain resultsets. Contents of the file in the specified column if the colname begins with "url".

For example, *MyList.#numRows* returns the number of rows in the list *MyList*. *MyList.@urlheader* retrieves the contents of the file specified by the column *urlheader*.

The value of a list is maintained during the processing of a page request. If the list contains a resultset, the resultset is maintained independently of a list that uses it.

errno

The possible values of `errno` include:

Value	Description
2	No row increment.
-500	No such row.
-503	No such list.

Example

This example displays a value from the first three rows of "mylist".

```
<SETROW LIST="mylist" ACTION="FIRST"/>
  First Row: <CSVAR NAME="ListName.val"/>
<SETROW LIST="mylist" ACTION="NEXT"/>
  Second Row: <CSVAR NAME="ListName.val"/>
<SETROW LIST="mylist" ACTION="NEXT"/>
  Third Row: <CSVAR NAME="ListName.val"/>
```

See Also

[GOTOROW](#)
[SELECTTO](#)
[STRINGLIST](#)

SETSSVAR

Sets the value of a session variable.

Syntax

```
<SETSSVAR NAME="SSVAR_NAME"
          VALUE="SSVAR_VALUE"/>
```

Parameters

NAME (required)
 Name of the session variable to set.

VALUE (required)
 Value of the session variable.

Description

The `SETSSVAR` tag sets the value of a session variable. The value of the session variable exists for the duration of the session unless it is explicitly deleted using `REMOVESSVAR`.

errno

The value of `errno` is unchanged.

Example

```
<SETSSVAR NAME="color"
          VALUE="red"/>
```

See Also

[SETVAR](#)
[REMOVEVAR](#)

SETVAR

Sets the value of a variable.

Syntax

```
<SETVAR
  NAME="VAR_NAME"
  VALUE="VAR_VALUE" />
```

Parameters

NAME (required)
 Name of the variable to set.

VALUE (required)
 Value to set the variable to.

Description

The `SETVAR` tag sets the value of a variable. The value of the variable exists for the duration of the page evaluation unless it is explicitly deleted using `REMOVEVAR`. For more information about variables, see the *CSEE Developer's Guide*.

errno

The value of `errno` is unchanged.

Example

This example sets the variable "color" to "red".

```
<SETVAR NAME="color" VALUE="red"/>
```

The next example assumes that the variable "color" has been set to "red" and appends "and blue".

```
<SETVAR NAME="color" VALUE="Variables.color and blue"/>
<CSVAR NAME="Variables.color"/>
```

The browser displays:

```
red and blue
```

If a variable was set by loading a file or URL and the content contains `Variables.varname` syntax, then the variable values can be resolved by:

```
<SETVAR NAME="myvar" VALUE="Variables.myvar"/>
```

Any variables as part of the value of `myvar` are resolved before the value is reset.

See Also

[REMOVEVAR](#)
[SETSSVAR](#)

SETVERSIONS

Sets the number of revisions stored for each row in a tracked table.

Syntax

```
<SETVERSIONS TABLE="TABLE_NAME"
    [NUMVERSIONS="MAX_VERSIONS"]/>
```

Parameters

TABLE (required)

Table to modify maximum number of versions

NUMVERSIONS (optional)

Maximum number of versions to store for each row. When the maximum is reached, the oldest version is removed. A value of 0 (zero) indicates an unbounded number of versions stored. If `MAX_VERSIONS` is smaller than the current number of versions on the table, those versions remain and can be removed manually.

Description

The `SETVERSIONS` tag sets the number of revisions stored for each row in a tracked table. The current user must have `rtadmin` privileges on the table.

errno

The possible values of `errno` include:

Value	Description
-1000	Command failure. Check the <code>errdetail</code> variable for the specific error.

Example

This example assumes all `SETVERSION` parameters are exposed on an HTML form. It first checks whether the parameters exist, clears any values from `errno`, and then sets the maximum number of revisions stored on the table.

```
<!-- Make sure required parameters exist -->
<IF COND="IsVariable.tablename=false">
    <THEN>
        <SETVAR NAME="tablename" VALUE=""/>
    </THEN>
```

```

</IF>
<IF COND="IsVariable.numversions=false">
  <THEN>
    <SETVAR NAME="numversions" VALUE=""/>
  </THEN>
</IF>
<!-- clear out errno -->
<SETVAR NAME="errno" VALUE="0"/>
<!-- do the track -->
<SETVERSIONS TABLE="Variables.tablename"
  NUMVERSIONS="Variables.numversions"/>
<!-- See if set succeeded -->
<IF COND="Variables.errno=0">
  <THEN>
    <!-- if succeeded, note it-->
    SetVersions succeeded for
    <CSVAR NAME="Variables.tablename"> <BR/>
  </THEN>
<ELSE>
  <!-- otherwise, call an error reporting element -->
  SetVersions failed.<BR/>
</ELSE>
</IF>

```

SQLEXP

Creates a string that can be used as part of an SQL WHERE clause.

Syntax

```

<SQLEXP OUTSTR="STRING_NAME"
  TYPE="TYPE_VAL "
  VERB="VERB_VAL "
  STR="VALUE_LIST"
  COLNAME="COL_NAME"/>

```

Parameters

OUTSTR (required)

Name of variable used to store resulting string.

TYPE (required)

Logical operator to use between relational comparison. Possible values are "AND" or "OR", and "IN".

VERB (required)

Relational operator to use. Possible values are:

- "IN" - items is in the source list.
- "NOT IN" - items are not in the source list.
- "LIKE" - item is like column
- "=" - item equals column
- "!=" - item does not equal column.

- "%3c" (hex <) - item less than column.
- "%3e" (hex >) - item greater than column.

STR (required)

A comma-separated list of items to test against a column name.

COLNAME (required)

Column name to use in relational comparison.

Description

The SQLEXP tag creates a string that can be used as part of an SQL WHERE clause. SQLEXP takes a comma-separated string of items and a column name and places a relational operator between each item/column name pair. A logical operator is placed between each item/column name comparison.

Note

Before using SQLEXP, you must set a variable called `tablename`. This enables SQLEXP to automatically quote values for text columns but not for numerical columns.

errno

The possible values of `errno` include:

Value	Description
-100	General error.
-103	Bad or missing table name.
-106	Bad column name.

Example

This example constructs a WHERE clause for use by EXECSQL.

```
<SETVAR NAME="tablename" VALUE="colors"/>
<SQLEXP OUTSTR="mywhere" TYPE="OR"
  VERB="LIKE" STR="red,blue" COLNAME="color"/>
<!--
  The value of mywhere is:
  color LIKE '%red%' OR color LIKE '%blue%'
-->
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
  <ELSE>
    <EXECSQL SQL="SELECT * FROM colors
      WHERE Variables.mywhere" LIST="colorlist"/>
  </ELSE>
</IF>
```


This example for SQLEXP sets the variable mywhere to:

```
username NOT IN ('joe','jim','sue')

<SETVAR NAME="tablename" VALUE="SystemUsers" />
<SQLEXP OUTSTR="mywhere"
      TYPE="IN"
      VERB="NOT IN"
      STR="joe,jim,sue"
      COLNAME="username" />
```

See Also

[EXECSQL](#)

STRINGLIST

Constructs a one-column list from a string separated by a delimiter.

Syntax

```
<STRINGLIST
  NAME="LIST_NAME"
  STR="STR_VALUE"
  DELIM="DELIMITER" />
```

Parameters

NAME (required)

Name of the string list.

STR (required)

Values to include in the string.

DELIM (required)

Delimiter to use. To use a blank as a delimiter, you must use space character; for example, %20.

Description

The **STRINGLIST** tag constructs a one-column list from a string separated by a delimiter. The list created always contains the column **ITEM**. The following delimiters are some of the more common ones supported:

%22	quotecstr (")
%20	emptystr (nothing)
%0a	newline (\n)
%09	tab (\t)
%0d	carriage return (\r)
%3c	less than sign (<)
%3e	greater than sign (>)

%26	ampersand (&)
%25	percent (%)

The column values of the current row in a list can be obtained using the syntax, *List.colname* where *List* is the name of the list and *colname* is the name of the column in the list.

Note

In the case of *StringLists*, the values can be obtained by using *List.Item*.

You can use the following attributes to return information about the list using the syntax, *List.attname*, where *attname* can be one of the following attributes:

#numRows	Number of rows in this list
#curRow	Current row in this list.
#moreRows	Boolean (true/false) to indicate whether there are more rows following the current row.
#numCols	Number of columns in this list.
@<colname>	Only used on lists that contain resultsets. Contents of the file in the specified column if the colname begins with "url".

For example, `MyList.#numRows` returns the number of rows in the list `MyList`. `MyList.@urlheader` retrieves the contents of the file specified by the column `urlheader`.

The value of a list is maintained during the processing of a page request. If the list contains a resultset, the resultset is maintained independently of a list that uses it.

errno

The possible values of `errno` include:

Value	Description
-16	Output list is empty.

Example

This example constructs a list called `tablenames` using the values in `Variables.list` and a blank as the delimiter. The loop iterates through the list of names in the table, displaying each item in the list.

```
<SETVAR NAME="errno" VALUE="0"/>
The Tables are: <br/>
```

```

<STRINGLIST NAME="tablenames" STR="Variables.list" DELIM="%20"/>
<
<LOOP LIST="tablenames">
    <CSVAR NAME="tablenames.ITEM"/><br/>
</LOOP>
<IF COND="IsError.Variables.errno=true">
    <THEN>
    <!--Handle error-->
    </THEN>
</IF>

```

This example constructs a list called `somelist` using the values `erin`, `alli`, and `ann` and a comma as the delimiter. The loop iterates through the list of names and displays them.

```

<SETVAR NAME="errno" VALUE="0"/>
<STRINGLIST NAME="somelist"
    STR="erin,alli,ann"
    DELIM=", "/>
NAMES:<br/>
    <LOOP LIST = "somelist">
        <CSVAR NAME="somelist.ITEM"/>
    </LOOP>
<IF COND="IsError.Variables.errno=true">
    <THEN>
    <!--Handle error-->
    </THEN>
</IF>

```

See Also

[RENAMELIST](#)
[ISINLIST](#)

SUBSTITUTE

Substitutes all occurrences of the specified substring within a string.

Syntax

```

<SUBSTITUTE STR="STRING_NAME"
    WHAT="WHAT_STRING"
    [WITH ="REPLACE_STRING"]
    OUTSTR="OUTSTR_NAME"/>

```

Parameters

STR (required)

String that contains substring to substitute.

WHAT (required)

Substring to replace within `STRING_NAME`. This parameter can contain escapable characters, such as `%20` for a space. For example, to remove all spaces and replace them with blanks, use `WHAT="%20" WITH=""`.

WITH (optional)

Replacement value of the substring. This parameter can contain escapable characters, such as %20 for a space. For example, to remove all spaces and replace them with blanks, use `WHAT="%20" WITH=""`.

OUTSTR (required)

Resulting string after substitution.

Description

The `SUBSTITUTE` tag substitutes all occurrences of the specified substring within a string. The substring comparison is case-sensitive. For more information about variables, see the *CSEE Developer's Guide*.

errno

The value of `errno` is unchanged.

Example

This example replaces all occurrences of the string `test` within the string `bar`.

```
<SETVAR NAME="bar"
      VALUE="This is a test of a test."/>
Bar is: <CSVAR NAME="Variables.bar"/> <br/>
<SUBSTITUTE STR="Variables.bar"
      OUTSTR="foo"
      WHAT="test"
      WITH="none"/>
And foo is: <CSVAR NAME="Variables.foo"/>
```

The browser displays:

```
Bar is: This is a test of a test.
And foo is: This is none of a none.
```

See Also

[SUBSTRING](#)

SUBSTRING

Constructs a new variable that contains a substring.

Syntax

```
<SUBSTRING
  STR="string value"
  OUTSTR="outstring name"
  [INDEX="OFFSET"]
  [ENDINDEX="END_OFFSET"]/>
```

Parameters

STR (required)

String containing substring.

OUTSTR (required)

Name of variable to contain substring.

INDEX (optional)

Beginning offset in STR. The default for this parameter is 0.

ENDINDEX (optional)

Ending offset in STR. The default for this parameter is the length of STR.

Description

The SUBSTRING tag constructs a new variable containing a substring. For more information about variables, *CSEE Developer's Guide*.

errno

The possible values of errno include:

Value	Description
-50	Bad start index.
-51	Bad end index.
-52	Empty output string.

Example

This example places the value "This is a test" in NewString. The value of the variable NewString is test after this tag executes. An error is set if either of the supplied index values is out of range.

```
<SETVAR NAME="errno" VALUE="0"/>
<SUBSTRING STR="This is a test"
  OUTSTR="NewString"
  INDEX="10"
  ENDINDEX="14"/>
<IF COND="IsError.Variables.errno=true">
  <THEN>
```

```

        <!--Handle error-->
    </THEN>
</IF>

```

See Also

[SUBSTITUTE](#)

SYSEXEC

Executes a system-dependent shell-level command.

Syntax

```

<SYSEXEC
    CMD="command"
    [SYSOUT="output return"]
    [SYSERR="error return"]
    [WAIT="wait val"]/>

```

Parameters

CMD (required)

Shell-level command to execute.

SYSOUT (optional)

Variable name to contain the system output returns. If not set, these returns are unavailable. Specify SYSOUT and/or SYSERR arguments for this tag only if WAIT is true, otherwise, errno -14 is set.

SYSERR (optional)

Variable name to contain the system error returns. If not set, these returns are unavailable. Specify SYSOUT and/or SYSERR arguments for this tag only if WAIT is true, otherwise, errno -14 is set.

WAIT (optional)

Indicates whether to execute the command immediately or wait. Possible values are true or false. The default is false. You could create an infinite loop if you set this parameter to true for NT Command shell commands.

errno

The possible values of errno include:

Value	Description
-14	Process exception error.

Example

This example executes a Windows NT shell-level directory command on the value of Variables.testfile (replaced before the command executes).

```

<SYSEXEC CMD="cmd /c dir /OD Variables.testfile"
    SYSOUT="execout"
    SYSERR="execerr"

```

```

        WAIT="true"/>
<SYSEXEC CMD="ls Variables.testfile"
        SYSOUT="execout"
        SYSERR="execerr"
        WAIT="true"/>

```

Your code should check to see if the variables exist (contain data) before printing, for example:

```

<IF COND="IsVariable.execerr=true">
  <THEN>
    Error string returned: <CSVAR NAME="Variables.execerr"/>
  </THEN>
</IF>

```

See Also

[CALLJAVA](#)

THROWEXCEPTION

Halts further processing of the element.

Syntax

```
<THROWEXCEPTION/>
```

Parameters

None.

Description

The THROWEXCEPTION tag halts further processing of the element. Text streamed prior to the THROWEXCEPTION in the element appears in the output.

errno

The value of errno is unchanged.

Example

This example checks the value of errno and uses THROWEXCEPTION if the value is not 0 (zero). If errno does not equal 0, the message is not be displayed.

```

<IF COND="Variables.errno!=0">
  <THEN>
    <THROWEXCEPTION/>
  </THEN>
</IF>

```

Operation was successful.

TIME.GET

Gets a time difference.

Syntax

```
<TIME.GET  
    NAME="some name"  
    [OUTPUT="some output"]/>
```

Parameters

NAME (required)

Specifies a name that is associated with the initial time.

OUTPUT (optional)

Specifies a variable name to use for output of the elapsed time. If this parameter is not specified, the elapsed time is streamed.

Description

This tag streams the elapsed time in milliseconds, from when the time was last set.

errno

None.

Example

The following example streams the elapsed time from when `time1` was set using the `TIME.SET` tag:

```
<TIME.GET NAME="time1"  
    OUTPUT="elapsed"/>  
<CSVAR NAME="Variables.elapsed/>
```

See Also

[TIME.SET](#)

TIME.SET

Names the current time.

Syntax

```
<TIME.SET  
    NAME="some name"/>
```

Parameters

NAME (required)

Specifies the name to be associated with the initial time.

Description

This tag associates the current time with a name.

errno

None.

Example

The following example saves the correct time as `time1`.

```
<TIME.SET NAME="time1"/>
```

TOLOWER

Converts a string to all lowercase.

Syntax

```
<TOLOWER  
    STR="string value"  
    OUTSTR="outstring name"/>
```

Parameters

STR (required)

String to convert to lowercase.

OUTSTR (required)

Name of the string to contain results.

errno

The value of `errno` is unchanged.

Example

This example changes the case in the string "This Is A Test". After the command executes, `Variables.newstring` has the value "this is a test".

```
<TOLOWER STR="This Is A Test" OUTSTR="newstring"/>  
<CSVAR NAME="Variables.newstring"/>
```

See Also

[TOUPPER](#)

TOUPPER

Converts a string to all uppercase.

Syntax

```
<TOUPPER  
    STR="string value"  
    OUTSTR="outstring name" />
```

Parameters

STR (required)

String to convert to uppercase.

OUTSTR (required)

Name of the variable to contain results.

errno

The value of errno is unchanged.

Example

This example changes the case in the string "this is a test". After the command executes, Variables.newstring contains the value "THIS IS A TEST".

```
<TOUPPER STR="this is a test" OUTSTR="newstring" />  
<CSVAR NAME="Variables.newstring" />
```

See Also

[TOLOWER](#)

TRACKTABLE

Enables revision tracking operations on a table.

Syntax

```
<TRACKTABLE  
    TABLE="table name"  
    [NUMVERSIONS="max versions"]  
    STORAGE="storage folder" />
```

Parameters

TABLE (required)

Table to track.

NUMVERSIONS (optional)

Maximum number of versions to store for each row. When the maximum is reached, the oldest version is removed.

STORAGE (required)

Location to store revisions associated with a row in the table. This is the root folder. A folder is created under the given root folder whose name is the same as the table being tracked.

Description

The TRACKTABLE tag enables revision tracking operations on a table. The maximum number of revisions per row can be set as well as the storage location for the revisions.

errno

The possible values of errno include:

Value	Description
-1000	Command failure. Check the errdetail variable for the specific error.

Example

This example assumes all TRACKTABLE parameters are exposed on an HTML form. It first checks whether the parameters exist, clears any values from errno, then tracks the given table.

```
<!-- Make sure required parameters exist -->
<IF COND="IsVariable.tablename=false">
  <THEN>
    <SETVAR NAME="tablename" VALUE=""/>
  </THEN>
</IF>
<IF COND="IsVariable.numversions=false">
  <THEN>
    <SETVAR NAME="numversions" VALUE=""/>
  </THEN>
</IF>
<IF COND="IsVariable.storage=false">
  <THEN>
    <SETVAR NAME="storage" VALUE=""/>
  </THEN>
</IF>
<!-- clear out errno -->
<SETVAR NAME="errno" VALUE="0"/>
<!-- do the track -->
<TRACKTABLE TABLE="Variables.tablename"
  NUMVERSIONS="Variables.numversions"
  STORAGE="Variables.storage"/>
<!-- See if track succeeded -->
<IF COND="Variables.errno=0">
  <THEN>
    <!-- if succeeded, brag about it -->
```

```

        TrackTable succeeded for
        <CSVAR NAME="Variables.tablename"/><BR/>
    </THEN>
    <ELSE>
        <!-- otherwise, call an error reporting element -->
        TrackTable failed.<BR/>
    </ELSE>
</IF>

```

TREEMANAGER

Calls the TreeManager servlet.

Syntax

```

<TREEMANAGER>
  <[ARGUMENT NAME="somearg" VALUE="somevalue"]/>
  <[CGIARGUMENT NAME="somecgiarg"
    VALUE="somecgivalue"]/>
</TREEMANAGER>

```

Parameters

ARGUMENT (optional)

Arguments passed to the TreeManager as it is called.

CGIARGUMENT (optional)

CGI arguments passed to the TreeManager.

SCOPED

Valid options for this attribute are as follows:

- GLOBAL - The default behavior
- LOCAL - Arguments passed in are restored (or destroyed) on return from the call.
- STACKED - The child only sees the explicit arguments passed in and can only return values in that same namespace.

Description

The TREEMANAGER tag calls the TreeManager servlet. The GLOBAL value of the SCOPED parameter does not protect input variables. Using LOCAL or STACKED for the SCOPED argument protects those variables from change, but disallows sharing of input space. Use <CGIARGUMENT NAME="name"/> to specify that specific CGI parameters be made available to the CatalogManager. Use [ARGUMENT](#) for specific string values.

By omitting SCOPED (or setting it to LOCAL) you can cause the CatalogManager to share all the existing input variables without scoping. Note that this means specified arguments overwrite existing variables. Using SCOPED="shared" means that you need not specify any argument; CGIARGUMENTS, specifically, are ignored. Note that variables set since program execution began are reset to the original values passed on the URL line. Use with care.

Use CGIARGUMENT to specify that specific CGI parameters be made available to EvalServer. Use [ARGUMENT](#) for specific string values.

Several TREEMANAGER commands ([addchild](#), [addchildren](#), [findnode](#), [getchildren](#), [getnode](#), [getparent](#), [listtrees](#), [nodepath](#)) return a list. Each of those commands has a `treename` argument. The name of the returned list is the value passed in the `treename` argument. In the following example, the tree name and the returned list name is `MyTree`:

```
<TREEMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="getparent"/>
  <ARGUMENT NAME="treename" VALUE="MyTree"/>
  <ARGUMENT NAME="node" VALUE="Variables.MyParent"/>
</TREEMANAGER>
```

The following columns are common to lists returned from every command:

- `nid` – Tree node ID.
- `nparentid` – Parent tree node ID. 0 (zero) means it is a root node.
- `nrank` – Application-specific rank of the node.
- `otype` – Object table referenced by this node.
- `oid` – Object ID referenced by this node.
- `oversion` – Reserved for future use by FatWire.
- `ncode` – Application-specific code related to the node.

Refer to a specific command for more information about the list that is returned by that command.

The following is a list of commands that you use with the TREEMANAGER tag:

- [addchild](#)
- [addchildren](#)
- [copychild](#)
- [createtree](#)
- [delchild](#)
- [delchildren](#)
- [deletetree](#)
- [findnode](#)
- [getchildren](#)
- [getnode](#)
- [getparent](#)
- [listtrees](#)
- [movechild](#)
- [nodepath](#)
- [setobject](#)
- [validatenode](#)
- [verifypath](#)

errno

The value of errno is unchanged.

Example

```

<SETVAR NAME="errno" VALUE="0"/>
<TREEMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="addchild" />
  <ARGUMENT NAME="treename" VALUE="TreetestTree"/>
  <ARGUMENT NAME="nparentid"
    VALUE="Variables.testroot" />
  <ARGUMENT NAME="otype" VALUE="TreetestObjects"/>
  <ARGUMENT NAME="oid" VALUE="3" />
  <ARGUMENT NAME="oversion" VALUE="1" />
  <ARGUMENT NAME="nrnk" VALUE="2" />
  <ARGUMENT NAME="ncode" VALUE="0" />
</TREEMANAGER>
<!-- Check status -->
<IF COND="Variables.errno=0">
  <THEN>
    Child of:
    <CSVAR NAME="Variables.testroot"/>
    successfully added.
  </THEN>
</IF>

```

See Also

[CONTENTSERVER \(Deprecated\)](#)

[DEBABBLE](#)

[CATALOGMANAGER](#)

addchild

Adds a node to a tree table.

Syntax

```

<TREEMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="addchild"/>
  <ARGUMENT NAME="treename" VALUE="name"/>
  <ARGUMENT NAME="parentid" VALUE="node ID"/>]
  [<ARGUMENT NAME="otype" VALUE="object table"/>]
  [<ARGUMENT NAME="oversion" VALUE="object version"/>]
  [<ARGUMENT NAME="nrnk" VALUE="node rank"/>]
</TREEMANAGER>

```

Parameters

ftcmd (required)

Value must be set to addchild.

treename (required)
Name of the tree.

parentid (optional)
Node ID of the parent node. Omit or set to NULL to make this node a top level node.
Top level nodes are identified by a 0. Must be an integer.

otype (optional)
Name of an object table.

oid (optional)
Identifies the current object on this node. The oid should correspond to a value in the object table's id column.

oversion (optional)
Numeric value that specifies the version of the object. This parameter is not supported for this release of Content Server, and is reserved for use in later versions.

ncode (optional)
16-character string value that is available for use by the application.

nrank (optional)
Numeric value of the rank to assign this node.

Description

The `addchild` command adds a node to a tree table.

After executing the `addchild` command, a list named after the tree table or found in a variable called `treename` is created. A unique identifier for the node in the tree, the `nid` value, is generated when you successfully add a node. The value of `nid` can be retrieved in the tree result list.

errno

The possible values of `errno` include:

Value	Description
-3	No access privileges to table.
-103	No table.
-105	Database error.
-106	Missing or invalid parameter.
-110	Invalid tree name.

Example

The following example adds a single record in the tree table that contains all the necessary information to identify the object in the tree. Note that `oversion` is passed a null value to ensure future product compatibility.

```
<TREEMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="addchild" />
  <ARGUMENT NAME="treename"
    VALUE="TreeMgrCatalog" />
  <ARGUMENT NAME="nparentid"
```

```

        VALUE="Variables.parent" />
    <ARGUMENT NAME="otype"
        VALUE="TreeObjectCatalog" />
    <ARGUMENT NAME="oid" VALUE="Variables.id" />
    <ARGUMENT NAME="oversion"
        VALUE="Variables.empty" />
    <ARGUMENT NAME="nrank" VALUE="2" />
    <ARGUMENT NAME="ncode" VALUE="testpage name" />
</TREEMANAGER>
<IF COND="IsList.TreeMgrCatalog=true">
    <THEN>
        <IF COND="Variables.errno!=-101">
            <THEN>
                <LOOP LIST="TreeMgrCatalog">
                    <SETVAR NAME="mychild"
                        VALUE="TreeMgrCatalog.nid"/>
                </LOOP>
            </THEN>
        </IF>
    </THEN>
<ELSE>
    <!-- The list did not get created so add -->
    <!-- handling here -->
    Failed to add a child to the tree table.
    Error number - <CSVAR NAME="Variablese.errno"/>
    <BR/>
</ELSE>
</IF>

```

See Also

[addchildren](#)
[delchild](#)

addchildren

Adds multiple nodes to a tree table.

Syntax

```

<TREEMANAGER>
    <ARGUMENT NAME="ftcmd" VALUE="addchildren"/>
    <ARGUMENT NAME="treename" VALUE="name"/>
    <ARGUMENT NAME="parentnodeN" VALUE="node_id"/>
    [<ARGUMENT NAME="otypeN" VALUE="object table"/>]
    [<ARGUMENT NAME="oidN" VALUE="row number"/>]
    [<ARGUMENT NAME="oversionN" VALUE="object version"/>]
    [<ARGUMENT NAME="ncodeN" VALUE="some string"/>]
    [<ARGUMENT NAME="nrankN" VALUE="node rank"/>]
</TREEMANAGER>

```


Parameters

ftcmd (required)

Value must be set to addchildren.

treename (required)

Name of the tree.

parentnodeN (required)

nid of the parent node. The value *N* corresponds to the row number to be added starting from 0 (zero). Top level nodes are identified by a zero value. Must be an integer. Unlike the addchild command, nparentidN must be provided and cannot be left NULL.

otypeN (optional)

Name of an object table. The value *N* corresponds to the row number to be added starting from 0 (zero).

oidN (optional)

Identifies the current object on this node. The oid should correspond to a value in the object table's id column. The value of *N* corresponds to the row number to be added starting from 0 (zero).

oversionN (optional)

A numeric value that specifies the version of the object. This parameter is not supported for this release of Content Server, and is reserved for use in later versions.

ncodeN (optional)

A 16-character string value that can be used by an application. The value of *N* corresponds to the row number to be added starting from 0 (zero).

nrankN (optional)

A numeric value of the rank to assign this node. The value of *N* corresponds to the row number to be added starting from 0 (zero).

errno

The possible values of errno include:

Value	Description
-3	No access privileges to table.
-103	No table.
-105	Database error.
-106	Missing or invalid parameter.
-110	Invalid tree name.

Example

The following example adds two nodes to the tree table and then browses the list to retrieve the two nodes' nid values for future use.

```
<TREEMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="addchildren" />
  <ARGUMENT NAME="treename"
    VALUE="TreeMgrCatalog" />
```

```

    <ARGUMENT NAME="nparentid0" VALUE="0" />
    <ARGUMENT NAME="otype0"
        VALUE="TreeObjectCatalog" />
    <ARGUMENT NAME="oid0"
        VALUE="Variables.myfirstobjID" />
    <ARGUMENT NAME="oversion0"
        VALUE="Variables.empty" />
    <ARGUMENT NAME="nranks0" VALUE="101" />
    <ARGUMENT NAME="ncode0"
        VALUE="top level node" />
    <ARGUMENT NAME="nparentid1"
        VALUE="parent.nid" />
    <ARGUMENT NAME="otype1"
        VALUE="TreeObjectCatalog" />
    <ARGUMENT NAME="oid1"
        VALUE="Variables.mysecondobjID" />
    <ARGUMENT NAME="oversion1"
        VALUE="Variables.empty" />
    <ARGUMENT NAME="nranks1" VALUE="202" />
    <ARGUMENT NAME="ncode1"
        VALUE="second level node" />
</TREEMANAGER>
<IF COND="IsError.Variables.errno !=true">
    <THEN>
        <LOOP LIST="Variables.treename">
            <IF COND="Variables.treename.nrank=101">
                <THEN>
                    <SETVAR NAME="topid"
                        VALUE="Variables.treename.nid"/>
                </THEN>
            </IF>
            <IF COND="Variables.treename.nrank=202">
                <THEN>
                    <SETVAR NAME="topid"
                        VALUE="Variables.treename.nid"/>
                </THEN>
            </IF>
        </LOOP>
    </THEN>
<ELSE>
    <!-- add error handling logic here -->
</ELSE>
</IF>

```

See Also

[addchild](#)
[createtree](#)

copychild

Copies one or more nodes from one tree to another tree.

Syntax

```
<TREEMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="copychild"/>
  <ARGUMENT NAME="treename" VALUE="name"/>
  <ARGUMENT NAME="node" VALUE="node ID"/>
  <ARGUMENT NAME="newtreename" VALUE="treename"/>
  [<ARGUMENT NAME="nparentid" VALUE="node ID"/>]
</TREEMANAGER>
```

Parameters

ftcmd (required)

Value must be set to copychild.

treename (required)

Name of the source tree from which the node is copied.

node (required)

Node ID of the node to copy.

newtreename (required)

Name of the destination tree where the node is copied.

nparentid (optional)

Node ID of the new parent of the node in the destination tree (newtreename). Omit or set NULL to make this node a top-level node.

Description

The copychild command copies one or more nodes from one tree to another tree.

Multiple nodes are copied if the node is the parent of one child or a number of children.

errno

The possible values of errno include:

Value	Description
-3	No access privileges to table.
-103	No table.
-105	Database error.
-106	Missing or invalid parameter.
-110	Invalid tree name.

Example

```
<TREEMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="copychild" />
  <ARGUMENT NAME="treename"
    VALUE="TreeMgrCatalog" />
  <ARGUMENT NAME="newtreename"
    VALUE="TreeMgrCatalog2" />
  <ARGUMENT NAME="node"
    VALUE="Variables.moveme" />
  <ARGUMENT NAME="nparentid" VALUE="0" />
</TREEMANAGER>
<IF COND="IsError.Variables.errno=false">
  <THEN>
    <BR/>copy succeeded<BR/>
    Copy tree node is now
    <CSVAR NAME="Variables.newtreename.nid"/>
    <BR/>
  </THEN>
  <ELSE>
    <BR/>copy failed<BR/>
  </ELSE>
</IF>
```

See Also

[getchildren](#)
[movechild](#)

createtree

Creates a new tree table.

Syntax

```
<TREEMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="createtree"/>
  <ARGUMENT NAME="treename" VALUE="name"/>
  [<ARGUMENT NAME="aclList" VALUE="ACLs"/>]
</TREEMANAGER>
```

Parameters

ftcmd (required)
 Value must be set to createtree.

treename (required)
 Name of the tree.

aclList (optional)
 ACLs that can access this tree.

Description

The `createtree` tag creates a new tree table. The system constructs a new table that represents a tree and registers it in the `SystemInfo` table as a tree table.

errno

The possible values of `errno` include:

Value	Description
-3	No access privileges to table.
-22	Table already exists.

Example

```
<TREEMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="createtree" />
  <ARGUMENT NAME="treename" VALUE="TreetestTree" />
  <ARGUMENT NAME="aclList" VALUE="Browser,ForumUser" />
</TREEMANAGER>
```

See Also

[deletetree](#)

delchild

Deletes a node from a tree.

Syntax

```
<TREEMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="delchild"/>
  <ARGUMENT NAME="treename" VALUE="name"/>
  <ARGUMENT NAME="node" VALUE="nodeid"/>
</TREEMANAGER>
```

Parameters

ftcmd (required)
Value must be set to `delchild`.

treename (required)
Name of the tree.

node (required)
Node ID of the child to remove.

Description

The `delchild` command deletes a node from a tree. All child nodes are also removed. The objects pointed to by the nodes are not affected.

errno

The possible values of `errno` include:

Value	Description
-3	No access privileges to table.
-103	No table.
-105	Database error.
-101	No rows returned.
-111	No nodes.

Example

```
<SETVAR NAME="errno" VALUE="0"/>
<TREEMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="delchild" />
  <ARGUMENT NAME="treename" VALUE="TreetestTree" />
  <ARGUMENT NAME="node" VALUE="34253224" />
</TREEMANAGER>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>
```

See Also

[delchildren](#)
[addchild](#)

delchildren

Deletes multiple nodes from a tree.

Syntax

```
<TREEMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="delchildren"/>
  <ARGUMENT NAME="treename" VALUE="name"/>
  <ARGUMENT NAME="nodeN" VALUE="nodeid"/>
</TREEMANAGER>
```

Parameters

ftcmd (required)

Value must be set to delchildren.

treename (required)

Name of the tree.

nodeN (required)

Where *N* is 0 to the number of nodes that you want to delete.

Note that these parameters are the same as for `delchild`. Children are deleted from a single tree and the node parameter must be specified by counting from 0. See [addchildren](#) for more information.

description

The `deletetechnicalchildren` command deletes multiple nodes from a tree. Child nodes of the nodes deleted are also removed. The objects pointed to by the nodes are not affected.

errno

The possible values of `errno` include:

Value	Description
-3	No access privileges to table.
-103	No table.
-105	Database error.
-110	Invalid tree name.
-111	No nodes.

Example

```
<SETVAR NAME="errno" VALUE="0"/>
<TREEMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="delchildren" />
  <ARGUMENT NAME="treename" VALUE="TreetestTree" />
  <ARGUMENT NAME="node0" VALUE="34253224" />
  <ARGUMENT NAME="node1" VALUE="94589343" />
  <ARGUMENT NAME="node2" VALUE="83473888" />
</TREEMANAGER>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>
```

See Also

[delchild](#)
[addchildren](#)

deletetree

Deletes a tree table.

Syntax

```
<TREEMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="deletetree"/>
  <ARGUMENT NAME="treename" VALUE="name"/>
</TREEMANAGER>
```

Parameters

ftcmd (required)
Value must be set to deletetree.

treename (required)
Name of the tree.

Description

The `deletetree` command deletes a tree table. This does not affect objects referenced by the tree. The tree is deleted and is no longer registered in the SystemInfo table.

errno

The possible values of `errno` include:

Value	Description
-3	No access privileges to table.
-103	No table.
-110	Invalid tree name.

Example

```
<SETVAR NAME="errno" VALUE="0"/>
<TREEMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="deletetree" />
  <ARGUMENT NAME="treename" VALUE="TreetestTree" />
</TREEMANAGER>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>
```

See Also

[createtree](#)

findnode

Gets a list of nodes from a tree.

Syntax

```
<TREEMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="findnode"/>
  <ARGUMENT NAME="treename" VALUE="name"/>
  [<ARGUMENT NAME="where" VALUE="otype|oid|overison"/>]
  [<ARGUMENT NAME="column name" value="value"/>]
  [<ARGUMENT NAME="orderby" VALUE="some order"/>]
</TREEMANAGER>
```

Parameters

ftcmd (required)

Value must be set to `findnode`.

treename (required)

Name of the tree.

where (optional)

Values of the column names containing data to use in constructing the query. Only variables whose column names appear in this list are used in constructing the query, and only columns referring to objects are allowed.

Legal values are: `otype`, `oid`, and `overison`.

For example, `where=otype` to use the values in the variables `otype` to build the query.

orderby (optional)

Specifies the tree column in which to sort results.

Wildcards are **not** allowed in any of the variables being searched for. If a % is found in a variable whose column is specified as part of the search, it is ignored.

The name of the list that is constructed is stored in the value of the variable, `treename` (that is, `Variables.treename`).

Description

The `findnode` command gets a list of nodes from a tree. This is meant to perform a fast lookup of nodes referring to specific objects.

errno

The possible values of `errno` include:

Value	Description
-3	No access privileges to table.
-103	No table.
-110	Invalid tree name.
-111	No nodes.

Example

```
<TREEMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="findnode" />
  <ARGUMENT NAME="treename" VALUE="TreetestTree" />
  <ARGUMENT NAME="where" VALUE="otype" />
  <ARGUMENT NAME="otype" VALUE="TreetestObjects" />
  <ARGUMENT NAME="orderby" VALUE="nrank"/>
</TREEMANAGER>
<IF COND="IsError.Variables.errno=false">
  <THEN>
    <LOOP LIST="Variables.treename">
      <BR/><CSVAR NAME="Variables.treename.nid"/>
    </LOOP>
  </THEN>
  <ELSE>
    <BR/> No nodes<BR/>
  </ELSE>
</IF>
```

See Also

[getchildren](#)

getchildren

Retrieves a list of child nodes from a tree, specifically when you want to retrieve the children of a particular node (specified by nparentid).

Syntax

```
<TREEMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="getchildren"/>
  <ARGUMENT NAME="treename" VALUE="name"/>
  [<ARGUMENT NAME="orderby" VALUE="tree column"/>]
  [<ARGUMENT NAME="where" VALUE="oversion"/>]
  [<ARGUMENT NAME="where2" VALUE="SQL filter"/>]
  [<ARGUMENT NAME="depth" VALUE="recursion depth"/>]
</TREEMANAGER>
```

Parameters

ftcmd (required)

Value must be set to getchildren.

treename (required)

Name of the tree.

orderby (optional)

Specifies the tree column in which to sort results. This order is valid only if depth=1.

where (optional)

Values of the column names containing data to use in constructing the query. Only variables whose column names appear in this list are used in constructing the query.

Legal values are: `nid`, `nparentid`, `nrank`, `ncode`, `otype`, `oid`, `oversion`.

For example, `where=nparentid, otype` uses the values in the variables `nparentid` and `otype` to build the query.

`nparentid` is **always** part of the query, if it is not specified in the where clause; or, if there is no `nparentid` variable set, the value of the tree root (0) is used.

Also, `otype` **must** be specified if the join is true.

where2 (optional)

Additional SQL filtering (must be well formed SQL).

depth (optional)

The default value is 1. Limits the depth of recursion when scanning a tree. Illegal values are converted to 1. Specify -1 to remove depth restrictions on scan. Note below the impact of a deep recursive search. The depth parameter **must** be set to 1 if join is true.

width (optional)

The default value is -1; that is, no limit. Limits the number of children per parent that are returned in the list. Illegal values are converted to -1. If width is specified, it overrides the value of limit as they become interchangeable in a single depth tree. If depth is greater than 1, width, limit, and depth are all relevant in defining the values returned in the list.

join (optional)

true or false to join to an object table. The object table holds the cache information. The default value is false.

limit (optional)

Maximum rows returned in a list.

Wildcards are **not** allowed in any of the variables being searched for. If a % is found in a variable whose column is specified as part of the search, it is ignored.

Object tables that are joined to are required to be registered as type `obj` in the `SystemInfo` table in order to support the caching of resultsets that are created. You cannot join to an object table if the where clause does not specify a unique classtype; that is, the returned resultset is a combination of a `treenode` and a single object type. Specifying a join forces the depth to 1. You may only retrieve the joined results for the current node.

Scan is depth first. Recursive (`depth > 1`) scan is considerably more expensive than a flat scan. All tree scan results can be cached based on system configuration. Then name of the LIST is *value-of-treename*.

When depth is specified as a legal positive value greater than one, an additional column `depth` is added to the resulting list. The name of the LIST constructed is stored in the value of the `Variables.treename`.

Retrieves a list of child nodes from a tree, specifically when you want to retrieve the children of a particular node (specified by `nparentid`). The parent node is not returned as part of the list. The corresponding object data for the nodes may be joined to the result if desired.

Description

The `getchildren` command retrieves a list of child nodes from a tree, specifically when you want to retrieve the children of a particular node (specified by `nparentid`). The parent node is not returned as part of the list. The corresponding object data for the nodes may be joined to the result if desired.

errno

The possible values of `errno` include:

Value	Description
-3	No access privileges to table.
-103	No table.
-106	Missing or invalid parameters.
-110	Invalid tree name.
-111	No nodes.

Example

```
<SETVAR NAME="errno" VALUE="0"/>
<TREEMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="getchildren" />
  <ARGUMENT NAME="treename" VALUE="TreetestTree" />
  <ARGUMENT NAME="depth" VALUE="-1" />
  <ARGUMENT NAME="where" VALUE="otype" />
  <ARGUMENT NAME="join" VALUE="false" />
  <ARGUMENT NAME="otype" VALUE="TreetestObjects" />
</TREEMANAGER>
<IF COND="IsError.Variables.errno=false">
  <THEN>
    <LOOP LIST="Variables.treename.nid"/>
      <BR/><CSVAR NAME="Variables.treename.nid"/>
    </LOOP>
  </THEN>
  <ELSE>
    <BR/> No nodes.<BR/>
  </ELSE>
</IF>
```

See Also

[findnode](#)

getnode

Retrieves a node from a table. The object data may be joined to the result.

Syntax

```
<TREEMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="getnode"/>
  <ARGUMENT NAME="treename" VALUE="name"/>
  <ARGUMENT NAME="node" VALUE="nodeid"/>
  [<ARGUMENT NAME="join" VALUE="true or false"/>]
  [<ARGUMENT NAME="depth" VALUE="depth setting"/>]
</TREEMANAGER>
```

Parameters

ftcmd (required)

Value must be set to `getnode`.

treename (required)

Name of the tree.

node (required)

ID of the node.

join (optional)

`true` or `false` to join the object table. Note that the object table holds the cache information.

depth (optional)

Defaults to 1. Limits the depth of recursion when scanning a tree. Illegal values are converted to 1. Specify -1 to remove depth restrictions on scan. Note below the impact of a deep recursive search. The depth parameter **must** be set to 1 if join is `true`.

Object tables that are joined to are required to be registered as type `obj` in the `SystemInfo` table in order to support the caching of resultsets that are created.

The name of the LIST constructed is stored in the value of the variables, `treename` (that is, `Variables.treename`).

errno

The possible values of `errno` include:

Value	Description
-3	No access privileges to table.
-3	No access privileges to table.
-103	No table.
-108	Node not input.
-110	Invalid tree name.

Example

```
<SETVAR NAME="errno" VALUE="0"/>
<TREEMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="getnode"/>
  <ARGUMENT NAME="treename" VALUE="TreetestTree2"/>
  <ARGUMENT NAME="join" VALUE="false"/>
  <ARGUMENT NAME="node" VALUE="Variables.setme"/>
</TREEMANAGER>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!-- Handle error -->
  </THEN>
</IF>
```

See Also

[findnode](#)

getparent

Gets the parent record of a given node.

Syntax

```
<TREEMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="getparent"/>
  <ARGUMENT NAME="treename" VALUE="name"/>
  <ARGUMENT NAME="node" VALUE="nodeid"/>
</TREEMANAGER>
```

Parameters

ftcmd (required)
Value must be set to `getparent`.

treename (required)
Name of the tree

node (required)
ID of the node.

errno

The possible values of `errno` include:

Value	Description
-3	No access privileges to table.
-101	No rows. (If node is invalid.)
-103	No table.
-105	Database error.
-106	Missing or invalid parameters.

Value	Description
-110	Invalid tree name.
-112	The parent node does not exist or this node is a root node. Note that versions of Content Server earlier than 3.5 reacted to these errors by returning an errno of 0 (with an empty resultset 0 rather than an errno of -112).

Example

```

<TREEMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="getparent"/>
  <ARGUMENT NAME="treename" VALUE="TreetestTree"/>
  <ARGUMENT NAME="node" VALUE="Variables.parent"/>
</TREEMANAGER>
<IF COND="IsError.Variables.errno="false">
  <THEN>
    <IF COND="IsList.TreetestTree="true">
      <THEN>
        <TABLE CELLSPACING="2" CELLPADDING="2"
          BORDER="1" RULES="all">
          <TR>
            <TD><FONT COLOR="#000080">
              <B>nid</B></FONT>
            </TD>
            <TD><FONT COLOR="#000080">
              <B>nparentid</B></FONT>
            </TD>
            <TD><FONT COLOR="#000080">
              <B>otype</B></FONT>
            </TD>
            <TD><FONT COLOR="#000080">
              <B>oid</B></FONT>
            </TD>
            <TD><FONT COLOR="#000080">
              <B>nrnk</B></FONT>
            </TD>
            <TD><FONT COLOR="#000080">
              <B>ncode</B></FONT>
            </TD>
          </TR>
          <LOOP LIST="Variables.treename">
            <TR>
              <TD>
                <CSVAR NAME="Variables.treename.nid"/>
              </TD>
              <TD>
                <CSVAR NAME="Variables.treename.nparentid"/>
              </TD>
              <TD>
                <CSVAR NAME="Variables.treename.otype"/>

```

```

        </TD>
        <TD>
            <CSVAR NAME="Variables.treename.oid"/>
        </TD>
        <TD>
            <CSVAR NAME="Variables.treename.nrank"/>
        </TD>
        <TD>
            <CSVAR NAME="Variables.treename.ncode"/>
        </TD>
        <TD>
            <CSVAR NAME="Variables.treename.oversion"/>
        </TD>
    </TR>
</LOOP>
</TABLE>
</THEN>
</IF>
</THEN>
<ELSE>
    <BR/>failed<BR/>
</ELSE>
</IF>

```

listtrees

Retrieves a list of all the trees in the system.

Syntax

```

<TREEMANAGER>
    <ARGUMENT NAME="ftcmd" VALUE="listtrees"/>
    [<ARGUMENT NAME="treename" VALUE="name of tree"/>]
</TREEMANAGER>

```

Parameters

ftcmd (required)

Value must be set to listtrees.

treename (optional)

Name of the tree. If specified, this parameter causes tree filtering. For example, treename=Article% finds all the trees that begin with the word Article.

Description

The listtrees command retrieves a list of all the trees in the system. The list created is SystemTrees when invoked via the TREEMANAGER tag. The list field names are from the fields in the SystemInfo table.

errno

The possible values of errno include:

Value	Description
-3	No access privileges to table.
-101	No rows returned. This may be because no trees match the requested filter.

Example

```

<SETVAR NAME="errno" VALUE="0"/>
<TREEMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="listtrees"/>
</TREEMANAGER>

<IF COND="IsError.Variables.errno=false">
  <THEN>
    <TABLE CELLSPACING="2" CELLPADDING="2"
      BORDER="1" RULES="all">
      <TR>
        <TD><FONT COLOR="#000080">
          <B>Treename</B></FONT></TD>
      </TR>
      <LOOP LIST="SystemTrees">
        <TR>
          <TD>
            <CSVAR NAME="SystemTrees.tblname"/>
          </TD>
        </TR>
      </LOOP>
    </TABLE>
  </THEN>
  <ELSE>
    <BR/> No trees<BR/>
  </ELSE>
</IF>

```

See Also

[createtree](#)
[deletetree](#)

movechild

Moves a node and its children to a new parent that may be in the same or a different tree table.

Syntax

```
<TREEMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="movechild"/>
  <ARGUMENT NAME="treename" VALUE="name"/>
  <ARGUMENT NAME="node" VALUE="nodeid"/>
  <ARGUMENT NAME="newtreename" VALUE="treename"/>
  [<ARGUMENT NAME="nparentid" VALUE="node id"/>]
</TREEMANAGER>
```

Parameters

ftcmd (required)

Value must be set to movechild.

treename (required)

Name of the source tree.

node (required)

ID of the node to move.

newtreename (required)

Name of the new (destination) tree.

nparentid (optional)

ID of the new parent of the node (omit or NULL to move to root).

errno

The possible values of errno include:

Value	Description
-3	No access privileges to table.
-103	No table.
-105	Database error.
-106	Missing or invalid parameters.
-110	Invalid tree name.
-112	The parent node does not exist or this node is a root node. Note that versions of Content Server earlier than 3.5 reacted to these errors by returning an errno of 0 (with an empty resultset 0 rather than an errno of -112).

Example

```
<SETVAR NAME="errno" VALUE="0"/>
<TREEMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="movechild" />
  <ARGUMENT NAME="treename" VALUE="TreetestTree" />
  <ARGUMENT NAME="newtreename" VALUE="TreetestTree" />
  <ARGUMENT NAME="node" VALUE="Variables.moveme" />
  <ARGUMENT NAME="nparentid" VALUE="Variables.emptynode" />
</TREEMANAGER>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>
```

See Also

[copychild](#)

nodepath

Traverses the tree beginning at the parent of the current node, returning a list of all ancestor nodes within this same tree.

Syntax

```
<TREEMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="nodepath"/>
  <ARGUMENT NAME="treename" VALUE="name"/>
  <ARGUMENT NAME="node" VALUE="nodeid"/>
</TREEMANAGER>
```

Parameters

ftcmd (required)
Value must be set to `nodepath`.

treename (required)
Name of the tree.

node (required)
Node ID of the node.

The list represents the path of the node backwards to its root node (that is, the first record added is the immediate parent of the node, the next row is that parent's parent, and so forth). It stops at a root node in the tree.

The node is not included in the returned data, only the parent's. An empty table/list can be returned if node is a top-level node; that is, if its parent is `0`. The name of the list that is constructed is stored in the value of the `Variables.treename`.

Description

The `nodepath` command traverses the tree beginning at the parent of the current node, returning a list of all ancestor nodes within this same tree.

Note

This command does not span trees. In other words, the path to a node is always within the tree. If a node is pointed to by a node from another tree, it is not in the node path.

errno

The possible values of `errno` include:

Value	Description
-3	No access privileges to table.
-103	No table.
-105	Database error.
-110	Invalid tree name.

Example

```
<SETVAR NAME="errno" VALUE="0"/>
<TREEMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="nodepath"/>
  <ARGUMENT NAME="treename" VALUE="TreetestTree"/>
  <ARGUMENT NAME="node" VALUE="Variables.traceme"/>
</TREEMANAGER>

<IF COND="IsError.Variables.errno=false">
  <THEN>
    <TABLE cellspacing="2" cellpadding="2" border="1"
rules="all">
      <TR>
        <TD><FONT COLOR="#000080">
          <B>parent id</B></FONT>
        </TD>
      </TR>
      <LOOP LIST="Variables.treename">
        <TR>
          <TD>
            <CSVAR NAME="Variables.treename.nid"/>
          </TD>
        </TR>
      </LOOP>
    </TABLE>
  </THEN>
  <ELSE>
    <BR/> No nodes<BR/>
  </ELSE>
</IF>
```

setobject

Sets an object on a node.

Syntax

```
<TREEMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="setobject"/>
  <ARGUMENT NAME="treename" VALUE="name"/>
  <ARGUMENT NAME="node" VALUE="nodeid"/>
  [<ARGUMENT NAME="otype" VALUE="object table"/>]
  [<ARGUMENT NAME="oid" VALUE="object"/>]
  [<ARGUMENT NAME="validate" VALUE="true or false"/>]
  [<ARGUMENT NAME="nrank" VALUE="rank value"/>]
</TREEMANAGER>
```

Parameters

ftcmd (required)

Value must be set to `setobject`.

treename (required)

Name of the tree.

node (required)

ID of the node.

otype (optional)

Name of the object table.

oid (optional)

Identifies the object in an object table.

ncode (optional)

16-character string available for use by an application.

validate (optional)

If `true`, the object is validated before being set (see also `validatenode`).

nrank (optional)

Value of rank to assign this node.

Description

The `setobject` command sets an object on a node.

The `otype`, `oid`, and `ncode` parameters may be `NULL` if `validate` is `false`. If `validate` is `true`, the existence of the table named with the `otype` parameter, and the existence of a record identified by the `oid` parameter, is verified.

errno

The possible values of `errno` include:

Value	Description
-3	No access privileges to table.
-103	No table.
-105	Database error.
-106	Missing or invalid parameters.
-110	Invalid tree name.
-114	Missing class type (If <code>validate</code> is <code>true</code>).

Example

```
<SETVAR NAME="errno" VALUE="0"/>
<TREEMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="setobject"/>
  <ARGUMENT NAME="treename" VALUE="TreetestTree2"/>
  <ARGUMENT NAME="node" VALUE="Variables.setme"/>
  <ARGUMENT NAME="otype" VALUE="TreetestObjects"/>
  <ARGUMENT NAME="oid" VALUE="345345764"/>
  <ARGUMENT NAME="oversion" VALUE="1"/>
  <ARGUMENT NAME="validatenode" VALUE="true"/>
</TREEMANAGER>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>
```

validatenode

Verifies that the given node is valid.

Syntax

```
<TREEMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="validatenode"/>
  <ARGUMENT NAME="treename" VALUE="name"/>
  <ARGUMENT NAME="node" VALUE="nodeid"/>
</TREEMANAGER>
```

Parameters

`ftcmd` (required)

Value must be set to `validatenode`.

`treename` (required)

Name of the tree.

node (required)

ID of the node. Returns success or failure in the result code.

Description

The `validatenode` command verifies that the given node is valid. There must exist an object table with the `otype` of the node and there must be an object in that table with the `oid` of the node.

errno

The possible values of `errno` include:

Value	Description
-3	No access privileges to table.
-110	Invalid tree name.
-114	Missing class type (If <code>validate</code> is true).
-115	Unknown object class (if <code>validate</code> is true).

Example

```
<SETVAR NAME="errno" VALUE="0"/>
<TREEMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="validatenode"/>
  <ARGUMENT NAME="treename" VALUE="TreetestTree2"/>
  <ARGUMENT NAME="node" VALUE="Variables.setme"/>
</TREEMANAGER>
<IF COND="IsError.Variables.errno=false">
  <THEN>
    <BR/>success<BR/>
  </THEN>
  <ELSE>
    <BR/>failed<BR/>
  </ELSE>
</IF>
```

verifypath

Verifies that the given path of nodes matches the node's actual path.

Syntax

```
<TREEMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="verifypath"/>
  <ARGUMENT NAME="treename" VALUE="name"/>
  <ARGUMENT NAME="npath" VALUE="path"/>
</TREEMANAGER>
```

Parameters

ftcmd (required)
Value must be set to `verifypath`.

treename (required)
Name of the tree.

npath (required)
A string representing the path to the node.

Description

The `verifypath` command verifies that the given path of nodes matches its actual path. `verifypath` returns success or failure in the result code.

errno

The possible values of `errno` include:

Value	Description
-3	No access privileges to table.
-109	Tree path not valid.
-110	Invalid tree name.

Example

```
<SETVAR NAME="errno" VALUE="0"/>
<TREEMANAGER>
  <ARGUMENT NAME="ftcmd" VALUE="verifypath"/>
  <ARGUMENT NAME="treename" VALUE="TreetestTree"/>
  <ARGUMENT NAME="npath"
    VALUE="950582562373:950582562375:950582562377"/>
</TREEMANAGER>
<IF COND="IsError.Variables.errno=false">
  <THEN>
    <BR/>worked
  </THEN>
  <ELSE>
    <BR/>failed
  </ELSE>
</IF>
```



```
</ELSE>
</IF>
```

UNLOCKRECORD

Unlocks a locked row.

Syntax

```
<UNLOCKRECORD TABLE="TABLE_NAME"
  ASSET="PRIMARY_KEY"/>
```

Parameters

TABLE (required)

Name of the tracked table.

ASSET (required)

Value of the primary key for the row to unlock.

Description

The UNLOCKRECORD tag unlocks a locked row. Only the user who has the row locked or a user who has `rtadmin` privileges against the table may unlock the row. UNLOCKRECORD does not restore the row to its most recent revision.

errno

The possible values of `errno` include:

Value	Description
-1000	Command failure. Check the <code>errdetail</code> variable for the specific error.

Example

This example assumes all UNLOCKRECORD parameters are exposed on an HTML form. It first checks whether the parameters exist, clears any values from `errno`, and then unlocks the row.

```
<!-- Make sure required parameters exist -->
<IF COND="IsVariable.tablename=false">
  <THEN>
    <SETVAR NAME="tablename" VALUE=""/>
  </THEN>
</IF>
<IF COND="IsVariable.asset=false">
  <THEN>
    <SETVAR NAME="asset" VALUE=""/>
  </THEN>
</IF>
<!-- clear out errno -->
<SETVAR NAME="errno" VALUE="0"/>
<!-- do the unlock -->
```

```

<UNLOCKRECORD TABLE="Variables.tablename"
  ASSET="Variables.asset"/>
<!-- See if unlock succeeded -->
<IF COND="Variables.errno=0">
  <THEN>
    <!-- if succeeded, brag about it -->
    Unlock succeeded for
    <CSVAR NAME="Variables.asset"/>
    <BR/>
  </THEN>
<ELSE>
  <!-- otherwise, call an error reporting element -->
  Unlock failed.<BR/>
</ELSE>
</IF>

```

UNTRACKTABLE

Stops tracking a table.

Syntax

```
<UNTRACKTABLE TABLE="TABLE_NAME"/>
```

Parameters

TABLE (required)

Name of table to stop tracking

Description

The UNTRACKTABLE tag stops tracking a table. Revision-tracking operations are then no longer available on the table.

errno

The possible values of `errno` include:

Value	Description
-1000	Command failure. Check the <code>errdetail</code> variable for the specific error.

Example

This example assumes all UNTRACKTABLE parameters are exposed on an HTML form. It first checks whether the parameters exist, clears any values from `errno`, then removes revision tracking from the table.

```

<!-- Make sure required parameters exist -->
<IF COND="IsVariable.tablename=false">
  <THEN>
    <SETVAR NAME="tablename" VALUE=""/>
  </THEN>

```

```

</IF>
<!-- clear out errno -->
<SETVAR NAME="errno" VALUE="0"/>
<!-- do the untrack -->
<UNTRACKTABLE TABLE="Variables.tablename"/>
<!-- See if untrack succeeded -->
<IF COND="Variables.errno=0">
  <THEN>
    <!-- if succeeded, note it-->
    UntrackTable succeeded for
      <CSVAR NAME="Variables.tablename">
        <BR/>
  </THEN>
  <ELSE>
    <!-- otherwise, call an error reporting element -->
    UnTrackTable failed.<BR/>
  </ELSE>
</IF>

```

USERISMEMBER

Checks if the current user is a member of one or more Access Control Lists (ACLs).

Syntax

```
<USERISMEMBER GROUP="ACL_NAME"/>
```

Parameters

GROUP (required)

Comma-separated list of ACL names.

Description

The USERISMEMBER tag checks if the current user is a member of one or more Access Control Lists (ACLs). `Variables.errno` is set to 1 (one) if the user is a member of any of the listed ACLs.

errno

The possible values of `errno` include:

Value	Description
1	User is member of one or more groups.
0	User is not a member of any group.

Example

This example checks whether Editors is one of the items in the current user ACL.

```
<SETVAR NAME="errno" VALUE="0"/>
<USERISMEMBER GROUP="Editors"/>
<IF COND="Variables.errno=1">
  <THEN>
    User is member of Editors.
  </THEN>
</IF>
```

USER.ADD (Deprecated)

As of Content Server version 4.0, replaced by [DIR.CREATE](#).

Syntax

```
<USER.ADD USERNAME="username"
  PASSWORD="user's password"
  ACL="user's acl"/>
```

Parameters

USERNAME (required)
Name of the user to be added.

PASSWORD (required)
Password for the user.

ACL (required)
ACL assigned to the new user.

Description

Adds a Content Server user to the system.

errno

The possible values of errno include:

Value	Description
-10004	A required parameter is missing.

Example

The following example adds a user to the Content Server system.

```
<USER.ADD USERNAME="Whitney"
  PASSWORD="rosebud"
  ACL="xceleeditor"/>
```

See Also

[USER.MODIFY \(Deprecated\)](#)

USER.CHANGEPASSWORD (Deprecated)

As of Content Server version 4.0, replaced by [DIR.REPLACEATTRS](#).

Syntax

```
<USER.CHANGEPASSWORD PASSWORD1=new password
                        PASSWORD2=new password/>
```

Parameters

PASSWORD1 (required)

The user's new password.

PASSWORD2 (required)

The new password confirmation.

Description

The USER.CHANGEPASSWORD tag changes the password of an existing user. This tag confirms that the password entries match, and then changes the password. When a user submits a form that changes a password, this tag verifies that change.

errno

The possible values of errno include:

Value	Description
-105	Database error
-10004	A required parameter is missing.

Example

The following example confirms the changes to the password of the current user.

```
<USER.CHANGEPASSWORD PASSWORD1="newpw"
                        PASSWORD2="newpwconfirm"/>
```

See Also

[USER.ADD \(Deprecated\)](#)

[USER.MODIFY \(Deprecated\)](#)

USER.DELETE (Deprecated)

As of Content Server version 4.0, replaced by [DIR.DELETE](#).

Syntax

```
<USER.DELETE USERNAME="username"/>
```

Parameters

USERNAME (required)
Name of the user to be deleted.

Description

The USER.DELETE tag deletes a user from the Content Server system.

errno

The possible values of errno include:

Value	Description
-10004	A required parameter is missing.

Example

The following example deletes a user named Whitney from the Content Server system.

```
<USER.DELETE USERNAME="Whitney"/>
```

See Also

[USER.ADD \(Deprecated\)](#)

[USER.MODIFY \(Deprecated\)](#)

USER.LOGIN

Log in as a particular user.

Syntax

```
<USER.LOGIN USERNAME=username  
                PASSWORD=user's password/>
```

Parameters

USERNAME (required)
User's login name.

PASSWORD (required)
User's password.

Description

The `USER.LOGIN` tag allows you to log in as a particular Content Server user.

errno

The possible values of `errno` are:

Value	Description
-10004	A required parameter is missing.

Example

The following example logs in a user named Whitney.

```
<USER.LOGIN USERNAME="Whitney" PASSWORD="rosebud"/>
```

See Also

[USER.LOGOUT](#)

[USER.SU](#)

USER.LOGOUT

Logs out the current user.

Syntax

```
<USER.LOGOUT/>
```

Parameters

None.

Description

The `USER.LOGOUT` tag logs out the current Content Server user.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.

Example

The following example logs a user out of the Content Server system.

```
<USER.LOGOUT/>
```

See Also

[USER.LOGIN](#)
[USER.SU](#)

USER.MODIFY (Deprecated)

As of Content Server version 4.0, replaced by [DIR.REPLACEATTRS](#).

Syntax

```
<USER.MODIFY USERNAME=username
                [PASSWORD=user password]
                [ACL=user acl]/>
```

Parameters

USERNAME (required)

The username of the user to be modified.

PASSWORD (optional)

Password of the user to be modified.

ACL (optional)

ACL of the user to be modified.

Description

The modify tag is used to modify a username, password, and ACL values.

errno

The possible values of errno include:

Value	Description
-105	Database error
-10004	A required parameter is missing
-10010	Invalid attribute value

Example

The following example modifies the ACL for the user.

```
<USER.MODIFY NAME="Whitney"
              ACL="ContentEditor"/>
```

See Also

[USER.CHANGEPASSWORD \(Deprecated\)](#)

USER.SU

Log in as a particular user.

Syntax

```
<USER.SU USERNAME="user name"
        PASSWORD="user's password"/>
```

Parameters

USERNAME (required)
Name of the user.

PASSWORD (required)
The user's password.

Description

Allows you to log in as a particular so you can make changes. This tag is useful when combined with the USER.MODIFY tag to make administrative user profile changes.

errno

The possible values of errno include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

Example

The following example logs in a user as Percy, changes the password, and logs out and back to the original user:

```
<USER.SU USERNAME="Percy" PASSWORD="rosebud">
  <USER.MODIFY USERNAME="Percy"
                PASSWORD="other"/>
</USER.SU/>
```

See Also

[USER.LOGIN](#)
[USER.LOGOUT](#)

WRITEFILE

Writes a string to a file.

Syntax

```
<WRITEFILE STR="STRING_VALUE"  
          FILE="FILE_NAME"/>
```

Parameters

STR (required)

String to write to a file

FILE (required)

Name of the file to write.

Description

The WRITEFILE tag writes a string to a file. The file path you specify must be legal; subfolders are created as needed. The syntax must be specific to your operating system.

errno

The possible values of errno include:

Value	Description
-301	File was not written.

Example

The following example writes the value of `Variables.junk` to the Windows NT file, `foo.bar`, in the `temp` directory.

You should create a property in the Content Server properties file that contains system-dependent file path syntax and append the file name to the property to construct the full path.

For example, you if you specify the folder location property as:

```
app.OutputFolder=c:\temp\
```

The example code to write the value of `Variables.junk` to the folder `foo.bar` is:

```
<SETVAR NAME="folder" VALUE="CS.Property.app.OutputFolder"/>  
<WRITEFILE STR="Variables.junk" FILE="Variables.folderfoo.bar" />
```

See Also

[DELETEFILE](#)

XMLFILTER

Invokes XML processing for content filtering and tag remapping.

Syntax

```
<XMLFILTER [FILTERFILE="FILTERFILE_NAME"
           [FILTERSTR="FILTER_STRING"
           [OUTFILE="OUTFILE_NAME"
           [FILE="FILE_NAME"
           [STR="STRING_NAME"
           VERIFY="VERIFY_VAL "
           [RUN="RUN_VAL "
           [RUNOUT="RUNOUT_VAL "]/>
```

Parameters

FILTERFILE (optional)

File defining the tag syntax. For example, a tag <Arial></Arial> whose definition is <"Font face=Arial">.

FILTERSTR (optional)

Name of the string whose content defines the syntax in **FILTERFILE**.

OUTFILE (optional)

Name of the file to contain results of filtered XML tags found in **FILE_NAME** or **STRING_NAME**.

FILE (optional)

Source file containing XML-tagged content. If the file is empty (not null), Content Server creates a default file.

STR (optional)

Source string containing XML-tagged content. If the string is empty (not null), Content Server creates a default string.

VERIFY (required)

Indicates **FILE_NAME** is verified against the filter (syntax checked). Legal values are true or false.

RUN (optional)

Indicates whether to process the content of **OUTFILE_NAME** for Content Server tags. Legal values are true or false.

If value is true and results are not saved to file, the result from the filter is expected to be well-formed XML. The required XML header/footer to specify the FutureTense DTD is prepended/appended before processing.

Otherwise, the file is saved to disk. If the users specifies XML as the file extension, that user must know how to mark an element properly.

RUNOUT (optional)

If **RUN** is set to true, this variable's value is set to the evaluated stream instead of writing to the file specified by **OUTFILE**.

errno

The possible values of **errno** include:

Value	Description
-401	No input.
-402	No filter definitions.
-403	Parsing failed.
-404	Write file failed.
-405	Bad file.
-406	Bad parameters
-407	Document failed.
-408	Syntax error occurred during parse.

Example

This example constructs a temporary name for filtered output. Because `RUN=true` in the example, the results of the filtered file are then "evaluated" as if they were an element. The content of the resulting file is inserted and then the file is deleted. `NewsDir` is a list constructed by querying the `SystemInfo` table for information about the `News` table.

```
<SETVAR NAME="tblname" VALUE="News" />
<SELECTTO FROM="SystemInfo" WHAT="defdir"
    LIST="NewsDir" WHERE="tblname" />
<SETVAR NAME="XMLOutFile"
    VALUE="CS.Property.cs.xmlfolder/CS.UniqueID.xml"/>
<SETVAR NAME="errno" VALUE="0" />
<XMLFILTER FILTERFILE="NewsDir.defdir/NewsStoryFilter.xml"
    OUTFILE="Variables.XMLOutFile"
    FILE="NewsDir.defdir/Stories.urlBody"
    RUN="true"/>
<IF COND="IsError.Variables.errno">
  <THEN>
    <!--Handle error.-->
  </THEN>
  <ELSE>
    <INSERT FILE="Variables.XMLOutFile"/>
    <DELETEFILE FILE="Variables.XMLOutFile"/>
  </ELSE>
</IF>
```

See Also

[INSERT](#)
[DELETEFILE](#)

Variables and Built-in Functions

The following section describes each Content Server variable and built-in function, beginning with [CS.AMPM](#).

CS.AMPM

Gets the value of AM or PM based on the current built-in Content Server time.

Syntax

CS.AMPM

Parameters

None.

Example

If the current time property is after 12 noon, this built-in displays "PM".

```
<CSVAR NAME="CS.AMPM"/>
```

See Also

[CS.Date](#)
[CS.Hour](#)
[CS.Min](#)
[CS.Mon](#)
[CS.CacheFile](#)
[CS.DDate](#)
[CS.Year](#)
[SETDATE](#)
[CS.Hour24](#)
[CS.Day](#)

CS.Browser

Gets the type of browser used to execute a request.

Syntax

CS.Browser

Parameters

None.

Description

The CS.Browser built-in gets the type of browser used to execute a request. This built-in returns the following values:

- Netscape for a Netscape browser.

- MSIE for a Microsoft Internet Explorer browser.
- Unknown Browser when the browser is unknown.

Example

```
<CSVAR NAME="CS.Browser"/>
```

CS.CacheFile

Gets the file name associated with a cached pagename and its parameters.

Syntax

```
CS.CacheFile [PAGENAME], [PARAM1=VAL1,PARAM2=VAL2...]
```

Parameters

PAGENAME (optional)

The full pagename including parameters.

PARAM1=VAL1,PARAM2=VAL2,... (optional)

Name/Value pairs. Use an "=" to associate a name with a certain value. Use "&" to separate multiple name/value pairs.

Description

The CS.CacheFile built-in gets the file name associated with a cached pagename and its parameters.

When caching is enabled, Content Server derives a cached page file name based on the incoming SiteCatalog entry for pagename and other relevant incoming environment information that a user must define.

You can specify such information on a URL. For example:

```
http://localhost/????webapppath??? (xyz)/  
ContentServer?pagename=somepage?someparam=somevalue
```

or

```
http://localhost/????webapppath??? (xyz)  
ContentServer?pagename=somepage?  
someparam=somevalue&someparam2=somevalue2>  
&someparam2=somevalue2
```

The value *someval* is a parameter passed to the root element of *somepage*. If any of the variables listed are empty, ContentServer ignores them.

Example

The following examples display the location and file names of two pages that are cached on disk. The first example shows you how to specify the tag parameters when there are no page arguments defined. The second example shows how you define the tag parameters where there are page arguments. Notice that in example 2 the ampersand (&) character is with its decimal representation, '&'.

Example 1.

```
<CSVAR NAME="CS.CacheFileName.OpenMarket.Samples.NewPortal.XML.
```

```
mainheadlines/>
```

Example 2.

```
<CSVARNAME="CS.CacheFileName.OpenMarket.Samples.NewPortal.XML.NewsBrick1,category=s&#038;title=Sports"/>
```

CS.CatalogDir

Gets the value of the upload (defdir) folder associated with a table.

Syntax

```
CS.CatalogDir.CATALOGNAME
```

Parameters

CATALOGNAME (required)
Name of the Content Server table.

Example

This example displays the upload (defdir) folder associated with a table called movies.

```
<CSVAR NAME="CS.CatalogDir.movies"/>
```

See Also

[CS.CacheFile](#)

CS.Date

Gets the value for the date based on the current Content Server time property.

Syntax

```
CS.Date
```

Parameters

None.

Description

The CS.Date built-in gets the value for the date based on the current Content Server time property. The format is: Ddd Mmm dd hr:min:sec TMZ yyyy.

Example

```
<CSVAR NAME="CS.Date"/>
```

This built-in displays information such as the following:

```
Mon Nov 09 22:47:07 EST 1998.
```

See Also

[CSVAR](#)
[CS.Day](#)
[CS.DDate](#)
[CS.Hour](#)
[CS.Hour24](#)
[CS.Min](#)
[CS.Mon](#)
[CS.Year](#)

CS.DDate

Gets the value for the date (numeric) based on the current Content Server time property.

Syntax

`CS.Ddate`

Parameters

None.

Description

The `CS.DDate` tag gets the value for the day (numeric) based on the current Content Server time property. Possible values are 1 to 31.

Example

If the date is November 9, 1998, this built-in displays 9.

```
<CSVAR NAME="CS.DDate"/>
```

See Also

[CSVAR](#)
[CS.Date](#)
[CS.Day](#)
[CS.Hour](#)
[CS.Hour24](#)
[CS.Min](#)
[CS.Mon](#)
[CS.Year](#)

CS.Day

Gets the value for the day based on the current Content Server time property.

Syntax

CS.Day

Parameters

None.

Description

The CS.Day tag gets the value of day based on the current Content Server time property. The format is Ddd.

Example

If the date is November 9, 1998, this built-in displays Mon.

```
<CSVAR NAME="CS.Day"/>
```

See Also

[CSVAR](#)
[CS.DDate](#)
[CS.Date](#)
[CS.Hour](#)
[CS.Hour24](#)
[CS.Min](#)
[CS.Mon](#)
[CS.Year](#)

CS.Elementname

Returns the name of the executing element.

Syntax

CS.Elementname

Parameters

None.

CS.Empty

Gets an empty or zero length string.

Syntax

CS.Empty

Parameters

None.

Example

The following example sets a variable called Value to an empty string.

```
<SETVAR NAME="Value" VALUE="CS.Empty" />
```

CS.Header

Returns the value of an HTTP header.

Syntax

CS.Header

Parameters

None.

Description

The header name is specified as an argument followed by a dot. Some examples are as follows:

- CS.Header.Host
- CS.Header.Accept
- CS.Header.Cookie
- CS.Header.Accept-Language
- CS.Header.Connection
- CS.Header.Accept-Encoding
- CS.Header.User-Agent

CS.Hour

Gets the value for the hour (1-12) based on the current Content Server time property.

Syntax

CS.Hour

Parameters

None.

Description

The CS.Hour button gets the value for the hour based on the current Content Server time property. The range is 1-12.

Example

If the time is 1:00 p.m., this built-in displays 1.

```
<CSVAR NAME="CS.Hour"/>
```

See Also

[CSVAR](#)
[CS.CacheFile](#)
[CS.Date](#)
[CS.DDate](#)
[CS.Hour24](#)
[CS.Mon](#)
[CS.Min](#)
[CS.Year](#)

CS.Hour24

Gets the value for the hour based on the current Content Server time property.

Syntax

CS.Hour24

Parameters

None.

Description

The CS.Hour24 built-in gets the value for the hour based on the current Content Server time property. The range is 0-23.

Example

If the time is 11:00 p.m., this built-in displays 23.

```
<CSVAR NAME="CS.Hour24"/>
```

See Also

[CSVAR](#)
[CS.Date](#)
[CS.DDate](#)
[CS.Hour](#)
[CS.Min](#)
[CS.Mon](#)
[CS.Year](#)

CS.HTTPS

Returns the string “on” if the current required used HTTPS; if the user used HTTP, it returns the string “off.”

Syntax

CS.HTTPS

Parameters

None.

Example

```
<IF COND="CS.HTTPS=on">  
  <THEN>  
    ...  
  <THEN/>  
</IF/>
```

CS.IOError

Returns true or false depending on whether an I/O error has occurred writing back to the response.

Syntax

CS.IOError

Parameters

None.

Description

If the client (for example, the browser) cancels the request, the page request on Content Server may still be processing. This processing could continue for a long time. By checking this flag, a page may decide to terminate prematurely if it can no longer write back to the client response.

CS.IsExporting

Indicates whether Content Server is in the process of exporting a page.

Syntax

CS.IsExporting

Parameters

None.

Description

The CS.IsExporting tag indicates whether Content Server is in the process of exporting a page. Returns `true` if the page is exporting, and `false` if it is not. Use this property, for example, in an element to conditionalize behavior based on page export status.

Example

This example displays a message if the page containing the tag is being exported.

```
<IF COND="CS.IsExporting=true">
  <THEN>
    Exporting the page.
  </THEN>
</IF>
```

CS.Min

Gets the value for the minute based on the current Content Server time property.

Syntax

CS.Min

Parameters

None.

Description

The CS.Min tag gets the value for the minute based on the current Content Server time property. The range is 0-59.

Example

If the current time was 11:03 pm, this tag displays 3.

```
<CSVAR NAME="CS.Min"/>
```

See Also

[CS.Date](#)
[CS.Hour](#)
[CSVAR](#)

[CS.Mon](#)
[CS.DDate](#)
[CS.Year](#)
[CS.Hour24](#)
[CS.Day](#)

CS.Mon

Gets the value for the month based on the current Content Server time property.

Syntax

CS.Mon

Parameters

None.

Description

The CS.Mon built-in gets the value for the month based on the current Content Server time property. The range is Jan-Dec.

Example

If the current month is November, Nov is displayed.

```
<CSVAR NAME="CS.Mon"/>
```

See Also

[CS.Date](#)
[CS.Hour](#)
[CSVAR](#)
[CS.Mon](#)
[CS.DDate](#)
[CS.Year](#)
[CS.Hour24](#)
[CS.Day](#)

CS.PATH_INFO

Gets path information for the CGI variable PATH_INFO.

Syntax

CS.PATH_INFO

Parameters

None.

Description

This tag returns the path of the CGI variable. The CGI path information returned appears in the request URL following the servlet path and preceding the query string.

Example

```
<CSVAR NAME="CS.PATH_INFO"/>
```

See Also

[CS.QUERY_STRING](#)

CS.Property

Gets the value of a property found in the Content Server property file.

Syntax

CS.Property.*PROPNAME*

Parameter

PROPNAME (required)

Name of a Content Server property.

Example

This example displays the value of the property keyname located in the Content Server property file.

```
<SETVAR NAME="suffix" VALUE="keyname"/>
<CSVAR NAME="wrong variable assignment: Variables.wrong_variable"/>
<BR/>
```

```
<SETVAR NAME="wrong_variable2"
VALUE="CS.Property.cc.contentkey=keyname"/>
<CSVAR NAME="wrong variable assignment option 2:
Variables.wrong_variable2"/><BR/>
```

```
<SETVAR NAME="correct_variable1" VALUE="key
name=CS.Property.cc.contentkey"/>
```

```
<CSVAR NAME="correct variable assignment option one:
Variables.correct_variable1"/><BR/>
```

```
<SETVAR NAME="prefix" VALUE="key name="/>
<SETVAR NAME="correct_variable2"
VALUE="Variables.prefixCS.Property.cc.contentkey"/>
<CSVAR NAME="correct variable assignment option two:
Variables.correct_variable2"/><BR/>
```

The browser displays the following output:

wrong variable assignment option 1: Variables.wrong_variable

wrong variable assignment option 2:

correct variable assignment option one: key name=id

correct variable assignment option two: key name=id

CS.QUERY_STRING

Returns the value of the Query String CGI variable.

Syntax

CS.QUERYSTRING

Parameters

None.

Description

This tag returns the Query String CGI variable that is located in the request URL path.

Example

```
<CSVAR NAME="CS.QUERY_STRING"/>
```

If the server request is:

HTTP://localhost/servlet/ContentServer?pagename=page1

and the variable name is:

```
<CSVAR NAME="CS.Query%String"/>
```

The following output is generated:

pagename=page1

See Also

[CS.PATH_INFO](#)

CS.REMOTE_ADDR

Returns the client's IP address.

Syntax

CS.REMOTE_ADDR

Parameters

None.

Description

This tag returns the requesting client's IP address.

Example

```
<CSVAR NAME="CS.REMOTE_ADDR"/>
```

See Also

[CS.REMOTE_HOST](#)

CS.REMOTE_HOST

Returns the fully qualified name of the client.

Syntax

CS.REMOTE_HOST

Parameters

None.

Description

This tag returns the fully qualified name of the requesting client. If the name of the client cannot be determined, Content Server returns the IP address.

Example

```
<CSVAR NAME="CS.REMOTE_HOST"/>
```

See Also

[CS.REMOTE_ADDR](#)

CS.REQUEST_METHOD

Returns the name of an HTTP method.

Syntax

CS.REQUEST_METHOD

Parameters

None.

Description

This tag returns the name of the requesting HTTP method. Use this tag to determine if the request was made by a GET, POST, INT or other HTTP method.

Example

```
<CSVAR NAME="CS.REQUEST_METHOD"/>
```

CS.SERVER_NAME

Returns the host name of the requesting server.

Syntax

CS.SERVER_NAME

Parameters

None.

Description

The CS.SERVER_NAME tag retrieves the host name of the requesting server.

Example

```
<CSVAR NAME="CS.SERVER_NAME"/>
```

See Also

[CS.SERVER_PORT](#)

CS.SERVER_PORT

Returns the port number of the requesting server.

Syntax

CS.SERVER_PORT

Parameters

None.

Description

Returns the port number of the requesting server.

Example

```
<CSVAR NAME="CS.SERVER_PORT"/>
```

See Also

[CS.SERVER_NAME](#)

CS.SERVER_PROTOCOL

Returns protocol name and version.

Syntax

CS.SERVER_PROTOCOL

Parameters

None.

Description

This tag returns the name and version of the protocol a request uses. The information is returned in the following form:

Protocol/Major version . Minor Version

For example:

HTTP/1.1

Example

```
<CSVAR NAME="CS.SERVER_PROTOCOL"/>
```

See Also

[CS.SERVER_NAME](#)

CS.SQLDate

Gets the value for the date in SQL format, based on the current Content Server time property.

Syntax

CS.SQLDate

Parameters

None.

Description

The CS.SQLDate built-in gets the value of date based on the current Content Server time property.

Example

```
<CSVAR NAME="CS.SQLDate"/>
```

If the date were November 9, 2001, this built-in displays 2001-11-09 23:12:53.

See Also

[CS.Hour](#)
[CS.Min](#)
[CSVAR](#)
[CS.CacheFile](#)
[CS.DDate](#)
[CS.Year](#)

CS.SystemAssetsRoot

Gets the root directory where assets are stored.

Syntax

CS.SystemAssetsRoot

Parameters

None.

Example

This example displays the value of the assets' root directory.

```
<CSVAR NAME="CS.SystemAssetsRoot"/>
```

CS.UniqueID

Generates a string that is ensured to be unique within the Content Server cluster.

Syntax

CS.UniqueID

Parameters

None.

Example

```
<CSVAR NAME="CS.UniqueID"/>
```

CS.Year

Gets the value for the year.

Syntax

CS.Year

Parameters

None.

Example

If the current year is 2002, the following tag displays 2002:

```
<CSVAR NAME="CS.Year"/>
```

See Also

[CSVAR](#)
[CS.Date](#)
[CS.Day](#)
[CS.Hour](#)
[CS.Hour24](#)
[CS.DDate](#)
[CS.Min](#)
[CS.Mon](#)

Chapter 2

CS-Direct XML Tags

This chapter describes the functionality and XML syntax of the CS-Direct object methods. These methods are implemented as Content Server custom application tags, which invoke application business logic for applications built with Content Server.

Use this chapter as a reference for the object tags that you need to do the following:

- Retrieve instances of assets from the database, storing them as objects in memory, so that you can display their content to your site visitors by rendering it in online pages.
- Extract information about the design or structure of your online site so that you can display that information to your site visitors by rendering it in online pages.
- Configure workflow on your system.

This chapter contains the following sections:

- [Asset Tags](#)
- [CCUser Tags](#)
- [Date Format Tags](#)
- [Email Tags](#)
- [Email Manager Tags](#)
- [InSite Editor Tag](#)
- [Participants Tags](#)
- [Property Tag](#)
- [Render Tags](#)
- [Role List Tags](#)
- [Site List Tags](#)
- [Site Plan Tags](#)
- [Type List Tags](#)
- [User List Tags](#)

- [User Manager Tags](#)
- [Variables](#)
- [Workflowable Object Tags](#)
- [Workflow Action Tags](#)
- [Workflow Action List Tags](#)
- [Workflow Asset Tags](#)
- [Workflow Assignment Tags](#)
- [Workflow Engine Tags](#)
- [Workflow Function Tags](#)
- [Workflow Group Tags](#)
- [Workflow Priv Tags](#)
- [Workflow Privs Tags](#)
- [Workflow Process Tags](#)
- [Workflow State Tags](#)
- [Workflow Step Tags](#)
- [Workflow Timed Action Tags](#)
- [Workflow Timed Action List Tags](#)

For descriptions of the Content Server XML tags—tags that you use to create cookies, counters, lists, variables, and conditions, to access databases or files, to set up events, search engines, and revision tracking, and to execute calls to the Content Server modules—see [Chapter 1, “Content Server XML Tags.”](#)

Asset Tags

Assets are the core of CS-Direct. Assets are objects that can be created, edited, inspected, deleted, duplicated, placed into workflow, searched for, checked out and in with revision tracking, previewed, and published. You use the ASSET tags to extract from the CS-Direct database the information about your assets that you want to display on your online site.

When you use an [ASSET.LOAD](#) tag to retrieve an asset from the database, CS-Direct loads an instance of the asset into memory as an object. You then have access to that instance of the asset with the other ASSET tags until either the session is flushed (the root element exits) or the name that is used for the asset object is overwritten.

Assets can have parent-child relationships that are set up either through **named associations** or through unnamed relationships. Named associations are defined, asset-type-specific relationships represented as fields in the asset forms. Unnamed relationships are established, for example, when you select assets from the generic Select list boxes on the Page form.

CS-Direct stores information about assets in the following database tables:

- Primary database table that holds assets of its type. For example, each page asset has a row in the Page table and each article asset has a row in the Article table. These tables store information such as the asset's name, its object ID, who created it, which template it uses, and so on. You use the ASSET tags to extract information from these tables.
- AssetRelationTree table, if the asset has named associations or unnamed relationships with other assets. You use the [ASSET.CHILDREN](#) tag to extract information from this table.
- AssetPublication table, which specifies which management **sites** (publications) give you access to the asset through CS-Direct's asset entry and edit forms. If the asset is shared between more than one site, there is a row entry for each **pubid**. A pubid is a unique value that identifies a site.
- SitePlanTree table, if the asset is a page asset. For information about this table, see [Role List Tags](#).

To create types of assets, you use the AssetMaker forms in the CSEE Administrator software. For more information about asset type development, see the *CSEE Developer's Guide*.

The following section describes each asset tag, beginning with [ASSET.ADDCHILD](#).

ASSET.ADDCHILD

Designates an asset to be added as a child asset in the AssetRelationTree table when the **ASSET.SAVE** tag is called.

Syntax

```
<ASSET.ADDCHILD
  NAME="asset object name"
  TYPE="asset type"
  CHILDDID="ID"
  [CODE="association name"]
  [RANK="integer"] />
```

Parameters

NAME (required)

Name of the asset instance object.

TYPE (required)

Asset type of child asset. Stored in OTYPE field.

CHILDDID (required)

ID of child asset to add. Stored in OID field.

CODE (optional)

Association name. Stored in NCODE field You can specify named or unnamed associations.

RANK (optional)

Integer indicating rank of child asset. Stored in NRANK field.

errno

The possible values of errno include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-10007	Non-numeric rank.
-10014	Asset cannot be its own child.
-10017	Non-numeric rank.

ASSET.CANEDIT

Checks whether a publish operation, which prevents the specified asset from being edited, is in progress.

Syntax

```
<ASSET.CANEDIT
  NAME="asset name" />
```

Parameters

NAME (required)
Name of previously loaded asset.

Description

The ASSET.CANEDIT tag checks the PubSession table for any running (status='R') publish operations. If the specified asset is approved for publishing in the target for any of those running publish sessions, the asset cannot be edited.

errno

The possible values of errno include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-12034	Publish in progress.

Example

This example checks whether an asset can be edited and displays an error if the asset cannot be edited.

```
<ASSET.LOAD NAME=myArticle TYPE="Article" OBJECTID=123456789012/>
<ASSET.CANEDIT NAME=myArticle/>
<if COND="IsError.Variables.errno=true">
<then>
  <callelement NAME="OpenMarket/Xcelerate/UIFramework/Util/
ShowMessage">
    <argument NAME="elem" VALUE=" EditNotAllowed"/>
    <argument NAME="severity" VALUE="info"/>
    <argument NAME="AssetType" VALUE=" Article "/>
    <argument NAME="id" VALUE="123456789012"/>
  </callelement>
</then>
</if>
```

ASSET.CHILDGATHER

Gathers information from variables about which child relationships in the AssetRelationTree table should be added, deleted, or updated for a loaded asset.

Syntax

```
<ASSET.CHILDGATHER
  NAME="asset instance object name"
  PREFIX="prefix"
  CODELIST="list name" />
```

Parameters

NAME (required)

Name of the asset instance object.

PREFIX (required)

Prefix of the variable name; the first part of the variable name.

CODELIST (required)

List of asset association names which serves both as the second part of the variables name and the value of the ncode field that are written to the AssetRelationTree.

Description

The ASSET.CHILDGATHER tag uses variables whose name is created in either of the following ways:

- Concatenating the prefix argument, a colon (:), and a value from the codelist argument, followed by :id to define the child asset's ID.
- Concatenating the prefix argument, a colon (:), a value from the codelist argument, followed by :asset type to define the child asset's asset type. The AssetRelationTree is updated when the [ASSET.SAVE](#) tag is called.

errno

The possible values of errno include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

ASSET.CHILDREN

Queries the AssetRelationTree table and builds a list of child assets for the specified parent asset.

Syntax

```
<ASSET.CHILDREN
  NAME="assetName"
  LIST="listName"
  [CODE="NameOfAssociation"]
  [OBJECTTYPE="typeOfObject"]
  [OBJECTID="objectID"]
  [ORDER="nrank"]/>
```

Parameters

NAME (required)

Input. Name assigned to the parent asset whose children you want to list. This parent asset must be loaded and assigned a name before you can pass its name to this tag. (See [ASSET.LOAD](#).)

LIST (required)

Input. The name to assign to the list that stores the results. The columns in the list match the columns in the AssetRelationTree table, and, if you use the OBJECTTYPE parameter, the asset table. If a list with this name already exists, it is replaced with this new list.

CODE (optional)

Input. Restricts the list to include only the child assets that have the relationship (association) specified by this parameter. You can restrict the list by named associations or by unnamed relationships. This value comes from the ncode field of the AssetRelationTree.

For example, if you include OBJECTTYPE="Image" and CODE="MainImage" for an article asset, CS-Direct lists only the image asset that is related to that article asset by the Main Image named association. Without the CODE entry, CS-Direct lists all the images associated with the article.

To list only the child assets that are associated with the parent asset by unnamed relationships, use a hyphen (-). For example: CODE= "-"

If you do not specify OBJECTTYPE or CODE, the list includes all children with named associations or unnamed relationships to the parent asset.

OBJECTTYPE (optional)

Input. Restricts the list of children by asset type. If you supply an OBJECTTYPE, the list of children is a join of the AssetRelationTree and the asset table for the type specified.

To request a specific child asset, combine this parameter with OBJECTID. Note that if you include an OBJECTID, OBJECTTYPE is not optional.

OBJECTID (optional)

Input. Restricts the list to include only the child asset that you specify. You must provide both an OBJECTID and an OBJECTTYPE to request a specific child.

ORDER (optional)

Input. The fields to sort the list by and whether the sort on those fields is ascending (asc) or descending (desc). By default, the sort is ascending. If you specify more than one field, separate the field names with a comma.

For example, if you specify `ORDER="nrank"`, the list is sorted by rank starting at number 1. If you want the list sorted by descending rank, use `ORDER="nrank desc"`.

Description

This tag queries the `AssetRelationTree` table for a list of the children of the asset that you specify, listing each child with a value for all of the fields from that table (`nid`, `nparentid`, `nrank`, `otype`, `oid`, and `ncode`). You must load the parent asset ([ASSET.LOAD](#)) before you can invoke this tag to query for and list its children. The list is a standard Content Server list.

Use this tag to retrieve the assets referred to by a collection asset, the image assets associated with an article asset, and so on. You can then loop through the list, or reference the data returned in the list to display the relevant information from those assets.

You can restrict the list of children by association name (`CODE`), object type, object ID, and rank (`ORDER`).

If you use the `OBJECTTYPE` parameter, the resulting list of children is a join of the `AssetRelationTree` and the asset table for the type specified and contains data from both tables. In such a case, you do not need to use an [ASSET.LOAD](#) for a child asset of that type if that asset type stores all of its asset data in the primary asset table—you can get that information from the list.

errno

The possible values of `errno` include:

Value	Description
-111	The asset has no children.
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-10006	The object ID is not valid.
-10007	The version of the object is not valid.

Example

This code retrieves the query asset that has been associated with a parent asset through the `LatestNews` association and writes it to a list named `LatestNewQuery`:

```
<ASSET.CHILDREN NAME="HomePage" LIST="LatestNewQuery"
OBJECTTYPE="Query" CODE="LatestNews"/>
```

This code retrieves a collection of articles, determines the members of the collection, and then displays the `Description` field of each article:

```
<ASSET.CHILDREN NAME="MainArticle" CODE="ImageGallery" LIST="IG"/>
```

```

<ASSET.LOAD NAME="IG" OBJECTTYPE="Collection" OBJECTID="IG.oid"/>
<ASSET.CHILDREN NAME="IG" CODE="-" ORDER= "nrank"
OBJECTTYPE="Image" LIST="IGL"/>
<OL>
  <LOOP LIST="IGL">
    <LI/> <CSVAR NAME="IGL.description"/>
  </LOOP>
</OL>

```

See Also

[ASSET.GETTEMPLATEROOTELEMENT](#)

Chapter 1, “Content Server XML Tags,” for the standard Content Server list tags.

ASSET.CHILDTYPES

Builds a list of asset types that are children of this asset.

Syntax

```

<ASSET.CHILDTYPES
  NAME="asset name"
  LIST="list name" />

```

Parameters

NAME (required)

Name of previously loaded asset.

LIST (required)

Name of asset type list. The list includes the asset type in the OTYPE field.

Description

The `ASSET.CHILDTYPE` tag helps to improve performance. For example, you can loop through the asset types and use an [ASSET.LOADALL](#) tag for each asset type to load all the children.

errno

The possible values of errno include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

Example

This example loads all child assets of myArticle using names derived from the asset type of the child.

```

<ASSET.LOAD NAME=myArticle OBJECTID=123456789012/>
<ASSET.CHILDTYPES NAME=myArticle LIST=assettypes/>
<LOOP LIST= assettypes>
    <ASSET.CHILDREN NAME=myArticle OBJECTTYPE=assettypes.otype
LIST=childList/>
    <ASSET.LOADALL TYPE= assettypes.otype PREFIX=Child
assettypes.otype LIST=childList/>
    <SETCOUNTER NAME=numassettypes.otype
VALUE=childList.#numRows/>
</LOOP>

```

ASSET.CREATE

Creates an asset object instance.

Syntax

```

<ASSET.CREATE
    NAME="asset instance name"
    TYPE="asset type" />

```

Parameters

NAME (required)

Name to assign the new asset instance.

TYPE (required)

Name of the asset type.

Description

Data from the object is written to the database by [ASSET . SAVE](#).

errno

The possible values of errno include:

Value	Description
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing.
-10010	Invalid attribute value.

ASSET.DEF

Returns a list of an asset type's attributes and indicates whether each attribute is optional, required, read only, or editable.

Syntax

```
<ASSET.DEF
  TYPE="asset type name"
  [PUBID="publication ID"]
  [SUBTYPE="asset subtype"]
  LIST="list name" />
```

Parameters

TYPE (required)

Name of the asset type.

PUBID (optional)

Site (formerly publication) ID.

SUBTYPE (optional)

Name of the asset subtype.

LIST (required)

Name of list to be created. Columns are:

- **name** – Attribute name, description.
- **editable** – Set to `true` if the attribute can be edited in the UI; set to `false` for read-only.
- **required** – Set to `true` if this is a required field in the database table; set to `false` if the field is optional.
- **description** – Name that can be displayed on the screen.
- **formattable** – Set to `true` if a field is a simple string field.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

ASSET.EXPORT

Gathers Content Server variables to create an XML rendition of an asset.

Syntax

```
<ASSET.EXPORT
  NAME="asset instance object"
  PREFIX="prefix"
  [PUBID="publication ID"]
  [SUBTYPE="asset subtype"]
  OUTPUT="variable name" />
```

Parameters

NAME (required)

Name of the asset instance object.

PREFIX (required)

First part of the name of all variables and/or lists used by this operation. Variables are typically named with the prefix, a delimiter, then the field name. For example, Variables.prefix:fieldname.

PUBID (optional)

Site (formerly publication) ID.

SUBTYPE (optional)

Name of the asset subtype.

OUTPUT (required)

Name of the output variable. This variable contains the resulting XML.

errno

The possible values of errno include:

Value	Description
-12066	No subtype supplied.

Example

The following code is an example of how to export a simple asset and import it in a newly created asset. This code exports a 'Collection' with id 968769614795 and then imports it into the new asset.

```
<SETVAR NAME="errno" VALUE="0"/>
<SETVAR NAME="errdetail1" VALUE="01"/>

<!-- To import 'HomePage Stories' which is a row of Collection -->
<SETVAR NAME="id" VALUE="968769614795"/>
<SETVAR NAME="AssetType" VALUE="Collection"/>

<SETVAR NAME="exportFL" VALUE="LocalFields,Association-
named,Association-unnamed,Publist"/>
```

```

<!-- The output file location -->
<SETVAR NAME="output" VALUE="C:/temp/export.xml"/>

<SETVAR NAME="impTime" VALUE="imported CS.SQLDate"/>

<!-- Login -->
<USER.LOGIN USERNAME="xceladmin" PASSWORD="xceladmin" />

<!-- Load the Collection with id specified earlier -->
<ASSET.LOAD NAME="assetName1" TYPE="Variables.AssetType"
OBJECTID="Variables.id"/>
<br/><br/>ASSET.LOAD: Error #: <CSVAR NAME="Variables.errno"/>,
Error Detail: <CSVAR NAME="Variables.errdetail1"/>

<ASSET.SCATTER NAME="assetName1" PREFIX="exportPrefix1"
EXCLUDE="true" />
<br/><br/>ASSET.SCATTER: Error #: <CSVAR NAME="Variables.errno"/>,
Error Detail: <CSVAR NAME="Variables.errdetail1"/>

<!-- Get the subtype for this asset -->
<ASSET.GETSUBTYPE NAME="assetName1" />
<br/><br/>ASSET.GETSUBTYPE: Error #: <CSVAR
NAME="Variables.errno"/>, Error Detail: <CSVAR
NAME="Variables.errdetail1"/>
<br/>ASSET.GETSUBTYPE: Subtype: <CSVAR NAME="Variables.subtype"/>

<!-- Export the asset -->
<ASSET.EXPORT NAME="assetName1" PREFIX="exportPrefix1"
OUTPUT="exportOutput" />
<br/><br/>ASSET.EXPORT: Error #: <CSVAR NAME="Variables.errno"/>,
Error Detail: <CSVAR NAME="Variables.errdetail1"/>

<!-- Write the xml to a desired location -->
<WRITEFILE STR="Variables.exportOutput" FILE="Variables.output"/>
<br/><br/>WRITEFILE: Error #: <CSVAR NAME="Variables.errno"/>,
Error Detail: <CSVAR NAME="Variables.errdetail1"/>

<!-- Create a new asset for importing the exported file -->
<ASSET.CREATE NAME="newAsset" TYPE="Variables.AssetType"/>
<br/><br/>ASSET.CREATE: Error #: <CSVAR NAME="Variables.errno"/>,
Error Detail: <CSVAR NAME="Variables.errdetail1"/>

<!-- Import the asset in the newly created asset -->
<ASSET.IMPORT NAME="newAsset" PREFIX="exportPrefix1"
XML="Variables.exportOutput" />
<br/><br/>ASSET.IMPORT: Error #: <CSVAR NAME="Variables.errno"/>,
Error Detail: <CSVAR NAME="Variables.errdetail1"/>

<ASSET.GATHER NAME="newAsset" PREFIX="exportPrefix1"
FIELDLIST="Variables.exportFL"/>
<br/><br/>ASSET.GATHER: Error #: <CSVAR NAME="Variables.errno"/>,
Error Detail: <CSVAR NAME="Variables.errdetail1"/>

```

```

<!-- Set the name of the asset to the imported+current time -->
<ASSET.SET NAME="newAsset" FIELD="name" VALUE="Variables.impTime"/>
>
<br/><br/>ASSET.SET: Error #: <CSVAR NAME="Variables.errno"/>,
Error Detail: <CSVAR NAME="Variables.errdetail1"/>

<!-- Save the asset -->
<ASSET.SAVE NAME="newAsset"/>
<br/><br/>ASSET.SET: Error #: <CSVAR NAME="Variables.errno"/>,
Error Detail: <CSVAR NAME="Variables.errdetail1"/>

<ASSET.GET NAME="newAsset" FIELD="name" OUTPUT="getNameOut" />
<br/><br/>ASSET.GET: Error #: <CSVAR NAME="Variables.errno"/>,
Error Detail: <CSVAR NAME="Variables.errdetail1"/>
<BR/> <CSVAR NAME="Variables.getNameOut"/>

<ASSET.GET NAME="newAsset" FIELD="subtype" OUTPUT="getTypeOut" />
<br/><br/>ASSET.GET: Error #: <CSVAR NAME="Variables.errno"/>,
Error Detail: <CSVAR NAME="Variables.errdetail1"/>
<BR/> <CSVAR NAME="Variables.getTypeOut"/>

<ASSET.GET NAME="newAsset" FIELD="updatedby"
OUTPUT="getUserNameOut" />
<br/><br/>ASSET.GET: Error #: <CSVAR NAME="Variables.errno"/>,
Error Detail: <CSVAR NAME="Variables.errdetail1"/>
<BR/> <CSVAR NAME="Variables.getUserNameOut"/>

```

See Also

[ASSET.GETTEMPLATEFORTARGET](#)

ASSET.GATHER

Gathers all variables, where the variable matches a column name for the asset, and sets data in the asset instance.

Syntax

```

<ASSET.GATHER
  NAME="asset instance object"
  [PREFIX="prefix"]
  [FIELDLIST="field names"]
  [EXCLUDE="true|false"] />

```

Parameters

NAME (required)
Name of the asset instance object.

PREFIX (optional)

First part of the name of all variables and/or lists used by this gather operation. Variables are typically named with the prefix together with the field name. If a delimiter, such as a colon, was used in the creation of the variables that you want to gather, you must use the same delimiter in the gather operation. For example, `Variables.prefix:fieldname`.

FIELDLIST (optional)

Specifies a comma-separated list of field names that are gathered. Any field in the list that does not exist in the environment is cleared. If this parameter is not specified, then the gather operation searches all local fields in the environment; but if none are found, the existing value is maintained. The following keywords represent groups of attributes:

- `LocalFields` – Fields of the asset's primary table.
- `Association-named` – Named associations.
- `Association-unnamed` – Unnamed associations.
- `Publist` – Site (formerly publication) IDs.

EXCLUDE (optional)

When set to `true`, this parameter specifies to scatter all fields except those specified in the `FIELDLIST`.

Description

The `ASSET.GATHER` tag performs multiple operations based on a variable naming convention.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

ASSET.GET

Retrieves the value of a single field from a loaded asset object and stores it in a Content Server variable.

Syntax

```
<ASSET.GET  
  NAME="assetName"  
  FIELD="fieldName"  
  [OUTPUT="outputVariable"]/>
```

Parameters

NAME (required)

Input. Name assigned to the asset whose field value you want to get. This asset must be loaded and assigned a name before you can pass its name to this tag. (See [ASSET.LOAD](#).)

FIELD (required)

Input. The name of the field whose value you want to retrieve.

OUTPUT (optional)

Input. The name of the output variable to create to store the value of the field. If you do not supply a name for this variable, CS-Direct uses one with the same name as the field.

Description

This tag obtains the value (or contents) from the specified field of a loaded asset and stores it in the specified output variable.

Use this tag to extract the asset information (from an asset object) that you want to display online. If you want to extract information from more than one field, use [ASSET.SCATTER](#).

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

Example

This code retrieves and displays the contents (value) of the description field:

```
<ASSET.GET NAME="Article1" FIELD="description" OUTPUT="headline"/>  
<CSVAR NAME="Variables.headline"/>
```

This code retrieves the value of the abstract field, writes it to the variable named `abstract`, and then displays the contents of the variable:

```
<ASSET.GET NAME="Article1" FIELD="abstract" />
<CSVAR NAME="Variables.abstract"/>
```

See Also

[ASSET.LOAD](#)

[ASSET.SAVEALL](#)

ASSET.GETASSETTYPE

Loads the asset type object of an asset instance.

Syntax

```
<ASSET.GETASSETTYPE
  NAME="asset instance object"
  OUTNAME="assettype instance object" />
```

Parameters

NAME (required)

Name of the asset instance object.

OUTNAME (required)

Name to assign the asset type instance object.

errno

The possible values of errno include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

ASSET.GETLEGALSUBTYPES

Generates a list of asset subtypes that are valid for this asset type.

Syntax

```
<ASSET.GETLEGALSUBTYPES
  LIST="variable name"
  TYPE="asset type"
  [PUBID="publication ID"] />
```

Parameters

LIST (required)

Name of the output variable.

TYPE (required)

Name of the asset type.

PUBID (optional)

Site (formerly publication) ID. The list of asset subtypes can be restricted by specifying the site ID.

Description

The generated list has a SUBTYPE column.

errno

The possible values of errno include:

Value	Description
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing.

ASSET.GETPUBDEPS

Gets the publish dependencies for an asset.

Syntax

```
<ASSET.GETPUBDEPS
  NAME="asset instance object"
  CONTEXT="PubContext object"
  LIST="variable name"
  [DEPTH="-1|1"] />
```

Parameters

NAME (required)

Name of the asset instance object.

CONTEXT (required)

Name of PubContext object.

LIST (required)

Name of the list to create containing dependent assets.

DEPTH (optional)

Levels to search for dependency. A value of -1 indicates to search for the all dependencies to all levels. The default value is 1.

Description

The default implementation of this tag includes any child asset in the AssetRelationTree table. The generated list contains the following columns: OTYPE, OID, OVERSION.

errno

The possible values of errno include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

Example

```

<ASSET.LOAD NAME="anAsset" TYPE="Variables.assettype" OBJECTID="
Variables.assetid"/>
<if COND="IsError.Variables.errno=false">
  <then>
    <!-- get asset dependents -->

    <ASSET.GETPUBDEPS NAME="anAsset" DEPTH="1"
LIST="myDeps"/>
    <if COND="IsList.myDeps=true">
      <then>
        <LOOP LIST="myDeps">
          <CSVAR NAME="myDeps.otype,myDeps.oid"/>
          <if COND="myDeps.deptype=V">
            <then>
              Exact dependent<br/>
            </then>
          <if>
            <if COND="myDeps.deptype=E">
              <then>
                Exists dependent<br/>
              </then>
            <if>
              </LOOP>
            </then>
          </if>
        </then>
      </if>
    </then>
  </if>
</if>

```

ASSET.GETSITENODE

Queries the SitePlanTree table and returns the node ID of the specified page asset.

Syntax

```
<ASSET.GETSITENODE  
    NAME="pageAssetName"  
    OUTPUT="nodeIDVariableName"/>
```

Parameters

NAME (required)

Input. Name assigned to the page asset whose node ID you want to retrieve. This page asset must be loaded and assigned a name before you can pass its name to this tag. (See [ASSET.LOAD](#).) If this asset is not a page, CS-Direct cannot return a node ID.

OUTPUT (required)

Input. The variable name to assign to the page asset's node ID.

Description

The relationships set up between page assets on the site tree in the CS-Direct main window are stored in the SitePlanTree table. This tag uses the object ID of a page asset to retrieve its node ID from that table.

You can then use the node ID to acquire information about the site's hierarchy to use for display: for example, to create a navigation bar with links to section pages or a link back to the parent page.

errno

The possible values of errno include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory)

Example

This code loads a page asset and then determines the site node of that page:

```
<ASSET.LOAD NAME="NewsPage" TYPE="Page" OBJECTID="Variables.cid"/>  
<ASSET.GETSITENODE NAME="NewsPage" OUTPUT="PageNodeId"/>
```

See Also

The tags described in [Role List Tags](#).

ASSET.GETSUBTYPE

Gets the subtype of the asset.

Syntax

```
<ASSET.GETSUBTYPE
  NAME="loaded asset"
  [OUTPUT="variable name"] />
```

Parameters

NAME (required)

Name of the loaded asset.

OUTPUT (optional)

Name of the variable in which to set the results. If the name is not supplied, subtype is used.

Description

If no subtype has been specified for this asset, Standard (the generic subtype) is returned as the default.

errno

The possible values of errno include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

ASSET.GETTEMPLATEROOTELEMENT

Calculates the name of an asset's template and gets that template asset's root element, without loading the template.

Syntax

```
<ASSET.GETTEMPLATEROOTELEMENT
  NAME="asset instance object"
  [TYPE="asset type"]
  [TEMPLATE="template name"]
  OUTPUT="variable name" />
```

Parameters

NAME (required unless TYPE and TEMPLATE parameters used)

Name of the asset instance object.

TYPE (optional unless TEMPLATE parameter used)

Asset type of the asset.

TEMPLATE (optional)

Name of the template. If none is provided, the template name is retrieved from the loaded asset instance.

OUTPUT (required)

Variable in which to store the root element of the calculated template.

errno

The possible values of errno include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-12004	An asset was not found for a specified ID.
-12006	There is no such asset type.

Example

```
<ASSET.LOAD NAME="anAsset" TYPE="Variables.assettype" OBJECTID="Variables.assetid"/>
```

```
<if COND="IsError.Variables.errno=false">
<then>
  <ASSET.GETTEMPLATEFOOTARGET NAME="anAsset"
  OUTPUT="render:element"/>
</then>
</if>
```

ASSET.GETTEMPLATEFOOTARGET

Returns a template name in the designated variable.

Syntax

```
<ASSET.GETTEMPLATEFOOTARGET
  NAME="asset object name"
  TARGET="publish ID"
  OUTPUT="variable name" />
```

Parameters

NAME (required)

Name of the asset object.

TARGET (required)

ID of the Publish target.

OUTPUT (required)

Name of the output variable to create.

Description

This tag checks if there is any data in the AssetDefaultTemplate table (configured by using the Admin/Publishing/publish destination/SetDefaultTemplate function) for the asset type /subtype combination. If there is no data, the tag gets the template from the asset. If there is no template from the asset, it returns the system default template (OpenMarket/DefaultTemplate). This tag displays the template that is used when approving the asset for export.

errno

The possible values of errno include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

Example

```
<ASSET.LOAD NAME="anAsset" TYPE="Variables.assettype" OBJECTID="Variables.assetid"/>
```

```
<if COND="IsError.Variables.errno=false">
<then>
<ASSET.GetTemplateForTarget NAME="anAsset"
TARGET="Variables.target" OUTPUT="defTemplate"/>
<CSVAR NAME="Variables.defteplate"/>
</then>
</if>
```

ASSET.IMPORT

Parses XML and sets Content Server variables which can then be gathered to set values in the asset.

Note

If you use `ASSET . IMPORT` to gather values into a flex asset, you must provide an asset subtype using the `SUBTYPE` parameter.

Syntax

```
<ASSET . IMPORT
  NAME="asset instance object"
  PREFIX="prefix"
  XML="XML code"
  [PUBID="publication ID"]
  [SUBTYPE="asset subtype"] />
```

Parameters

NAME (required)

Name of the asset instance object.

PREFIX (required)

Prefix to use when setting ICS variable names. The variable names are prefix:attribute.

XML (required)

XML code that is parsed.

PUBID (optional)

Site (formerly publication) ID.

SUBTYPE (optional, unless using this tag for a flex asset)

Asset subtype.

errno

The possible values of `errno` include:

Value	Description
-408	Bad XML syntax.
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-12004	An asset was not found for a specified ID.
-12037	Required field missing.
-12038	Incorrect data type.

Value	Description
-12043	Data out of range.
-12060	A field sent to the save asset function (after XML parsing) was too long.
-12066	No subtype supplied.

Example

The following code is an example of how to export a simple asset and import it in a newly created asset. This code exports a 'Collection' with id 968769614795 and then imports it into the new asset.

```
<SETVAR NAME="errno" VALUE="0"/>
<SETVAR NAME="errdetail1" VALUE="01"/>

<!-- To import 'HomePage Stories' which is a row of Collection -->
<SETVAR NAME="id" VALUE="968769614795"/>
<SETVAR NAME="AssetType" VALUE="Collection"/>

<SETVAR NAME="exportFL" VALUE="LocalFields,Association-
named,Association-unnamed,Publist"/>

<!-- The output file location -->
<SETVAR NAME="output" VALUE="C:/temp/export.xml"/>

<SETVAR NAME="impTime" VALUE="imported CS.SQLDate"/>

<!-- Login -->
<USER.LOGIN USERNAME="xceladmin" PASSWORD="xceladmin" />

<!-- Load the Collection with id specified earlier -->
<ASSET.LOAD NAME="assetName1" TYPE="Variables.AssetType"
OBJECTID="Variables.id"/>
<br/><br/>ASSET.LOAD: Error #: <CSVAR NAME="Variables.errno"/>,
Error Detail: <CSVAR NAME="Variables.errdetail1"/>

<ASSET.SCATTER NAME="assetName1" PREFIX="exportPrefix1"
EXCLUDE="true" />
<br/><br/>ASSET.SCATTER: Error #: <CSVAR NAME="Variables.errno"/>,
Error Detail: <CSVAR NAME="Variables.errdetail1"/>

<!-- Get the subtype for this asset -->
<ASSET.GETSUBTYPE NAME="assetName1" />
<br/><br/>ASSET.GETSUBTYPE: Error #: <CSVAR
NAME="Variables.errno"/>, Error Detail: <CSVAR
NAME="Variables.errdetail1"/>
<br/>ASSET.GETSUBTYPE: Subtype: <CSVAR NAME="Variables.subtype"/>

<!-- Export the asset -->
```

```

<ASSET.EXPORT NAME="assetName1" PREFIX="exportPrefix1"
OUTPUT="exportOutput" />
<br/><br/>ASSET.EXPORT: Error #: <CSVAR NAME="Variables.errno"/>,
Error Detail: <CSVAR NAME="Variables.errdetail1"/>

<!-- Write the xml to a desired location -->
<WRITEFILE STR="Variables.exportOutput" FILE="Variables.output"/>
<br/><br/>WRITEFILE: Error #: <CSVAR NAME="Variables.errno"/>,
Error Detail: <CSVAR NAME="Variables.errdetail1"/>

<!-- Create a new asset for importing the exported file -->
<ASSET.CREATE NAME="newAsset" TYPE="Variables.AssetType"/>
<br/><br/>ASSET.CREATE: Error #: <CSVAR NAME="Variables.errno"/>,
Error Detail: <CSVAR NAME="Variables.errdetail1"/>

<!-- Import the asset in the newly created asset -->
<ASSET.IMPORT NAME="newAsset" PREFIX="exportPrefix1"
XML="Variables.exportOutput" />
<br/><br/>ASSET.IMPORT: Error #: <CSVAR NAME="Variables.errno"/>,
Error Detail: <CSVAR NAME="Variables.errdetail1"/>

<ASSET.GATHER NAME="newAsset" PREFIX="exportPrefix1"
FIELDLIST="Variables.exportFL"/>
<br/><br/>ASSET.GATHER: Error #: <CSVAR NAME="Variables.errno"/>,
Error Detail: <CSVAR NAME="Variables.errdetail1"/>

<!-- Set the name of the asset to the imported+current time -->
<ASSET.SET NAME="newAsset" FIELD="name" VALUE="Variables.impTime"/>
>
<br/><br/>ASSET.SET: Error #: <CSVAR NAME="Variables.errno"/>,
Error Detail: <CSVAR NAME="Variables.errdetail1"/>

<!-- Save the asset -->
<ASSET.SAVE NAME="newAsset"/>
<br/><br/>ASSET.SET: Error #: <CSVAR NAME="Variables.errno"/>,
Error Detail: <CSVAR NAME="Variables.errdetail1"/>

<ASSET.GET NAME="newAsset" FIELD="name" OUTPUT="getNameOut" />
<br/><br/>ASSET.GET: Error #: <CSVAR NAME="Variables.errno"/>,
Error Detail: <CSVAR NAME="Variables.errdetail1"/>
<BR/> <CSVAR NAME="Variables.getNameOut"/>

<ASSET.GET NAME="newAsset" FIELD="subtype" OUTPUT="getTypeOut" />
<br/><br/>ASSET.GET: Error #: <CSVAR NAME="Variables.errno"/>,
Error Detail: <CSVAR NAME="Variables.errdetail1"/>
<BR/> <CSVAR NAME="Variables.getTypeOut"/>

<ASSET.GET NAME="newAsset" FIELD="updatedby"
OUTPUT="getUserNameOut" />
<br/><br/>ASSET.GET: Error #: <CSVAR NAME="Variables.errno"/>,
Error Detail: <CSVAR NAME="Variables.errdetail1"/>
<BR/> <CSVAR NAME="Variables.getUserNameOut"/>

```


See Also

[ASSET.EXPORT](#)

ASSET.INSPECT

Creates an object with the asset's attributes and the specification for each attribute.

Syntax

```
<ASSET.INSPECT
  LIST="list name"
  TYPE="asset type"
  [PUBID="publication ID"]
  [SUBTYPE="asset subtype"] />
```

Parameters

LIST (required)
Name of the list.

TYPE (required)
Asset type.

PUBID (optional)
Site (formerly publication) ID.

SUBTYPE (optional)
Asset subtype.

Description

The returned list object has columns NAME and SPEC.

errno

The possible values of `errno` include:

Value	Description
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-12004	An asset was not found for a specified ID.

ASSET.LIST

Queries the database and retrieves a list of assets that meet the specified criteria. It creates a list of assets of one type.

Syntax

```
<ASSET.LIST
  TYPE="assetType"
  LIST="nameOfList"
  [FIELD1="fieldName"]
  [VALUE1="Variables.id"]
  [ORDER="order"]
  [PUBID="site id"/>
```

Parameters

TYPE (required)

Input. The asset type of the assets to list.

LIST (required)

Input. The name to assign to the list of assets. If a list with this name already exists, it is replaced with this new list.

FIELD (1-9) (optional)

Input. The name of a field to use to restrict the list. You can specify up to nine fields. If you specify a field name, you must also pass in a corresponding **VALUE** for the field. For example: **FIELD1**="Category" **VALUE1**= "Sports".

VALUE (1-9) (optional)

Input. Required if **FIELD** specified. The field value to use to restrict the list. You can specify up to nine **FIELD**/**VALUE** pairs.

ORDER (optional)

Input. The fields to sort the list by and whether the sort on those fields is ascending (asc) or descending (desc). By default, the sort is ascending. If you specify more than one field, separate the field names with a comma.

For example, if you specify **ORDER**="nrank", the list is sorted by rank starting at number 1. If you want the list sorted by descending rank, use **ORDER**="nrank, desc".

PUBID (optional)

ID of the site (formerly publication) to which the query is restricted. When this optional parameter is used, the query is a database join operation against the AssetPublication table.

Description

This tag queries the database table for assets of this type and then builds a list of assets, selecting those that meet certain criteria. The criteria you can use include asset type and up to nine field/value pairs of fields that define assets of that type.

You typically use this tag to retrieve multiple assets of the same type that do not necessarily have associations or relationships with each other.

errno

The possible values of **errno** include:

Value	Description
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing.

Example

This code creates a list named "ArticleList" composed of all the article assets that have the keyword "President":

```
<ASSET.LIST TYPE="Article" LIST="ArticleList" FIELD1="keyword"
VALUE1="President"/>
```

See Also

[Chapter 1, "Content Server XML Tags,"](#) for the standard Content Server list tags.

ASSET.LOAD

Queries the database for the specified basic asset, and then loads an instance of the asset into memory as an object.

Note

This tag cannot be used to load a flex asset. However it can be used to load attributes of type assets within flex assets. To load flex assets, see [Assetset Tags](#).

Syntax

```
<ASSET . LOAD
    NAME="assetName"
    TYPE="assetType"
    OBJECTID="object.id"
    [FIELD="fieldName"]
    [VALUE="fieldValue"]
    [DEPTYPE="exact|exists|greater"] />
```

Parameters

NAME (required)

Input. The name to use to identify the asset after it is retrieved and stored as an object in memory. All references to the asset by other ASSET tags (ASSET . CHILDREN and ASSET . GET, for example) during this session use this name to identify the asset.

TYPE (required)

Input. The asset type of the asset that you want to retrieve from the database. Typically, you provide TYPE and OBJECTID. In special cases, when you know that a

specific FIELD/VALUE pair can uniquely identify the asset, you can provide TYPE and the FIELD/VALUE pair.

OBJECTID (required)

Input. The object ID of the asset. It is not required if you use TYPE with a FIELD/VALUE pair (defined below).

FIELD (optional)

Input. Used in a pair of values with VALUE to identify the asset by supplying a field name whose corresponding value uniquely identifies the asset. Note that if the FIELD/VALUE pair that you supply identifies more than one asset, CS-Direct loads the first one that it finds.

VALUE (optional)

Input. Used in a pair of values with FIELD to identify the asset by supplying a value for a field that uniquely identifies the asset.

DEPTYPE (optional)

When using the ASSET.LOAD tag in a template, and the assets using this template have been approved for export, this attribute specifies whether approval dependency is exact, exists, or greater.

Legal values are:

- **exact(default)** – Specifies a version dependency. The version of the dependent asset must be equal to the version of the asset when it was approved.
- **exists** – Specifies that any version of the asset satisfies the dependency condition.
- **greater** – Specifies the asset must be at a or above a certain version.

Description

This tag executes a database query to retrieve an instance of the specified asset and then saves the instance into memory (session information), assigning it the name provided by the NAME parameter. That object is then available until the session is flushed (the root element exits) or the name is overwritten. Object names are global in scope.

You can identify the asset that you want to load with its asset type and object ID, or the asset type combined with a FIELD/VALUE pair that uniquely identifies the asset.

Typically, each template element starts with an ASSET . LOAD tag so that subsequent code can extract and display the loaded data on pages for your online site.

Note that ASSET . LOAD does not load information from the AssetPublication table or the AssetRelationTree table, for example. If you need information from the AssetRelationTree table, use the ASSET . CHILDREN tag. If you need information from other auxiliary tables, you can issue queries using the object ID of the loaded asset.

When you load flex assets, the data from other variables is loaded. When you load a page asset, you also have access to information about that asset's node in the SitePlanTree table.

errno

The possible values of errno include:

Value	Description
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.

Value	Description
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-10006	The object ID is not valid.
-10007	The version of the object is not valid.
-10009	More than one object met the specified criteria.

Example

This code loads an article asset, identifying it by type and object ID:

```
<ASSET.LOAD NAME="FirstArticle" TYPE="Article"
OBJECTID="Variables.cid"/>
```

This code loads a page asset, identifying it by type and a FIELD/VALUE pair:

```
<ASSET.LOAD NAME="NewsSectionFront" TYPE="Page" FIELD="category"
VALUE="News"/>
```

See Also

[ASSET.GETTEMPLATEROOTELEMENT](#)

ASSET.LOADALL

Calls `ASSET.LOAD` for all IDs passed in the input list and creates objects with names made up of the prefix name plus a counter.

Syntax

```
<ASSET.LOADALL
  TYPE="asset type"
  [LIST="list object name"]
  [IDS="asset IDs"]
  [IDFIELD="field name"]
  [PREFIX="prefix"]
  [INITCOUNT="integer"]
  [DEPTYPE="exact, exists, or greater"] />
```

Parameters

TYPE (required)

Type of the asset to load.

LIST (optional)

You must use either the **LIST** or **IDS** parameter. Name of an object containing the object IDs of assets to load.

IDS (optional)

You must use either the LIST or IDS parameter. A variable containing a semicolon-separated list of asset IDs to load.

IDFIELD (optional)

Name of the field in the list object which specifies the object ID of the asset. The default value is ID.

PREFIX (required)

Base name of the loaded asset. The object name consists of this value plus a counter incremented by 1 for each asset loaded. A variable named Prefix+Total holds the number of assets loaded.

INITCOUNT (optional)

Initial value of the counter to be appended to the PREFIX value to determine the object name of the loaded asset. The default value is 1.

DEPTYPE (optional)

When the ASSET.LOADALL tag is used in a template and assets using this template are approved for export, this parameter specifies whether approval dependency is exact, exists, or greater. The default value is exists.

Description

If a list is passed as input and the list contains at least as many columns as the primary storage table for the asset type, the asset is loaded from the list without further querying of the database. If a list is passed with fewer columns than the number of columns in the asset type's primary storage table, then the ID is extracted and used to query the asset type's storage table to load the table. If a string is passed with a list of IDs, those are used to load the asset type table. A variable, with a name consisting of the prefix+Total indicates how many assets were loaded.

errno

A list of error messages is written to Variables.errdetailsn, where n is an iterator beginning with 1.

Example

This example loads all images in the database and applies the names img1, img2, img3, etc. Because the input list contains all the fields of the asset type's primary storage table, the asset is loaded from the list data without further queries to the database.

```
<ASSET.LIST TYPE="Image" LIST="AllImages"/>
<ASSET.LOADALL LIST="AllImages" TYPE="Image" PREFIX="img"/>
<CSVAR NAME="Variables.imgTotal"/> images loaded.
<CSVAR NAME="Variables.errno"/>
```

This example loads all the images for a particular publication from the AssetPublication table. This makes at least one query to get the assets to load.

```
<SETVAR NAME="pubid" VALUE="949611132389"/>
<SETVAR NAME="assettype" VALUE="Image"/>
<SELECTTO LIST="ezineImages" FROM="AssetPublication"
WHERE="pubid,assettype"/>
<ASSET.LOADALL LIST="ezineImages" TYPE="Image" PREFIX="eZineImg"
IDFIELD="assetid"/>
<CSVAR NAME="Variables.ezineImagesTotal"/> images loaded.
```

```
<CSVAR NAME="Variables.errno"/>
```

This example loads assets by passing a semicolon separated list of IDs. This is useful as the Post element of a form where the user checked the IDs of assets that the user wanted to use.

```
<SETVAR NAME="idlist"
VALUE="960215146767;960215146907;960215148305"/>
<ASSET.LOADALL IDS="Variables.idlist" TYPE="Image" PREFIX="image"/
>
<CSVAR NAME="Variables.imageTotal"/> images loaded.
<CSVAR NAME="Variables.errno"/>
```

errno

The possible values of errno include:

Value	Description
-1000	Revision tracking error.
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing.

ASSET.LOADREVISION

Loads a previous revision of a revision-tracked asset using the ASSET.LOAD operation.

Syntax

```
<ASSET.LOADREVISION
  TYPE="asset type"
  NAME="object name"
  OBJECTID="asset ID"
  REVISION="integer" />
```

Parameters

TYPE (required)

Type of the asset to load.

NAME (required)

Object name to assign the loaded asset.

OBJECTID (required)

Asset ID.

REVISION (required)

Integer indicating which asset revision to load.

errno

The possible values of `errno` include:

Value	Description
-1000	Revision tracking error.
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing.
-12004	An asset was not found for a specified ID.

More specific error numbers are set in `Variables.errrdetail`. See [errrdetail Error Values](#) for more information.

ASSET.REFERENCEDBY

Builds a list of assets that reference this asset.

Syntax

```
<ASSET.REFERENCEDBY
    NAME="asset name"
    LIST="list name" />
```

Parameters

NAME (required)

Name of previously loaded asset

LIST (required)

Name of the list of assets that reference the asset. The list contains a column called `OTYPE`, which contains the asset type and the `OID` for the ID of the referencing asset.

Description

The list contains the fields of the `AssetRelationTree`.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.

Value	Description
-10005	The requested object is not in the object pool (is not loaded into memory).

Example

This example loads assets referencing myArticle and displays the name, description, and asset type of each.

```
<ASSET.LOAD NAME=myArticle OBJECTID=123456789012/>
<ASSET.REFERENCEDBY NAME=myArticle LIST=referers/>
<LOOP LIST=referers>
  <ASSET.LOAD NAME=referer TYPE=referers.otype
OBJECTID=referers.oid/>
  <ASSET.GETASSETTYPE NAME= referer OUTNAME=at/>
  <ASSETTYPE.GET NAME=at FIELD=description OUTPUT=assettype/>
  <ASSET.GET NAME=referer FIELD=name/>
  <ASSET.GET NAME=referer FIELD=description/>
  <CSVAR NAME= Asset Type: Variables.assettype Name:
Variables.name Description: Variables.description/>
</LOOP>
```

ASSET.REMOVECHILDREN

Designates which entries to remove from the AssetRelationTree.

Syntax

```
<ASSET.REMOVECHILDREN
  NAME="asset instance object"
  TYPE="asset type"
  CHILDID="child asset ID"
  CODE="association name" />
```

Parameters

NAME (required)

Name of the asset instance object.

TYPE (optional)

Asset type of children to remove. If this parameter is omitted, children of all types are removed.

CHILDID (optional)

ID of a child asset to remove.

CODE (optional)

Association name of child asset to remove.

Description

The AssetRelationTree is updated during an [ASSET.SAVE](#) operation.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

ASSET.SAVE

Saves an asset.

Syntax

```
<ASSET.SAVE
  NAME="asset instance object"
  [FLUSH="true|false"] />
```

Parameters

NAME (required)

Name of the asset instance object.

FLUSH (optional)

If set to `true`, flushes the asset from the object pool.

Description

This tag saves an asset to its storage and, if it is a new asset, to the AssetPublication table. Associations defined by using the `AddChild` `RemoveChildren` or `ChildGather` methods are also saved there.

The following fields in the asset are updated by default:

- **status** – If no value is supplied, the status is `m_sStatusCreated` (PL).
- **createdby** – If no value is supplied and this is a new asset set to the session variable `username`.
- **createddate** – If no value is supplied and this is a new asset, set to the current date.
- **updatedby** – If no value is supplied, set to session variable `username`.
- **createddate** – If no value is supplied, set to the current date.

Saving to the AssetPublication table uses the following logic and variables:

- **primarypubid** (variable) – An entry is made based on the variable `primarypubid`.
- **pubid** (session variable) – If `primarypubid` does not exist, the session variable `pubid` defines the publication in which to put the asset.

errno

The possible values of `errno` include:

Value	Description
-105	Database error.
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-12018	Missing site.
-12034	Publish in progress.
-12065	Error updating AssetPublication.

ASSET.SAVEALL

Calls `ASSET.SAVE` for all assets named with the prefix name plus a counter.

Syntax

```
<ASSET.SAVEALL
  TYPE="asset type"
  PREFIX="loaded asset name"
  [INITCOUNT="integer"]
  [FLUSH="true or false"] />
```

Parameters

TYPE (required)

Type of the asset to save.

PREFIX (required)

Base name of the loaded asset. The object name consists of this value plus a counter incremented by 1.

INITCOUNT (optional)

The initial value of the counter to be appended to the **PREFIX** value to determine the object name of the loaded asset. The default value is 1.

FLUSH (optional)

If set to `true`, the object is flushed from the object pool. The default value is `false`.

errno

When an error is encountered, other assets are saved. A list of error messages is written to `Variables.errrdetailsn`, where *n* is an iterator beginning with 1.

The possible values of `errno` include:

Value	Description
-105	Database error.
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-12018	Missing site.
-12034	Publish in progress.
-12065	Error updating AssetPublication.

Example

This example saves the images named `image100`, `image101`, `image102`, etc.

```
<ASSET.LOADALL IDS="Variables.idlist" TYPE="Image" PREFIX="image"
INITCOUNT="100"/>
<CSVAR NAME="Variables.errno"/>

<ASSET.SAVEALL TYPE="Image" PREFIX="image" INITCOUNT="100"/>
<CSVAR NAME="Variables.errno"/>
<SETCOUNTER NAME="errCount" VALUE="1"/>
<LOOP UNTIL="IsVariable.errdetailCounters.errcount=false">
  <CSVAR NAME="Variables.errdetailCounters.errcount"/><br/>
  <INCCOUNTER NAME="errCount" VALUE="1"/>
</LOOP>
```

See Also

[ASSET.LOADALL](#)

ASSET.SCATTER

Retrieves data, creates Content Server variables for all data of the loaded asset, and lists (“scatters”) the values from all the fields of a loaded asset object. (To obtain just one field value, use [ASSET.GET](#).)

Syntax

```
<ASSET . SCATTER
    NAME="assetName"
    [PREFIX="variablePrefix"]/>
```

Parameters

NAME (required)

Input. The name assigned to the asset whose fields you want to list (scatter). This asset must be loaded and assigned a name before you can pass its name to this tag. (See [ASSET.LOAD](#).)

PREFIX (optional)

Input. Optional but rarely omitted. Specifies a string to be included in the name of the variables that store the asset’s information. When you specify a prefix, you then create your variables by using a colon to separate the prefix from the field name. For example, if you specify `PREFIX="article"`, your variables would be `Variables.article:fieldname`.

Description

This tag retrieves values from the fields of a previously loaded asset and then writes them into memory (session information) as variables. The variables are constructed with the prefix that you supply with the `PREFIX` parameter. After you use this tag, you can display the information contained in the asset’s fields. This tag only supports plain text or HTML files.

Use this tag rather than multiple `ASSET.GET` commands when you want to retrieve more than a couple of fields.

`ASSET.SCATTER` does not retrieve and scatter the contents of URL fields. To retrieve the contents of URL fields use the `ICS.GETVAR` tag or the BlobServer.

The property `xcelerate.assettype.loadonscatter` located in the `futuretense_xcel.ini` file, can contain a comma-separated list of URL fields to be loaded when the `ASSET.SCATTER` tag is used. For instance, if you want Article’s `urlbody` field to behave this way set the property `xcelerate.Article.loadonscatter` to `urlbody`.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

Example

This code retrieves values from a loaded asset named `Article1`, writes variables for those values into memory, and then displays the data in the asset's description and abstract fields:

```
<ASSET.SCATTER NAME="Article1" PREFIX="article"/>
  <CSVAR NAME="Variables.article:description"/>
  <CSVAR NAME="Variables.article:abstract"/>
```

See Also

[ASSET.LOAD](#)

[ASSET.GET](#)

ASSET.SEARCH

Locates a list of asset primary table rows based on the asset type and a set of search criteria.

Syntax

```
<ASSET.SEARCH
  TYPE="asset type"
  [SUBTYPE="asset subtype"]
  PREFIX="prefix"
  [FIELDLIST="field name(s)"]
  LOCALFIELDS="fields"
  PUBLIST="publication ID"
  [EXCLUDE="true|false"]
  [LIMIT="maximum assets"]
  [ORDER="order-by string"]
  LIST="list variable name" />
```

Parameters

TYPE (required)

Asset type of assets for which to search.

SUBTYPE (optional)

Asset subtype.

PREFIX (required)

Prefix for variable names used to specify search criteria .

FIELDLIST (optional)

A comma-separated list of field names that may have search criteria specified. If this argument is not specified, then all local fields are considered.

LOCALFIELDS

Fields of the asset's primary table

PUBLIST

Site (formerly publication) ID.

EXCLUDE (optional)

When set to `true`, it specifies to scatter all fields except those specified by the **FIELDLIST** parameter.

LIMIT (optional)

Maximum number of assets to return.

ORDER (optional)

An Order-By string to add to the end of the query. You can only use fields in the asset type's primary table for the order-by string.

LIST (required)

Name of resulting list.

Description

The search criteria are gathered from Content Server variables. For attributes of simple types, the variable search criteria is gathered from the following:

- *prefix:attribute_op* – Search operation. Valid values are =, !=, <, >, <=, >=, like, or not like.
- *prefix:attributeTotal* – Number of criteria for this attribute to string together with ORs.
- *prefix:attributen* – Where *n* is a number between 0 and *prefix:attributeTotal* -1, indicating the search value.

If there is no *prefix:attributeTotal* variable:

- *prefix:attribute* - search value

errno

The possible values of errno include:

Value	Description
-105	Database error.
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10001	The implementing class is invalid.
-12006	There is no such asset type.

Example

This example shows a search for non-voided Articles in a certain publication with a name containing World or General.

```
<SETVAR NAME="searchfor:status_op" VALUE="!="/>
<SETVAR NAME="searchfor:status" VALUE="V0"/>
<SETVAR NAME="searchfor:name_op" VALUE="like"/>
<SETVAR NAME="searchfor:nameTotal" VALUE="2"/>
<SETVAR NAME="searchfor:name0" VALUE="General"/>
<SETVAR NAME="searchfor:name1" VALUE="World"/>
<SETVAR NAME="searchfor:Publist_op" VALUE=""/>
<SETVAR NAME="searchfor:PublistTotal" VALUE="1"/>
```

```

<SETVAR NAME="searchfor:Publist0" VALUE="968251170475"/>

<ASSET.SEARCH TYPE="Article" PREFIX="searchfor" LIST="mylist"
FIELDLIST="Publist,LocalFields"/>
<CSVAR NAME="mylist.#numRows"/> assets returned<br/>
<LOOP LIST="mylist">
    <CSVAR NAME="mylist.name"/>, <CSVAR NAME="mylist.description"/>
<br/>
</LOOP>

```

ASSET.SET

Sets a field in an asset.

Syntax

```

<ASSET.SET
    NAME="asset instance object"
    FIELD="field name"
    VALUE="value" />

```

Parameters

NAME (required)
Name of the asset instance object.

FIELD (required)
Field to set.

VALUE (required)
Value to set the field to.

errno

The possible values of errno include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-12065	Error updating AssetPublication.

ASSET.SHARE

Updates the AssetPublication table to have an entry for each of the requested sites.

Syntax

```
<ASSET.SHARE
    NAME="asset instance object"
    PUBLIST="publication IDs" />
```

Parameters

NAME (required)

Name of the asset instance object.

PUBLIST (required)

Semicolon-separated list of site (formerly publication) IDs from which the asset is accessible.

Example

```
<!-- Make sure that current pub is part of the list -->

<if COND="IsVariable.pubidlist=true">
<then>
    <SETVAR NAME="pubidlist"
VALUE="Variables.pubidlist;SessionVariables.pubid"/>
</then>

<else>
    <SETVAR NAME="pubidlist" VALUE="SessionVariables.pubid"/>
</else>
</if>

<ASSET.LOAD TYPE="Variables.AssetType" OBJECTID="Variables.id"
NAME="theCurrentAsset"/>
<ASSET.SHARE NAME="theCurrentAsset" PUBLIST="Variables.pubidlist"/
>
```

ASSET.SITEPARENT

Queries the SitePlanTree table and then loads the parent page of the specified page asset into memory as an object.

Syntax

```
<ASSET . SITEPARENT
    NAME="pageName"
    OUTNAME="parentPageName" />
```

Parameters

NAME (required)

Input. The name of the page asset whose parent page you want to load. This page asset must be loaded and assigned a name before you can pass its name to this tag. (See [ASSET . LOAD](#).)

OUTNAME (required)

Input. The name to assign to the parent page object.

Description

This tag queries the SitePlanTree table and then loads as an object the parent page of the specified page asset. It functions similarly to `ASSET . LOAD`.

You typically use this tag to display information about the hierarchical position of the current page; for example, to create a link to the current page's parent page. This tag determines the parent page and then loads the parent page in one database query.

errno

The possible values of errno include:

Value	Description
-112	No parent exists.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-10007	The version of the object is not valid.
-10008	The node ID of the object is not valid.
-10009	More than one object met the specified criteria.
-12007	The specified value is not valid.

Example

This code loads a page asset, loads its parent page, extracts the name of the parent page asset, and then displays the name:

```
<ASSET.LOAD NAME="myPage" TYPE="Page" OBJECTID="Variables.p"/>
<ASSET.SITEPARENT NAME="myPage" OUTNAME="grandparent"/>
<ASSET.GET NAME="grandparent" FIELD="name"
OUTPUT="grandparent:name"/>
Up one level to <CSVAR NAME="Variables.grandparent:name"/>.
```

See Also

The tags described in [Role List Tags](#).

ASSET.SITES

Creates a list object of sites (formerly known as publications) to which the asset is shared. The list contains the columns of the Publication table.

Syntax

```
<ASSET.SITES
  NAME="asset name"
  LIST="list name"
  PUBID="publication ID" />
```

Parameters

NAME (required)
Name of previously loaded asset.

LIST (required)
Name of list that is created.

PUBID (optional)
Publication ID.

errno

The possible values of errno include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

Example

This example shows all the sites with which this asset is shared.

```
<ASSET.LOAD NAME=myArticle OBJECTID=123456789012/>
<ASSET.SITES NAME=myArticle LIST=sites/>
<LOOP LIST=sites>
    <CSVAR NAME=sites.description/><br/>
</LOOP>
```

This example loops through the list of publications for which the user is enabled and determines whether an asset is in a particular site.

```
<LOOP LIST= pubsICanSee>
    <ASSET.SITES NAME=theAsset LIST=isinPub
PUBID=pubsICanSee.id/>
    <IF COND=IsError.Variables.errno=false>
        <THEN>
            <CSVAR NAME=isinPub.description/><br/>
        </THEN>
    </IF>
</LOOP>
```

ASSET.STREAM

Streams the value of a field from an object.

Syntax

```
<ASSET.STREAM
    NAME="asset instance object"
    FIELD="field name"
    [RESOLVE="true|false"] />
```

Parameters

NAME (required)

Name of the asset instance object.

FIELD (required)

Field to get.

RESOLVE (optional)

If true, resolves Content Server variable names. The default value is true.

errno

The possible values of errno include:

Value	Description
-10004	A required parameter is missing.

Value	Description
-10005	The requested object is not in the object pool (is not loaded into memory).

Example

```
<ASSET.LOAD NAME="anAsset" TYPE="Variables.assettype"
OBJECTID="Variables.assetid"/>

<if COND="IsError.Variables.errno=false">
<then>
    <ASSET.STREAM NAME="anAsset" FIELD="description"
RESOLVE="false"/>
</then>
</if>
```

ASSET.VOID

Sets the status code of the asset to VO and removes any child relations from the AssetRelationTree.

Syntax

```
<ASSET.VOID
    NAME="asset instance object" />
```

Parameters

NAME (required)
Name of the asset instance object.

Description

If there were children in the AssetRelationTree, the root reference remains.

errno

The possible values of errno include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-12018	Missing publication.
-12034	Publish in progress.

CCUser Tags

These tags manage the user that is enabled for the site and is eligible to participate in the workflow process. The following section describes each CCUser tag, beginning with [CCUSER.CLEARROLES](#).

CCUSER.CLEARROLES

Removes all roles for the user.

Syntax

```
<CCUSER.CLEARROLES
    NAME="user name" />
```

Parameters

NAME (required)
Name of the user for whom to clear roles.

Example

```
<!-- Clear all the roles of the user for this site -->
<CCUSER.CLEARROLES NAME="u" />
<CCUSER.GETSITEROLES NAME="u" SITE="968251170475"
OBJVARNAME="roleobject"/>
<ROLELIST.GETALL NAME="roleobject" VARNAME="roles"/>
<IF COND="IsVariable.roles=true">
    <THEN>
        <BR/> After clear roles - CCUSER.GETSITEROLES: <CSVAR
NAME="Variables.roles"/><BR/>
    </THEN>
    <ELSE>
        <BR/> After clear roles - CCUSER.GETSITEROLES: Variable not
set!! <BR/><BR/>
    </ELSE>
</IF> <BR/>
```

CCUSER.GETEMAIL

Gets the e-mail address for the user.

Syntax

```
<CCUSER.GETEMAIL
    NAME="user name"
    VARNAME="variable name" />
```

Parameters

NAME (required)
Name of the user.

VARNAME (required)

Name of the output variable to create.

Example

```
<!-- Get the email for the userobject -->
<CCUSER.GETEMAIL NAME="u" VARNAME="emailOutput1"/>
<IF COND="IsVariable.emailOutput1=true">
  <THEN>
    <BR/> CCUSER.GETEMAIL <CSVAR NAME="Variables.emailOutput1"/
  ><BR/>
  </THEN>
  <ELSE>
    <BR/> CCUSER.GETEMAIL: Variable not set!! <BR/><BR/>
  </ELSE>
</IF> <BR/>
```

CCUSER.GETID

Gets the ID of the user.

Syntax

```
<CCUSER.GETID
  NAME="user name"
  VARNAME="variable name" />
```

Parameters

NAME (required)

Numeric row number for the user. For example, if you had three users, A, B, and C, and you wanted the ID for user B, this parameter would be NAME="2".

VARNAME (required)

Name of the output variable to create.

Example

```
<!-- For each user get the id of the user using CCUSER.ID -->
<setcounter NAME="userCounter" VALUE="0"/>

<loop COUNT="Variables.Users:Total">
  <CCUSER.GETID NAME="Users:Counters.userCounter"
  VARNAME="user_id"/>
  <BR/> CCUSER.GETID: <CSVAR NAME="Variables.user_id"/>
  <inccounter NAME="userCounter" VALUE="1"/>
</loop><BR/>
```

CCUSER.GETNAME

Gets the user name that is displayed on the screen for the user.

Syntax

```
<CCUSER.GETNAME  
    NAME="user name"  
    VARNAME="variable name" />
```

Parameters

NAME (required)

Name of the user.

VARNAME (required)

Name of the output variable to create.

Example

```
<!-- Get the name of the user if you know the id of that user  
using CCUSER.GETNAME -->  
<USERMANAGER.GETUSER USER="1024340678794" OBJVARNAME="usrObj" />  
<CCUSER.GETNAME NAME="usrObj" VARNAME="uname"/>  
<BR/> CCUSER.GETNAME: <CSVAR NAME="User ID: Variables.uname"/>
```

CCUSER.GETSITEROLES

Gets the roles for the user for the specified site.

Syntax

```
<CCUSER.GETSITEROLES  
    NAME="user name"  
    SITE="site ID"  
    OBJVARNAME="variable name" />
```

Parameters

NAME (required)

Name of the user.

SITE (required)

Site ID.

OBJVARNAME (required)

Name of the role list object to create.

Example

```
<!-- Get the roles of the user for this site -->  
<USERMANAGER.GETUSERFROMNAME USERNAME="xceladmin" OBJVARNAME="u"/>  
<CCUSER.GETSITEROLES NAME="u" SITE="968251170475"  
OBJVARNAME="roleobject"/>  
<ROLELIST.GETALL NAME="roleobject" VARNAME="roles"/>
```



```
<BR/> CCUSER.GETSITEROLES: <CSVAR NAME="Variables.roles"/><BR/
><BR/>
```

CCUSER.GETSITES

Gets all the sites for which the specified user is enabled.

Syntax

```
<CCUSER.GETSITES
  NAME="user name"
  OBJVARNAME="variable name" />
```

Parameters

NAME (required)

Name of the user.

OBJVARNAME (required)

Name of the output variable to create.

Example

```
<!-- Get the sites that this user belongs to -->
<USERMANAGER.GETUSER USER="1024340678794" OBJVARNAME="myUserObj" /
>
<CCUSER.GETSITES NAME="myUserObj" OBJVARNAME="siteList"/>
<SITELIST.GETALL NAME="siteList" VARNAME="pubidList"/>
<BR/> CCUSER.GETSITES: <CSVAR NAME="Variables.pubidList"/><BR/
><BR/>
```

CCUSER.SETEMAIL

Sets the e-mail address for a user.

Syntax

```
<CCUSER.SETEMAIL
  NAME="user name"
  VALUE="e-mail address" />
```

Parameters

NAME (required)

Name of the user.

VALUE (required)

User's e-mail address.

Example

```
<!-- Set an email for the userobject -->
<CCUSER.SETEMAIL NAME="u"  VALUE="urmila.singhal@divine.com"/>
<CCUSER.GETEMAIL NAME="u"  VARNAME="emailOutput2"/>
<BR/> CCUSER.GETEMAIL <CSVAR NAME="Variables.emailOutput2"/><BR/
><BR/>
```

CCUSER.SETNAME

Set the username to another name.

Syntax

```
<CCUSER.SETNAME
    NAME="old username"
    VALUE="new username" />
```

Parameters

NAME (required)

Old username.

VALUE (required)

New username.

Example

```
<!-- Set the username to something else -->
<CCUSER.SETNAME NAME="usrObj" VALUE="Urmila"/>
<BR/><BR/> CCUSER.SETNAME: Error #: <CSVAR NAME="Variables.errno"/
>
<CCUSER.GETNAME NAME="usrObj" VARNAME="uname2"/>
<BR/> CCUSER.GETNAME: <CSVAR NAME="User ID: Variables.uname2"/
><BR/><BR/>
```

CCUSER.SETSITEROLES

Sets roles for the user.

Syntax

```
<CCUSER.SETSITEROLES
  NAME="user name"
  SITE="site ID"
  OBJECT="user object name" />
```

Parameters

NAME (required)
Name of the user.

SITE (required)
Site ID.

OBJECT (required)
Name of the user object.

Example

```
<!-- Add roles for this site -->
<CCUSER.GETSITEROLES NAME="u" SITE="968251170475"
OBJVARNAME="srl"/>
<STRINGLIST NAME="useraccls"
STR="Approver,Publisher,WorkflowAdmin,SiteAdmin,Marketer,GeneralAd
min" DELIM=","/>
<LOOP LIST="useraccls">
  <ROLELIST.HASROLE NAME="srl" ROLE="useraccls.ITEM"
VARNAME="hasit"/>
  <IF COND="Variables.hasit=false">
    <THEN>
      <ROLELIST.ADD NAME="srl" ROLE="useraccls.ITEM"/>
      <BR/> Added Role - <CSVAR NAME="useraccls.ITEM"/>
    </THEN>
  </IF>
</LOOP><BR/>
<CCUSER.SETSITEROLES NAME="u" SITE="968251170475" OBJECT="srl"/>
<CCUSER.GETSITEROLES NAME="u" SITE="968251170475"
OBJVARNAME="roleobject"/>
<ROLELIST.GETALL NAME="roleobject" VARNAME="roles"/>
<IF COND="IsVariable.roles=true">
  <THEN>
    <BR/> After setting new roles - CCUSER.GETSITEROLES: <CSVAR
NAME="Variables.roles"/><BR/>
  </THEN>
<ELSE>
    <BR/> After setting new roles - CCUSER.GETSITEROLES:
Variable not set!! <BR/><BR/>
  </ELSE>
</IF><BR/>
```

Date Format Tags

These tags are used for formatting date values in different locales. The following section describes each date format tag, beginning with [DATEFORMAT.CREATE](#).

DATEFORMAT.CREATE

Creates a date-format object, which is later used to convert between a date value and a displayable string.

Syntax

```
<DATEFORMAT.CREATE
  NAME="format object"
  [LOCALE="locale"]
  [DATESTYLE="style"]
  [TIMESTYLE="style"]
  [TIMEZONEID="zone"] />
```

Parameters

NAME (required)

Name of the date format object to create.

LOCALE (optional)

Name of the locale object to use for this date-format object. If not specified, the default is the current Locale object.

DATESTYLE (optional)

Style for date. Default is **MEDIUM**. The possible values are as follows:

- **SHORT** – Completely numeric, such as 12.13.52.
- **MEDIUM** – Longer, such as Jan 12, 1952.
- **LONG** – Longer, such as January 12, 1952.
- **FULL** – Fully specified, such as Tuesday, April 12, 1952 AD.

TIMESTYLE (optional)

Style for time.

- **SHORT** – Completely numeric, such as 3:30pm .
- **MEDIUM** – Longer.
- **LONG** – Longer, such as 3:30:32pm.
- **FULL** – Fully specified, such as 3:30:42pm PST.

TIMEZONEID (optional)

TimeZoneID for the DateFormat object. See [Appendix C, “TimeZoneID Numbers”](#) for a list of ID numbers.

errno

None.

Release Introduced

This tag was introduced in CSEE release 5.0.

DATEFORMAT.GETDATE

Converts a date string from JDBC date format or milliseconds into a displayable date string according to the format of the current DateFormat object.

Syntax

```
<DATEFORMAT.GETDATE  
    NAME="object name"  
    VALUE="date string"  
    VALUETYPE="type"  
    VARNAME="variable name" />
```

Parameters

NAME (required)

Name of the DateFormat object.

VALUE (required)

Date string in JDBC date format or milliseconds.

VALUETYPE (required)

Specify the value type. Possible values are as follows:

- `jdbcdatetime` – Value is in JDBC date format.
- `milliseconds` – Value is in milliseconds.

VARNAME (required)

Name of the variable to create with the converted date string.

errno

None.

Release Introduced

This tag was introduced in CSEE release 5.0.

DATEFORMAT.GETDATETIME

Converts a date string from JDBC date format or milliseconds into a displayable date-time string according to the format of the current DateFormat object.

Syntax

```
<DATEFORMAT.GETDATETIME  
    NAME="object name"  
    VALUE="date string"  
    VALUETYPE="type"  
    VARNAME="variable name" />
```

Parameters

NAME (required)

Name of the DateFormat object.

VALUE (required)

Date string in JDBC date format or milliseconds.

VALUETYPE (required)

Specify the value type. Possible values are as follows:

- jdbcdate – Value is in JDBC date format.
- milliseconds – Value is in milliseconds.

VARNAME (required)

Name of the variable to create with the converted datetime string.

errno

None.

Release Introduced

This tag was introduced in CSEE release 5.0.

DATEFORMAT.GETTIME

Converts a date string from JDBC date format or milliseconds into a displayable time string according to the format of the current DateFormat object.

Syntax

```
<DATEFORMAT.GETDATETIME  
    NAME="object name"  
    VALUE="date string"  
    VALUETYPE="type"  
    VARNAME="variable name" />
```

Parameters

NAME (required)

Name of the DateFormat object.

VALUE (required)

Date string in JDBC date format or milliseconds.

VALUETYPE (required)

Specify the value type. Possible values are as follows:

- jdbcdate – Value is in JDBC date format.
- milliseconds – Value is in milliseconds.

VARNAME (required)

Name of the variable to create with the converted time string.

errno

None.

Release Introduced

This tag was introduced in CSEE release 5.0.

Email Tags

These tags enable you to alter the attributes of an e-mail object after it has been created. The following section describes each Email tag, beginning with [EMAIL.GATHER](#).

EMAIL.GATHER

Gathers all e-mail variables named with the `PREFIX+email` convention into an e-mail object.

Syntax

```
<EMAIL.GATHER
  NAME="email_object_name"
  PREFIX="variablePrefix" />
```

Parameters

NAME (required)

Name of the e-mail object.

PREFIX (required)

First part of the name of all variables and/or lists used by this gather operation. Variables are typically named with the prefix together with the field name. If a delimiter, such as a colon, was used in the creation of the variables that you want to gather, you must use the same delimiter in the gather operation. For example, `Variables.prefix:fieldname`.

errno

The possible values of `errno` include:

Value	Description
-105	Email object has a name conflict.

Example

The following code creates and loads an e-mail object, and then gathers all variables with the specified prefix (`ePrefix` in this example) into this object.

```
<!-- Create and Load e-mail object-->
<EMAILMANAGER.CREATE OBJVARNAME="emailObj"/>
<EMAILMANAGER.LOAD NAME="AssignmentDueReminder"
OBJVARNAME="emailObj"/>
<br/>
<!-- Gathers set variables into email object -->
<EMAIL.GATHER PREFIX="ePrefix:" NAME="emailObj"/>
```


EMAIL.GETBODY

Gets the e-mail body text.

Syntax

```
<EMAIL.GETBODY
  NAME="email_object_name"
  VARNAME="variable_name" />
```

Parameters

NAME (required)

Name of the e-mail object.

VARNAME (required)

Name of the output variable to create. This variable contains the e-mail body.

errno

The possible values of `errno` include:

Value	Description
-105	Email object has a name conflict.

Example

The following code creates and loads an e-mail object, sets the value for e-mail body, and then gets the value for the e-mail body.

```
<!-- Create and Load Email object -->
<EMAILMANAGER.CREATE OBJVARNAME="emailObj"/>
<EMAILMANAGER.LOAD NAME="AssignmentDueReminder"
OBJVARNAME="emailObj"/>
<br/>
<!-- Sets the value for email body -->
<EMAIL.SETBODY NAME="emailObj" VALUE="This is my email body"/>
<br/>
<!-- Gets the value for email body -->
<EMAIL.GETBODY NAME="emailObj" VARNAME="emailBodyOutput"/>
<br/>
Body is: <CSVAR NAME="Variables.emailBodyOutput"/><br/><br/>
```

EMAIL.GETDESCRIPTION

Gets the e-mail description.

Syntax

```
<EMAIL.GETDESCRIPTION  
    NAME="email_object_name"  
    VARNAME="variable_name" />
```

Parameters

NAME (required)

Name of the e-mail object.

VARNAME (required)

Name of the output variable to create. This variable contains the e-mail description.

errno

The possible values of errno include:

Value	Description
-105	Email object has a name conflict.

Example

The following code creates and loads an e-mail object, sets the value for the e-mail description, and then gets the value for the e-mail description.

```
<!-- Create and Load Email object-->  
<EMAILMANAGER.CREATE OBJVARNAME="emailObj"/>  
<EMAILMANAGER.LOAD NAME="AssignmentDueReminder"  
OBJVARNAME="emailObj"/>  
<br/>  
<!-- Sets the value for the email description -->  
<EMAIL.SETDESCRIPTION NAME="emailObj" VALUE="This is my mail  
description"/>  
<br/>  
<!-- Gets the value for the e-mail description -->  
<EMAIL.GETDESCRIPTION NAME="emailObj" VARNAME="emailDescOutput"/>  
<br/>  
Description is: <CSVAR NAME="Variables.emailDescOutput"/>
```

EMAIL.GETID

Gets the e-mail ID.

Syntax

```
<EMAIL.GETID
  NAME="email_object_name"
  VARNAME="variableName" />
```

Parameters

NAME (required)

Name of the e-mail object.

VARNAME (required)

Name of the output variable to create. This variable contains the ID of the e-mail object.

errno

The possible values of `errno` include:

Value	Description
-105	Email object has a name conflict.

Example

The following code creates and loads an e-mail object, and then gets the e-mail ID.

```
<!-- Creates and Loads e-mail object-->
<EMAILMANAGER.CREATE OBJVARNAME="emailObj"/>
<EMAILMANAGER.LOAD NAME="AssignmentDueReminder"
OBJVARNAME="emailObj"/>
<br/>
<!-- Gets the e-mail ID -->
<EMAIL.GETID NAME="emailObj" VARNAME="emailID"/>
<br/>
ID is: <CSVAR NAME="Variables.emailID"/><br/><br/>
```

EMAIL.GETNAME

Gets the e-mail name.

Syntax

```
<EMAIL.GETNAME  
    NAME="email_object_name"  
    VARNAME="variableName" />
```

Parameters

NAME (required)

Name of the e-mail object.

VARNAME (required)

Name of the output variable to create. This variable contains the e-mail name.

errno

The possible values of `errno` include:

Value	Description
-105	Email object has a name conflict.

Example

The following code creates and loads an e-mail object, and then gets the e-mail name.

```
<!-- Create and Load e-mail object-->  
<EMAILMANAGER.CREATE OBJVARNAME="emailObj"/>  
<EMAILMANAGER.LOAD NAME="AssignmentDueReminder"  
OBJVARNAME="emailObj"/>  
<br/>  
<!-- Gets the value for e-mail name -->  
<EMAIL.GETNAME NAME="emailObj" VARNAME="emailNameOutput"/>  
<br/>  
Name is: <CSVAR NAME="Variables.emailNameOutput"/><br/><br/>
```

EMAIL.GETSUBJECT

Gets the e-mail subject text.

Syntax

```
<EMAIL.GETSUBJECT
    NAME="email_object_name"
    VARNAME="variableName" />
```

Parameters

NAME (required)

Name of the e-mail object.

VARNAME (required)

Name of the output variable to create. This variable contains the subject text.

errno

The possible values of `errno` include:

Value	Description
-105	Email object has a name conflict.

Example

The following code creates and loads an e-mail object, sets the value for the e-mail subject, and then gets the e-mail subject.

```
<!-- Create and Load Email object-->
<EMAILMANAGER.CREATE OBJVARNAME="emailObj"/>
<EMAILMANAGER.LOAD NAME="AssignmentDueReminder"
OBJVARNAME="emailObj"/>
<br/>
<!-- Sets the subject of the e-mail -->
<EMAIL.SETSUBJECT NAME="emailObj" VALUE="Subject for my mail"/>
<br/>
<!-- Gets the subject of the e-mail -->
<EMAIL.GETSUBJECT NAME="emailObj" VARNAME="emailSubjectOutput"/>
<br/>
Subject is: <CSVAR NAME="Variables.emailSubjectOutput"/><br/><br/>
```

EMAIL.SCATTER

Writes (or “scatters”) the values of the loaded e-mail object into the variable, PREFIX+email.

Syntax

```
<EMAIL.SCATTER  
  NAME="email_object_name"  
  PREFIX="variable_prefix" />
```

Parameters

NAME (required)

Name of the e-mail object.

PREFIX (required)

First part of the name of all variables and/or lists used by this scatter operation. Variables are typically named with the prefix together with the field name. A delimiter is not automatically inserted after the prefix. Therefore, if you want a delimiter, such as a colon, you must specify it with the prefix. For example, Variables.prefix:fieldname.

errno

The possible values of errno include:

Value	Description
-105	Email object has a name conflict.

Example

The following code creates and loads an e-mail object, sets the value for the e-mail subject, and then scatters the values of the e-mail object.

```
<!-- Create and Load e-mail object-->  
<EMAILMANAGER.CREATE OBJVARNAME="emailObj"/>  
<EMAILMANAGER.LOAD NAME="AssignmentDueReminder"  
OBJVARNAME="emailObj"/>  
<br/>  
<!-- Scatters the values of the e-mail object -->  
<EMAIL.SCATTER PREFIX="ePrefix:" NAME="emailObj"/>
```

EMAIL.SETBODY

Sets the e-mail body text for the e-mail object.

Syntax

```
<EMAIL.SETBODY
    NAME="email_object_name"
    VALUE="body_text" />
```

Parameters

NAME (required)

Name of the e-mail object.

VALUE (required)

Body text.

errno

The possible values of errno include:

Value	Description
-105	Email object has a name conflict.

Example

The following code creates and loads an e-mail object, sets the value for e-mail body, and then gets the value for the e-mail body.

```
<!-- Create and Load e-mail object-->
<EMAILMANAGER.CREATE OBJVARNAME="emailObj"/>
<br/><br/>
<EMAILMANAGER.LOAD NAME="AssignmentDueReminder"
OBJVARNAME="emailObj"/>
<br/><br/>
<!-- Sets the value for e-mail body -->
<EMAIL.SETBODY NAME="emailObj" VALUE="This is my email body"/>
<br/><br/>
<!-- Gets the value for e-mail body -->
<EMAIL.GETBODY NAME="emailObj" VARNAME="emailBodyOutput"/>
<br/><br/>
Body is: <CSVAR NAME="Variables.emailBodyOutput"/><br/><br/>
```

EMAIL.SETDESCRIPTION

Sets the e-mail description for the e-mail object.

Syntax

```
<EMAIL.SETDESCRIPTION  
  NAME="email_object_name"  
  VALUE="description text" />
```

Parameters

NAME (required)

Name of the e-mail object.

VALUE (required)

Description text.

errno

The possible values of `errno` include:

Value	Description
-105	Email object has a name conflict.

The following code creates and loads an e-mail object, sets the value for e-mail description, and then gets the value for the e-mail description.

```
<!-- Create and Load e-mail object-->  
<EMAILMANAGER.CREATE OBJVARNAME="emailObj"/>  
<EMAILMANAGER.LOAD NAME="AssignmentDueReminder"  
OBJVARNAME="emailObj"/>  
<br/>  
<!-- Sets the value for e-mail description -->  
<EMAIL.SETDESCRIPTION NAME="emailObj" VALUE="This is my mail  
description"/>  
<br/>  
<!-- Gets the value for e-mail description -->  
<EMAIL.GETDESCRIPTION NAME="emailObj" VARNAME="emailDescOutput"/>  
<br/>  
Description is: <CSVAR NAME="Variables.emailDescOutput"/>
```


EMAIL.SETNAME

Sets the e-mail name.

Syntax

```
<EMAIL.SETNAME
    NAME="email_object_name"
    VALUE="email_name" />
```

Parameters

NAME (required)

Name of the e-mail object.

VALUE (required)

Specified e-mail name.

errno

The possible values of `errno` include:

Value	Description
-105	Email object has a name conflict.

The following code creates and loads an e-mail object, sets the value for e-mail name, and then gets the value for the e-mail name.

```
<!-- Create and Load e-mail object-->
<EMAILMANAGER.CREATE OBJVARNAME="emailObj"/>
<EMAILMANAGER.LOAD NAME="AssignmentDueReminder"
OBJVARNAME="emailObj"/>
<br/>
<!-- Sets the value for e-mail name -->
<EMAIL.SETNAME NAME="emailObj" VALUE="firstEmailName"/>
<br/>
<!-- Gets the value for e-mail name -->
<EMAIL.GETNAME NAME="emailObj" VARNAME="emailNameOutput"/>
<br/>
Name is: <CSVAR NAME="Variables.emailNameOutput"/><br/><br/>
```

EMAIL.SETSUBJECT

Sets the e-mail subject text for the e-mail object.

Syntax

```
<EMAIL.SETSUBJECT  
    NAME="email_object_name"  
    VALUE="subject_text" />
```

Parameters

NAME (required)

Name of the e-mail object.

VALUE (required)

Subject text.

errno

The possible values of `errno` include:

Value	Description
-105	Email object has a name conflict.

Example

The following code creates and loads an e-mail object, sets the value for the e-mail subject, and then gets the value for the e-mail subject.

```
<!-- Create and Load e-mail object-->  
<EMAILMANAGER.CREATE OBJVARNAME="emailObj"/>  
<EMAILMANAGER.LOAD NAME="AssignmentDueReminder"  
OBJVARNAME="emailObj"/>  
<br/>  
<!-- Sets the subject of the e-mail -->  
<EMAIL.SETSUBJECT NAME="emailObj" VALUE="Subject for my mail"/>  
<br/>  
<!-- Gets the subject of the e-mail -->  
<EMAIL.GETSUBJECT NAME="emailObj" VARNAME="emailSubjectOutput"/>  
<br/>  
Subject is: <CSVAR NAME="Variables.emailSubjectOutput"/><br/><br/>
```

EMAIL.TRANSLATEBODY

Translates the body of the e-mail.

Syntax

```
<EMAIL.TRANSLATEBODY
  NAME="email_object_name"
  PARAMS="names_values"
  VARNAME="variable_name" />
```

Parameters

NAME (required)

Name of the e-mail object.

PARAMS (required)

Set of names and values. This parameter sets the values of all variables used in the subject and body. It is used as name/value pairs separated by an ampersand (that is, &).

VARNAME (required)

Name of the output variable to create. This variable contains the modified body of the e-mail.

errno

The possible values of `errno` include:

Value	Description
-105	Email object has a name conflict.

Example

The following code creates and loads an e-mail object, sets the body of the e-mail, and then translates the body of the e-mail from the original body to the modified body.

```
<!-- Create and Load e-mail object-->
<EMAILMANAGER.CREATE OBJVARNAME="emailObj"/>
<EMAILMANAGER.LOAD NAME="AssignmentDueReminder"
OBJVARNAME="emailObj"/>
<br/>
<!-- Sets the value for e-mail body -->
<EMAIL.SETBODY NAME="emailObj" VALUE="This is my email body"/>
<br/>
<!-- Gets the value for e-mail body -->
<EMAIL.GETBODY NAME="emailObj" VARNAME="emailBodyOutput"/>
<br/>
<!-- Translates the body of the e-mail -->
<SETVAR NAME="paramBody" VALUE="instruction=Translate the body"/>
<EMAIL.TRANSLATEBODY NAME="emailObj" PARAMS="Variables.paramBody"
VARNAME="body"/>
<br/>
Modified Body is: <CSVAR NAME="Variables.body"/><br/><br/>
```

EMAIL.TRANSLATESUBJECT

Translates the e-mail subject.

Syntax

```
<EMAIL.TRANSLATESUBJECT
    NAME="email_object_name"
    PARAMS="names_values"
    VARNAME="variable_name" />
```

Parameters

NAME (required)

Name of the e-mail object.

PARAMS (required)

Set of names and values in standard form.

VARNAME (required)

Name of the output variable to create. This variable contains the modified subject.

errno

The possible values of `errno` include:

Value	Description
-105	Email object has a name conflict.

Example

The following code creates and loads an e-mail object, sets the subject, and then translates the subject of the e-mail from the original subject to the modified subject.

```
<!-- Create and Load E-mail object-->
<EMAILMANAGER.CREATE OBJVARNAME="emailObj"/>
<EMAILMANAGER.LOAD NAME="AssignmentDueReminder"
OBJVARNAME="emailObj"/>
<br/>
<!-- Sets the subject of the e-mail -->
<EMAIL.SETSUBJECT NAME="emailObj" VALUE="Subject for my mail"/>
<br/>
<!-- Gets the original e-mail subject -->
<EMAIL.GETSUBJECT NAME="emailObj" VARNAME="emailSubjectOutput"/>
Subject is: <CSVAR NAME="Variables.emailSubjectOutput"/><br/><br/>
<br/>
<!-- Translates original subject to modified subject -->
<SETVAR NAME="Time" VALUE="00.00" />
<EMAIL.TRANSLATESUBJECT NAME="emailObj"
PARAMS="instruction=Variables.emailSubjectOutput&username=xcedit
or&timetocomplete=Variables.Time" VARNAME="subject"/>
<br/>
Translated Subject is: <CSVARNAME="Variables.subject"/><br/><br/>
```

Email Manager Tags

These tags enable you to create and manage e-mail objects. The following section describes each Email Manager tag, beginning with [EMAILMANAGER.CREATE](#)

EMAILMANAGER.CREATE

Creates an e-mail manager object.

Syntax

```
<EMAILMANAGER.CREATE
  OBJVARNAME="email manager object" />
```

Parameters

OBJVARNAME (required)
Name of an e-mail manager object to create.

errno

The possible values of `errno` include:

Value	Description
-105	Email object has a name conflict.

Example

The following code creates a new e-mail object:

```
<!-- Create a new e-mail object -->
<EMAILMANAGER.CREATE OBJVARNAME="emailObj"/>
```

EMAILMANAGER.DELETE

Deletes an e-mail when based on its ID.

Syntax

```
<EMAILMANAGER.DELETE
  ID="email_ID" />
```

Parameters

ID
ID of the e-mail to delete.

errno

The possible values of `errno` include:

Value	Description
-105	Email object has a name conflict.

Example

The following code creates and loads an e-mail object, and then deletes the e-mail object.

```
<!-- Create e-mail object-->
<EMAILMANAGER.CREATE OBJVARNAME="emailObj"/>
<br/>
<!-- Get ID of e-mail object -->
<EMAIL.GETID NAME="emailObj" VARNAME="emailID"/>
<br/>
<!-- Load e-mail object -->
<EMAILMANAGER.LOAD NAME="Name of EmailManager"
ID="Variables.emailID" OBJVARNAME="emailObj" />
<br/>
<!-- Deleting an EmailManager object -->
<EMAILMANAGER.DELETE ID="Variables.emailID"/>
```

EMAILMANAGER.GETALL

Gets all available e-mail manager objects.

Syntax

```
<EMAILMANAGER.GETALL
    PREFIX="variable_prefix" />
```

Parameters

PREFIX (required)

The first part of the name of all variables and/or lists used by this operation. Variables are typically named with the prefix, a delimiter, and then the field name. For example, *Variables.prefix:fieldname*.

Definition

The following variables are returned as a result of this operation:

- PREFIX+Total** — the total number of e-mail manager objects
- PREFIX+n** (where *n* is a number) — the string used in naming result objects

errno

The possible values of `errno` include:

Value	Description
-105	Email object has a name conflict.

Example

The following code creates and loads an e-mail object, and then gets all available e-mail objects.

```
<!-- Create and Load e-mail object-->
<EMAILMANAGER.CREATE OBJVARNAME="emailObj"/>
<EMAILMANAGER.LOAD NAME="AssignmentDueReminder"
OBJVARNAME="emailObj"/>
<br/>
<!-- Get all the EmailManager's -->
<EMAILMANAGER.GETALL PREFIX="email:"/>
Total # of objects: <CSVAR NAME="Variables.email:Total"/>
<br/>
<!-- Gets the name of the e-mail object-->
<EMAIL.GETNAME NAME="email:1" VARNAME="emailNameOutput"/>
Name of email: <CSVAR NAME="Variables.emailNameOutput"/>
```

EMAILMANAGER.LOAD

Loads the e-mail manager object.

Syntax

```
<EMAILMANAGER.LOAD
    [ID="email_ID"]
    [NAME="process_name"]
    OBJVARNAME="object_name" />
```

Parameters

ID (optional)
ID of the e-mail instance to load.

NAME (optional)
Name of the e-mail instance to load.

OBJVARNAME
Name of an e-mail instance object to create.

Description

Either the e-mail ID or name must be specified.

errno

The possible values of `errno` include:

Value	Description
-105	Email object has a name conflict.

Example

The following code creates and loads an e-mail object.

```
<!-- Create and Load e-mail object-->
<EMAILMANAGER.CREATE OBJVARNAME="emailObj"/>
<EMAILMANAGER.LOAD NAME="AssignmentDueReminder"
OBJVARNAME="emailObj"/>
<br/>
```

EMAILMANAGER.SAVE

Saves the e-mail manager object.

Syntax

```
<EMAILMANAGER.SAVE
    [NAME="process_name"]
    [PREFIX="variable_prefix"]
    OBJECT="email_object_name" />
```

Parameters

NAME (optional)

Name of the workflow process object

PREFIX (optional)

The first part of the name of all variables and/or lists used by this operation. Variables are typically named with the prefix, a delimiter, and then the field name. For example, *Variables.prefix:fieldname*.

OBJECT (required)

Name of an e-mail instance object to save.

errno

The possible values of `errno` include:

Value	Description
-105	Email object has a name conflict.

Example

The following code creates and loads an e-mail object, sets the attributes for the e-mail object, and then saves the e-mail object.

```
<!-- Create and Load e-mail object-->
<EMAILMANAGER.CREATE OBJVARNAME="emailObj"/>
<EMAILMANAGER.LOAD NAME="AssignmentDueReminder"
OBJVARNAME="emailObj"/>
<br/>
<!-- Set some of the e-mail attributes -->
<EMAIL.SETNAME NAME="emailObj" VALUE="Name for Email"/>
<EMAIL.SETDESCRIPTION NAME="emailObj" VALUE="Description"/>
<EMAIL.SETSUBJECT NAME="emailObj" VALUE="Subject of Email"/>
```



```
<EMAIL.SETBODY NAME="emailObj" VALUE="OK. I guess I need to add  
some more information in the body of the Email"/>  
<br/>  
<!-- Save the e-mail object -->  
<EMAILMANAGER.SAVE OBJECT="emailObj"/>
```

InSite Editor Tag

The following section describes the InSite Editor tag, [INSITE.EDIT](#).

INSITE.EDIT

Edits assets from the user interface.

Syntax

```
<INSITE.EDIT
  ASSETID="id of asset"
  ASSETFIELD="Attribute_name of asset's field"
  ASSETFIELDVALUE="incoming field value"
  ASSETTYPE="type of asset"
  [EWEBEDITPRO="true|false"]
  [WIDTH="pixels or percentage"]
  [HEIGHT="pixels"]
  [VARIABLE="variable name"]
  [SSVARIABLE="session variable name"]
  [PROPERTY="property name"]
  [LIST="list name"]
  [COLUMN="column name"] />
```

Parameters

ASSETID (required)

Id of the asset to be edited.

ASSETFIELD (required)

Name of the field of the asset to be edited.

ASSETFIELDVALUE (required)

Value of the incoming field.

ASSETTYPE (required)

Type of asset to be edited.

EWEBEDITPRO (optional)

Legal values are true or false. The default value is true.

WIDTH (optional)

Width of the asset in percentage or pixels. The default value is 100%.

HEIGHT (optional)

Height of the asset in pixels only. The default value is 200 pixels. Please note that percentage values are unsupported.

VARIABLE (optional)

You must use at least one of the following four parameters: VARIABLE, SSVARIABLE, PROPERTY, or LIST and COLUMN (LIST and COLUMN must be used together). Name of the Content Server variable that you want to stream to a page. For example, for a variable called foo, set the parameter equal to foo rather than to Variables.foo.

SSVARIABLE (optional)

You must use at least one of the following four parameters: VARIABLE, SSVARIABLE, PROPERTY, or LIST and COLUMN (LIST and COLUMN must be used together). Name of the Content Server session variable that you want to stream to a page.

PROPERTY (optional)

You must use at least one of the following four parameters: VARIABLE, SSVARIABLE, PROPERTY, or LIST and COLUMN (LIST and COLUMN must be used together). A Content Server property from the futuretense.ini property file.

LIST (optional)

You must use at least one of the following four parameters: VARIABLE, SSVARIABLE, PROPERTY, or LIST and COLUMN (LIST and COLUMN must be used together). Name of the list.

COLUMN (optional)

You must use at least one of the following four parameters: VARIABLE, SSVARIABLE, PROPERTY, or LIST and COLUMN (LIST and COLUMN must be used together). Name of the field in the list.

Description

This tag allows you to edit assets in the Content Server system. When editing standard assets (the ASSETFILEDID is the column name, for example), the Headline field is the description column for Article assets. When editing flex assets the attribute name should be preceded by the prefix `Attribute_`; for example, the body attribute would be `Attribute_body`. For more information about editing assets, see the *CSEE Developer's Guide*.

Example

```
<INSITE.EDIT ASSETID="Variables.cid"
ASSETFIELD="Attribute_headline"
ASSETFIELDVALUE="assetlist.SORT_headline" ASSETTYPE="AArticles"
[EWEBEDITPRO="true"] [WIDTH="100%"] [HEIGHT="200"]/>
```

Participants Tags

This group of tags manage a participants object. The following section describes each Participants tag, beginning with [PARTICIPANTS.ADDPARTICIPANT](#).

PARTICIPANTS.ADDPARTICIPANT

Adds a participant to the participants object.

Syntax

```
<PARTICIPANTS.ADDPARTICIPANT
  NAME="participant object"
  ROLE="role of user"
  USER="user ID" />
```

Parameters

NAME (required)
Name of the participants object.

ROLE (required)
Role of the user to be added.

USER (required)
User ID.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

PARTICIPANTS.CLEAR

Clears all participants from a participants object.

Syntax

```
<PARTICIPANTS.CLEAR
  NAME="participant object" />
```

Parameters

NAME (required)
Name of the participants object.

errno

The possible values of `errno` include:

Value	Description
-12020	No object by name.

PARTICIPANTS.GATHER

Gathers variables into the participant object.

Syntax

```
<PARTICIPANTS.GATHER
  NAME="participant object"
  PREFIX="variable prefix" />
```

Parameters

NAME (required)

Name of the participants object.

PREFIX (required)

First part of the name of all variables and/or lists used by this gather operation. Variables are typically named with the prefix together with the field name. If a delimiter, such as a colon, was used in the creation of the variables that you want to gather, you must use the same delimiter in the gather operation. For example, `Variables.prefix:fieldname`.

errno

The possible values of `errno` include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

PARTICIPANTS.GETALLPARTICIPANTS

Gets all participants from the participants object.

Syntax

```
<PARTICIPANTS.GETALLPARTICIPANTS  
  NAME="participant object"  
  LISTVARNAME="variable list name" />
```

Parameters

NAME (required)

Name of the participant object.

LISTVARNAME (required)

Name of a list to create. This variable contains a column called ITEM that contains the user ID of all participants.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

PARTICIPANTS.GETALLROLES

Gets all roles of the participants object.

Syntax

```
<PARTICIPANTS.GETALLROLES  
  NAME="participant object"  
  VARNAME="variable name" />
```

Parameters

NAME (required)

Name of the participants object.

VARNAME (required)

Name of the output variable to create. This variable contains a comma-separated list of roles.

errno

The possible values of `errno` include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

PARTICIPANTS.GETPARTICIPANTSFORROLE

Get all participants with a specified role from the participants object.

Syntax

```
<PARTICIPANTS.GETPARTICIPANTSFORROLE
  NAME="participant object"
  LISTVARNAME="variable list name"
  ROLE="role name" />
```

Parameters

NAME (required)

Name of the participants object.

LISTVARNAME (required)

Name of a list to create. This variable contains a column called **ITEM** that contains the user IDs.

ROLE (required)

Participants role.

errno

The possible values of `errno` include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

PARTICIPANTS.REMOVEPARTICIPANT

Removes a participant from the participant object.

Syntax

```
<PARTICIPANTS.REMOVEPARTICIPANT  
  NAME="participant object"  
  ROLE="role of user"  
  USER="user ID" />
```

Parameters

NAME (required)
Name of the participants object.

ROLE (required)
Role of the user to be removed.

USER (required)
User ID.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

PARTICIPANTS.SCATTER

Scatters the participant list into prefix+participants.

Syntax

```
<PARTICIPANTS.SCATTER  
  NAME="participant object"  
  PREFIX="variable prefix" />
```

Parameters

NAME (required)
Name of the participants object.

PREFIX (required)
The first part of the name of all variables and/or lists used by this scatter operation. Variables are typically named with the prefix together with the field name. A delimiter is not automatically inserted after the prefix. Therefore, if a delimiter, such as a colon, is desired, it must be included with the prefix. For example, Variables.prefix:fieldname.

Description

The following variables are set as a result of this scatter operation:

- `prefix + roleTotal` – Total number of roles.
- `prefix + rolei` – Role (*i* is 0 to `prefix + roleTotal`).
- `prefix + roleiuserTotal` – Total number of users for this role.
- `prefix + rolei + userj` – User ID of the *j*th user of the *i*th role (*j* is 0 to `prefix + roleiuserTotal - 1`)

errno

The possible values of `errno` include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

PARTICIPANTS.SETPARTICIPANTSFORROLE

Set participants with a specified role for the participants object.

Syntax

```
<PARTICIPANTS.SETPARTICIPANTSFORROLE
  NAME="participant object"
  ROLE="required role"
  LIST="list of users" />
```

Parameters

NAME (required)

Name of the participants object.

ROLE (required)

Role name.

LIST (required)

Name of the list of users to create. This variable contains a column called `ITEM` which contains the user ID.

errno

The possible values of `errno` include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

Property Tag

There is one property tag: [PROPERTY.GET](#).

PROPERTY.GET

Gets a property value from the loaded property files.

Syntax

```
<PROPERTY.GET  
  NAME="PropertyName"  
  [FILE="Property File Name"]  
  [OUTPUT="Some Output"] />
```

Parameters

NAME (required)

Name of the property to retrieve.

FILE (optional)

A semicolon-separated list of property file names.

OUTPUT (optional)

Specifies an optional output variable. If this parameter is not specified, the output is streamed to the browser.

Description

Retrieves a property value from the loaded or specified property file. The property may then be used in conjunction with other XML tags.

errno

None.

Render Tags

CS-Direct renders content in three ways:

- By rendering pages and pagelets into static HTML files through the Export to Disk publishing method.
- By rendering pages and pagelets when a visitor requests them on a dynamic public site.
- By rendering assets in response to the **Preview** function on a CS-Direct management or development system.

CS-Direct determines how to render content by using an argument named `rendermode`, which has four valid settings:

- `DEPS` – Used when the system calculates which assets are dependent on the assets being approved
- `export` – CS-Direct uses the Export to Disk publishing method to create static HTML files
- `live` – CS-Direct invokes Content Server to serve content dynamically
- `preview` – CS-Direct invokes Content Server to render a selected asset in a browser pointed at a CS-Direct management or development system

`rendermode` is set through the `resargs` of page entries in the SiteCatalog table or in the URL.

When CS-Direct creates page entries for templates, it sets `rendermode` as an argument in the `resargs2` column for that entry and sets it to `live`. If you export or preview an asset that uses that template, CS-Direct sets the `rendermode` to `export` or `preview`, as appropriate.

The `RENDER` tags process your CS-Direct page and element requests and create URLs and arguments for URLs. Some of the tags ([RENDER.CALLELEMENT](#) and [RENDER.SATELLITEPAGE](#), for example) function similarly to tags that are available through Content Server or CS-Satellite. However, the `RENDER` tags preserve the CS-Direct rendering environment and the variables needed in that environment without requiring additional code.

The following section describes each render tag, beginning with [RENDER.CALLELEMENT](#).

RENDER.CALLELEMENT

Calls an element from the ElementCatalog table.

Syntax

```
<RENDER.CALLELEMENT
    ELEMENTNAME="nameOfElement"
    [ARGS_var1="value"]/>
```

Parameters

ELEMENTNAME (required)

Input. Name of the element to invoke.

ARGS_ (optional)

Input. Name/value pairs to pass to the called element, with the string **ARGS_** as the prefix for the variable name. For example, CS-Direct sets several variables through the `resargs2` column in a page entry. To pass a value for one of these variables to a **RENDER.CALLELEMENT** tag, you affix **ARGS_** to the beginning of the variable: `ARGS_cid="Variables.id", ARGS_c="article",` and so on.

Note that you can pass in packed arguments using this same syntax. For example: `ARGS_PACKEDARGS="mypackedargs"`

Description

This tag is the CS-Direct equivalent of the Content Server **CALLELEMENT** tag with the following differences:

- Its scope is always local.
- You use CS-Direct syntax for passing it variables. That is, rather than passing arguments with an argument statement (`<ARGUMENT NAME= "variableName" VALUE="variableValue"/>`) you pass them with the **ARGS_** parameter (`ARGS_variableName="variableValue"`).
- `rendermode` is passed to the element if `rendermode` is set to export or preview.

Use this tag to process the content of an element that you wrote for CS-Direct and you want the scope of that element to be stacked. The element must exist in the ElementCatalog.

As with the Content Server **CALLELEMENT** tag, variables that exist prior to **RENDER.CALLELEMENT** are available to tags within the called element. Variables passed to an element are subject to the following order of precedence:

1. Variables set in arguments
2. Variables set in scripts
3. Variables set in the `resdetails1` and `resdetails2` columns in the ElementCatalog

However, because the scope of an element called with this tag is local, variables set within the called element are not available within the parent (calling) element.

errno

The possible values of errno include:

Value	Description
-10	The element does not exist.
-12	The element cannot be evaluated.

Example

The following code sample calls an element named `TopPageBreadcrumb`. It displays the path you took to get to the currently displayed page. For example, if you visited the News page directly from a link on the Home page, the `TopPageBreadcrumb` element displays “Home > News”.

```
<RENDER.CALLELEMENT
  ELEMENTNAME="BurlingtonFinancial/Site/TopPageBreadcrumb"
  ARGS_c="article"/>
```

Because the scope is local, parent page information is not inherited by the called element or returned to the calling element.

See Also

[CALLELEMENT](#)

RENDER.CONTENTSERVER

Calls and renders a page or a pagelet from the SiteCatalog table.

Syntax

```
<RENDER.CONTENTSERVER
  PAGENAME="nameOfPageEntry"
  [ARGS_var1="value"]/>
```

Parameters

PAGENAME (required)

Input. The name of the page entry from the SiteCatalog to render.

ARGS_ (optional)

Input. Name/value pairs to pass to the called element, with the string `ARGS_` as the prefix for the variable name. For example, CS-Direct sets several variables through the `resargs2` column in a page entry. To pass a value for one of these variables to a `RENDER.CALLELEMENT` tag, you affix `ARGS_` to the beginning of the variable: `ARGS_cid="Variables.id"`, `ARGS_c="article"`, and so on.

Note that you can pass in packed arguments using this same syntax. For example: `ARGS_PACKEDARGS="mypackedargs"`

Description

This tag is the CS-Direct equivalent of the Content Server <CONTENTSERVER/> tag with the following differences:

- Its scope is as follows: variables in the calling element are not available in the called element (unless explicitly passed in); changes made to variable values in the called element are not reflected in the calling element; and new variables set in the called element are not available to the calling element.
- You use CS-Direct syntax for passing it variables. That is, rather than passing arguments with an argument statement (<ARGUMENT NAME= "variableName" VALUE="variableValue"/>) you pass them with the ARGS_ parameter (ARGS_variableName="variableValue").
- rendermode is passed to the page entry automatically.

Use this tag to call the ContentServer servlet to serve a CS-Direct page or pagelet. The page or pagelet must have a page entry in the SiteCatalog. If the specified pagelet is already cached, this tag loads it. If the specified pagelet is not already cached, this tag caches it (if caching is enabled for it) and loads it.

Variables passed to the ContentServer Module are subject to the following order of precedence:

1. Variables set in arguments
2. Variables set in scripts
3. Variables set in the resargs1 and resargs2 columns of the SiteCatalog
4. Variables set in the resdetails1 and resdetails2 columns of the ElementCatalog

If an element is coded with a RENDER.SATTLELITEPAGE tag but CS-Satellite is not present, RENDER.SATELLITEPAGE passes the call to RENDER.CONTENTSERVER which then serves the page or pagelet.

errno

The value of errno is unchanged.

Example

This code calls for the page entry of a template named PlainList, passing the ID of the asset that it should display and the ID of the asset's parent page:

```
<RENDER.CONTENTSERVER PAGENAME="BurlingtonFinancial/Collection/  
PlainList" ARGS_cid="SectionHighlights.oid  
ARGS_p="Variables.asset:id"/>
```

See Also

[CONTENTSERVER \(Deprecated\)](#)

[RENDER.SATELLITEPAGE](#)

RENDER.DELETECACHE (Deprecated)

As of Content Server version 4.0 caching is handled by the cache manager.

Syntax

```
<RENDER.DELETECACHE [ARGNAME="SOME ARGUMENT NAME"]
    [VALUE="SOME VALUE"] />
```

Parameters

ARGNAME (optional)

The name of the argument that conditionally limits the cache deletion.

VALUE (optional)

The value corresponding to the argument name in the ARGNAME parameter.

Description

Use this tag to clean out the Content Server element cache. You can specify the conditions under which the cache is deleted.

RENDER.ENCODE

This tag produces a complete URL (from the servlet name onward) from the arguments, previously packed arguments, and the servlet name (usually ContentServer or BlobServer) provided.

Syntax

```
<RENDER.ENCODE
    STR="servletName"
    OUTSTR="variableName"
    [ADDSSESSION="true|false"]
    [ARGS_var1="value"]
    [PACKEDARGS="stringFromPACKARGStag"] />
```

Parameters

STR (required)

Input. Name of the servlet to use when rendering the URL, which is either ContentServer or BlobServer. Or you can include all the information for the URL to the left of the page name (which is supplied dynamically). For example:

`http://localhost/servlet/ContentServer?pagename=index/home`

OUTSTR (required)

Input. Name of the variable that stores the URL generated by this tag.

ADDSSESSION (optional)

Input. Whether or not to include session IDs in the URL when a browser is set to reject cookies. `true` means to encode session IDs in the URL. `false` means not to. If this parameter is not specified, it is set to `true` by default.

ARGS_ (optional)

Input. Name/value pairs to pass to the tag, with the string `ARGS_` as the prefix for the variable name. The names and values of these arguments are URL-encoded and incorporated into the final URL.

An argument passed in with this parameter takes precedence over an argument passed in with the `PACKEDARGS` parameter. If the same argument is passed in with both, the `PACKEDARGS` version is overridden by the `ARGS_` version.

PACKEDARGS (optional)

Input. Previously created URL-encoded packed arguments, in standard form; that is, the output of a previous `RENDER.PACKARGS` tag.

Description

This tag creates a URL based on the arguments and packed arguments that are passed to it. All characters in the URL are URL-encoded.

Typically, you call the standard CS-Direct `GetPageURL` element to obtain URLs for hyperlinks on a page. That element is coded to preserve and resolve the various CS-Direct variables that are necessary for rendering pages in the CS-Direct environment. However, in special cases, you might need to use this tag to obtain a URL instead.

For example, if your site is static but you want to present a dynamic URL inside a static HTML file, use this tag. A URL created with this tag remains dynamic, even when passed through the Export to Disk publishing method. That is, Export to Disk does not translate a dynamic URL created by this tag into a static HTML file.

On a dynamic site, you can use this tag if your page or pagelet exists completely outside of the CS-Direct environment and you do not need to preserve any of the CS-Direct variables (`c`, `cid`, `ct`, `p`) other than `rendermode` when creating a URL for it.

This tag completes the appropriate URL encoding required for all the arguments passed to it—with the exception of arguments passed in as `PACKEDARGS`, which are assumed to be already URL-encoded. (The `RENDER.PACKARGS` tag URL-encodes arguments passed to it, which are the arguments you pass in to `RENDER.ENCODE` as `PACKEDARGS`.)

This tag also includes session IDs in the URL, according to the conventions of the local application server, if a browser is set to reject cookies.

errno

The possible values of `errno` include:

Value	Description
-1004	An argument is missing.

Example

This code uses `RENDER.PACKARGS` to URL-encode the name of the site as a string and then uses `RENDER.ENCODE` to create a URL that includes the previously encoded string:

```
<RENDER.PACKARGS OUTSTR="myPackedArgs"
  ARGS_assetID="Variables.oid"/>
<RENDER.ENCODE STR="ContentServer"
  PACKEDARGS="Variables.myPackedArgs"
  OUTSTR= "myURL"/>
```


See Also

[RENDER.PACKARGS](#)

RENDER.EXPORT

This tag begins the export process. It starts with the specified page and recursively exports all links (both pages and blobs).

Syntax

```
<RENDER.EXPORT
  PAGENAME="page_name"
  ROOTFOLDER="root_directory"
  VIRTUALFOLDER="URL_prefix"
  cid="asset_ID"
  [c="assettype"]
  [p="parent_page_ID"]
  [PACKEDARGS="stringFromPACKARGStag"]
  [ARGS_var1="value"] />
```

Parameters

PAGENAME (required)

Name of the page to export.

ROOTFOLDER (required)

Export root directory. Its value equals the base directory (*cs.pgexportfolder*) plus optional subdirectory (DIR from publish destination arguments).

VIRTUALFOLDER (required)

URL prefix to add to the beginning of all assets' URLs contained in exported files.

cid (required)

Asset ID.

c (optional)

Asset type.

p (optional)

Asset ID of the parent Page asset.

PACKEDARGS (optional)

Contains already URL-encoded packed arguments in standard form, with ampersands (&) separating the arguments, and equals signs (=) separating the argument name from value.

ARGS_XXX

A family of generic arguments. The argument name is XXX and the value is the attribute's value. These arguments are passed to the specified page.

errno

The possible values of `errno` include:

Value	Description
-1004	An argument is missing.

RENDER.FILTER

Filters single assets or lists of assets which have not been approved for publishing from being exported into the generated file.

Syntax

```
<RENDER.FILTER
  LIST="list name"
  LISTVARNAME="output list name"
  LISTIDCOL="assetID column"
  [LISTTYPECOL="assettype column"
  [TYPE="asset type"]
  [ID="asset id"]
  [VARNAME="output variable"/>
```

Parameters

LIST (required)

Name of the list to filter.

LISTVARNAME (required)

Name of the list that contains the filtered assets.

LISTIDCOL (required)

Name of the asset ID column in the list to be filtered.

TYPE (optional)

Asset type to filter.

ID (optional)

Asset ID to filter.

VARNAME (optional)

Name of the output variable that includes a true or false value. A true value indicates it is safe to include the asset.

Description

This tag filters a single asset or list of assets based on whether the asset is approved for the destination being exported. Use this tag within templates that generate a list of assets, including unapproved assets. Templates used during during an export publish operation which employs query assets, [EXECSQL](#) commands, [ASSET.LIST](#), and [SELECTTO](#) may be candidates for the `RENDER.FILTER` tag.

errno

The possible values of errno include:

Value	Description
-10004	An argument is missing.

Example

```
<RENDER.FILTER LIST="ArticlesFromTheQuery"
    LISTVARNAME="ArticlesFromTheQuery"
    LISTCOLID="id"/>
<if COND="ArticlesFromTheQuery.#numRows!=0">
<then>
    <LOOP LIST="ArticlesFromTheQuery">
        <CALLELEMENT NAME="OpenMarket/Xcelerate/Actions/Publish/
GetPageURL">
            <ARGUMENT NAME="pagename" VALUE="BurlingtonFinancial/
Article/Variables.ct"/>
            <ARGUMENT NAME="cid" VALUE="ArticlesFromTheQuery.id"/>
            <ARGUMENT NAME="ctype" VALUE="Article"/>
        </CALLELEMENT>
        <A class="wirelink" HREF="Variables.referURL"
REPLACEALL="Variables.referURL"><csvar
NAME="ArticlesFromTheQuery.subheadline"/></A><P/>
    </LOOP>
</then>
</if>
```

Release Introduced

This tag was introduced in CSEE release 4.0.

RENDER.GETBLOBURL

This tag creates a BlobServer URL without embedding it in an HTML tag (see also [RENDER.SATELLITEBLOB](#)).

Syntax

```
<RENDER.GETBLOBURL
    OUTSTR="VariableName"
    BLOBTABLE="blobTable"
    BLOBKEY="primaryKeyName"
    BLOBWHERE="primaryKeyValue"
    BLOBCOL="column name"
    [BLOBHEADER="MIMEtype"]
    [BLOBNOCACHE="false"]
    [ADDSSESSION="true"]
    [DYNAMIC="true"]
    [PACKEDARGS="arguments"]
    [BLOBLHEADERNAMEN="blob header name"]
    [BLOBHEADERVALUEN="blob header value"]
    [CSBLOBID="session variable value"]/>
```

Parameters

OUTSTR (required)

Input. Name of the variable that stores the URL generated by this tag.

BLOBTABLE (required)

Input. Name of the CS-Direct table that stores assets of this type. For example, BLOBTABLE="ImageFile".

BLOBKEY (required)

Input. Name of the column used as the primary key. Typically, this is id.

BLOBWHERE (required)

Input. Object ID of the asset.

BLOBCOL (required)

Input. Name of the column that contains the binary data. For example, for the imagefile asset type, BLOBCOL="urlpicture".

BLOBHEADER (optional)

Input. Mimetype for returned data, in the form *description/extension*. For example, the mimetype for a GIF image is image/gif.

BLOBNOCACHE (optional)

Input. Whether to disable blob caching. true prevents the blob from being cached.

ADDSSESSION (optional)

Input. Whether or not to include session IDs in the URL when a browser is set to reject cookies. true means to encode session IDs in the URL. If this parameter is not specified, it is set to true by default.

DYNAMIC (optional)

Input. Whether to create a static or a dynamic URL for the asset when the page is being rendered by the Export to Disk publishing method. DYNAMIC=true means to create a dynamic URL even if rendermode is set to export. DYNAMIC=false means to create a static URL if rendermode is set to export. The default value is false.

PACKEDARGS (optional)

URL-encoded packed arguments in standard form, separated by ampersands (&) and equal signs (=) separating the argument name from the value.

BLOBHEADERNAME (optional)

This parameter sets HTTP header variables to suit your needs. The BLOBHEADERNAME parameter is used in conjunction with the BLOBHEADERVALUE parameter to produce name/value pairs. This parameter represents the variable name.

BLOBHEADERVALUE (optional)

This parameter is used in conjunction with BLOBHEADERNAME to specify and represent HTTP header variable values.

CSBLOBID (optional)

The value of this parameter must correspond to a session variable of the same name when BlobServer security is on. When a request is made to the BlobServer, the img src service parameter sends a new request to the application server after the browser reads the containing page. BlobServer then verifies if the csblobid parameter was set. When BlobServer security is set to true and the csblobid value matches the session variable value, the blob is served. If the values do not match, a security violation is reported and no blob is served.

Description

This tag creates a BlobServer URL. If rendermode is set to export, the tag creates a static URL (that is, a file name for an HTML file). It creates the URL based on the parameters described in the following section. These are a set of arguments that specify the table the blob is stored in, the name of the column that contains the binary data, the name of the primary key of the table, the value in that column to use to identify the object, and so on.

If rendermode is set to `live`, the tag creates a dynamic URL by concatenating the following:

- The name of the server (which is the same server as the one serving the page that is requesting the URL).
- The approot (application root) of the application server software, which it obtains from the `ft.approot` property in the `futuretense.ini` file.
- The CGI path, which it obtains from the `ft.cgipath` property in the `futuretense.ini` file. For example, for WebLogic this path is `/servlet/` and for iAS6, this path is `/NASapp/cs/`.
- The string `BlobServer?blobtable=`
- The name of the table the blob is stored in, passed in through a parameter that is described in the following section.
- Other arguments, passed in as parameters that are described in the following section.

Based on this information, `RENDER.GETBLOBURL` creates a URL for the blob asset and returns it to the element that requested it in a variable specified by the `OUTSTR` parameter. The requesting element can then use the variable in an HTML tag to create the link.

To create BlobServer URL embedded in an HTML tag, use the [RENDER.SATELLITEBLOB](#) tag.

You can use `BLOBHEADERNAME` and `BLOBHEADERVERVALUE` to specify http headers in the BlobServer response. For example, you can specify the content-type as follows:

```
BLOBHEADERNAME1="Content-Type" BLOBHEADERVERVALUE1="image/gif"
```

If you want to use http headers to prevent browser caching of BlobServer requests for all browsers, use both HTTP 1.0 and HTTP 1.1 cache control directives. In the following example, the first header (`Cache-Control:no-cache`) is the HTTP 1.1 directive and the second header (`Pragma:no-cache`) is the HTTP 1.0 directive.

```
BLOBHEADERNAME1="Cache-Control" BLOBHEADERVERVALUE1="no-cache"
BLOBHEADERNAME2="Pragma" BLOBHEADERVERVALUE2="no-cache"
```

errno

None.

Example

This code creates a BlobServer URL for an ImageFile asset identified through an ID passed in with `BLOBWHERE="Variables.asset:id"` and stores it in a variable named `"theURL"`:

```
<RENDER.GETBLOBURL BLOBTABLE="ImageFile" BLOBCOL="urlpicture"
BLOBHEADER="Variables.asset:mimetype" BLOBKEY="id"
BLOBWHERE="Variables.asset:id" OUTSTR="theURL"/>
```

Release Introduced

This tag was introduced in CSEE release 4.0.

See Also

[RENDER.PACKARGS](#)
[RENDER.SATELLITEBLOB](#)

RENDER.GETPAGEURL

Creates a URL for an asset, processing the arguments passed to it from the calling element into a URL-encoded string and returning it in a variable.

Syntax

```
<RENDER.GETPAGEURL
  OUTSTR="theURLVariable"
  PAGENAME="SiteCatalogPageEntry"
  cid="Assets_ID"
  [p="NameOfParentPage"]
  [c="AssetType"]
  [ADDESSION="true/false"]
  [DYNAMIC="true"]
  [PACKEDARGS="stringFromPACKARGStag"]
  [ARGS_xxx="y"]
  [EXPORT ="true/false"]/>
```

Parameters

OUTSTR (required)

Input. Name of the variable that stores the URL generated by this tag.

PAGENAME (required)

Input. A page name from a SiteCatalog entry. The page name can be referred to by a specific name or it can be determined through some combination of the four standard CS-Direct page criteria variables (c, cid, ct, and p). For example:

PAGENAME="BurlingtonFinancial/Article/Variables.ct"

cid (required)

Input. ID of the asset that the URL is for.

p (optional)

Input. ID of the parent page. If a value for p is not explicitly provided, the tag tries to determine whether a value exists.

c (optional)

Input. Type of the asset. This parameter is optional but if c (asset type) is not supplied, and rendermode is set to `export`, and assets of this type have a **Filename** field, then the file name that the Export to Disk publishing method creates for the asset does not include the information from the **Filename** field of the asset. In other words, it is highly recommended that you supply a value for c.

ADDESSION (optional)

Input. Whether to include session IDs in the URL when a browser is set to reject cookies. `true` means to encode session IDs in the URL. `false` means not to. The default value is `true`.

DYNAMIC (optional)

Input. Whether to create a static or a dynamic URL for the asset when the page is being rendered by the Export to Disk publishing method. `DYNAMIC="true"` means to create a dynamic URL even if `rendermode` is set to `export`. `DYNAMIC="false"` means to create a static URL if `rendermode` is set to `export`. If this parameter is not specified, it is set to `"false"` by default.

PACKEDARGS (optional)

Input. Previously created URL-encoded packed arguments, in standard form; that is, the output of a previous `RENDER.PACKARGS` tag.

ARGS_ (optional)

Input. Name/value pairs to be passed in and included in the URL, with the string `ARGS_` as the prefix for the variable name.

EXPORT (optional)

Specifies whether or not the URL reference is exported. The default value is `false`.

Description

This tag creates URLs for assets that are not blobs.

If `rendermode` is set to `export`, the tag creates a static URL (that is, a file name for an HTML file) and returns it as a variable. It creates the URL based on the standard CS-Direct page criteria variables `c`, `cid`, and `p`, a page name from a SiteCatalog entry, and other parameters, which are described in the following section.

If `rendermode` is set to `live`, the tag creates a dynamic URL by concatenating and encoding the following:

- The name of the server (which is the same server as the one serving the page that is requesting the URL).
- The approot of the application server software, which it obtains from the `ft.approot` property in the `futuretense.ini` file.
- The CGI path, which it obtains from the `ft.cgipath` property in the `futuretense.ini` file. For example, for WebLogic this path is `/servlet/` and for iAS6, this path is `/NASapp/cs/`.
- The string `ContentServer?pagename=`
- A page name from a SiteCatalog entry, passed in as a parameter (described in the following section).
- Information passed in with parameters for the the standard CS-Direct page criteria variables, `c`, `cid`, and `p`, (described in the following section).

Based on this information, the tag creates a URL for the asset and returns it to the element that requested it in the variable specified by the `OUTSTR` parameter. The requesting element can then use the variable to create a hyperlink for the asset.

Note

You use `RENDER.GETPAGEURL` to create URLs for CS-Direct assets. You cannot use it to create links to other sites, HTML files that are created outside of the Export publishing process, and so on.

errno

None.

Example

This code creates a URL for an article asset and stores it in a variable named theURL:

```
<RENDER.GETPAGEURL PAGENAME="BurlingtonFinancial/Article/Full"
cid="Variables.cid" c="Article" OUTSTR="theURL"/>
```

See Also

[RENDER.PACKARGS](#)
[RENDER.GETBLOBURL](#)

RENDER.LOGDEP

Designates asset dependencies during the approval process.

Syntax

```
<RENDER.LOGDEP
    [ASSET="asset name"]
    [CID="asset id"]
    [C="asset type"]
    [DEPTYPE="exact|exists|greater"]/>
```

Parameters

ASSET (optional)

Name of the asset. Either ASSET or CID and C must be set.

CID (optional)

ID of the asset. When using the CID parameter you must also specify the C parameter. Either ASSET or CID and C must be set.

C (optional)

Type of asset. This parameter must be used in conjunction with the CID parameter. Either ASSET or CID and C must be set.

DEPTYPE (optional)

Legal values are "exact", "exists" or "greater". This attribute specifies whether approval dependency is "exact", "exists" or "greater".

- **Exact** specifies a version dependency. The version of the dependent asset must be equal to the version of the asset when it was approved.
- **Exists** specifies that any version of the asset satisfies the dependency condition.
- **Greater** specifies the asset must be at a or above a certain version. Description

Description

This tag keeps track of asset dependencies during publishing. When the `RENDER.LOGDEP` tag is encountered during an export publishing operation it marks assets as dependent on the asset being approved. Use the `RENDER.LOGDEP` tag when methods other than `ASSET.LOAD` are employed to insert data into templates.

When a template using this tag is encountered during the rendering phase of a preview or live operation, the `CacheManager` is notified to track any changes.

errno

The possible values of errno include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-10006	The object ID is not valid.

Example

```
<ASSET.LIST LIST="newsArticles"
    FIELD="category"
    VALUE="n"/>
<RENDER.FILTER LIST="newsArticles"
    LISTVARNAME="filteredlist"
    LISTIDCOL="id"/>
<LOOP LIST="filteredlist"
    <RENDER.LOGDEP C="Article"
        CID="FILTEREDLISTID"/>
</LOOP>
```

RENDER.PACKARGS

Encodes in the URL the name/value argument pairs passed in and writes them to an output string, in standard form, with ampersand characters (&) separating each name/value pair.

Syntax

```
<RENDER.PACKARGS
    OUTSTR="myPackedArguments"
    [ARGS_arg1="horse"]
    [PACKEDARGS="cat=shadow&dog=darby"]/>
```

Parameters

OUTSTR (required)

Input. The name of a variable to create that stores the encoded output string.

ARGS_ (optional)

Input. Name/value pairs to be packed by the tag, with the string **ARGS_** as the prefix for the variable name. The names and values of these arguments are URL- encoded and incorporated into the final URL.

An argument passed in with this parameter takes precedence over an argument passed in with the **PACKEDARGS** parameter. If the same argument is passed in with both, the **PACKEDARGS** version is overridden by the **ARGS_** version.

PACKEDARGS (optional)

Input. Previously created URL-encoded packed arguments, in standard form; that is, the output of a previous **RENDER.PACKARGS** tag.

Description

The GetPageURL element encodes (“packs”) the four standard CS-Direct arguments (c, cid, ct, p) before creating a dynamic URL with them. If you need to include additional arguments in a URL, use this tag to URL-encode them (“pack” them) before passing them to the GetPageURL element.

You use this tag to pack up the arguments and store them as a string. Then, you pass that string to the GetPageURL element through the PACKEDARGS parameter of a CALLELEMENT tag.

errno

The possible values of errno include:

Value	Description
-10004	An argument is missing.

Example

This code packs a name/value pair with restricted characters into a URL-encoded string, calls the GetPageURL element, and passes in several arguments, including the URL-encoded (packed) one:

```
<RENDER.PACKARGS OUTSTR="myPackedArgs" ARGS_location="Brazil/
Portugal"/>
<CALLELEMENT NAME="OpenMarket/Xcelerate/Actions/Publish/
GetPageURL"/>
  <ARGUMENT NAME="PACKEDARGS" VALUE="myPackedArgs"/>
  <ARGUMENT NAME="ctype" VALUE="Article"/>
  <ARGUMENT NAME="cid" VALUE="Variables.cid"/>
</CALLELEMENT>
```

See Also

[RENDER.ENCODE](#)

RENDER.SATELLITEBLOB

Creates a URL in a complete HTML tag that retrieves a blob from a table or from the CS-Satellite cache.

Syntax

```
<RENDER.SATELLITEBLOB
  SERVICE="HTML tagName"
  BLOBTABLE="blobTable"
  BLOBKEY="primaryKeyName"
  BLOBWHERE="primaryKeyValue"
  BLOBCOL="column name"
  BLOBHEADER="MIMEtype"
  [ARGS_format1="5"]
  [CACHECONTROL="expirationDateAndTime"
  [BLOBLHEADERNAMEN="blob header name"]
  [BLOBHEADERVALUEN="blob header value"]
  [CSBLOBID="session variable value" />
```

Parameters

SERVICE (required)

Input. The start of the HTML tag to hold the reference to the blob. For example, if the blob is an image, use SERVICE="IMG SRC".

BLOBTABLE (required)

Input. The name of the CS-Direct table that stores assets of this type. For example, BLOBTABLE="ImageFile".

BLOBKEY (required)

Input. The name of the column used as the primary key. Typically this is "id".

BLOBWHERE (required)

Input. The object ID of the asset.

BLOBCOL (required)

Input. The name of the column that contains the binary data. For example, for the imagefile asset type, BLOBCOL="urlpicture".

BLOBHEADER (required)

The mimetype for returned data, in the form *description/extension*. For example, the mimetype for a GIF image is image/gif.

ARGS_ (optional)

Input. Name/value modifiers for the HTML tag, with the string ARGS_ as the prefix. For example, ARGS_vspace="5", ARGS_hspace="5", and so on.

CACHECONTROL (optional)

Input. Determines for how long to cache the blob. See [satellite.blob](#) for details about this parameter.

BLOBLHEADERNAMEN (optional)

This parameter set specific HTTP header variables to suit your needs, The BLOBLHEADERNAME parameter is used in conjunction with the BLOBHEADERVALUE parameter to produce name/value pairs. This parameter represents the variable name.

BLOBHEADERVALUE*N* (optional)

This parameter is used in conjunction with BLOBHEADERNAME to specify and represent HTTP header variable values.

CSBLOBID (optional)

The value of this parameter must correspond to a session variable of the same name when BlobServer security is on. When a request is made to the BlobServer, the `img src` service parameter sends a new request to the application server after the browser reads the containing page. BlobServer then verifies if the `csblobid` parameter was set. When BlobServer security is set to true and the `csblobid` value matches the session variable value, the blob is served. If the values do not match, a security violation is reported and no blob is served.

Description

This tag is the CS-Direct equivalent of the CS-Satellite SATELLITE.BLOB tag, with the following exception:

- `rendermode` is passed in automatically.

You can use BLOBHEADERNAME and BLOBHEADERVALUE to specify http headers in the BlobServer response. For example, you can specify the content-type as follows:

```
BLOBHEADERNAME1="Content-Type" BLOBHEADERVALUE1="image/gif"
```

If you want to use http headers to prevent browser caching of BlobServer requests for all browsers, use both HTTP 1.0 and HTTP 1.0 cache control directives. In the following example, the first header (Cache-Control:no-cache) is the HTTP 1.1 directive and the second header (Pragma:no-cache) is the HTTP 1.0 directive.

```
BLOBHEADERNAME1="Cache-Control" BLOBHEADERVALUE1="no-cache"
```

```
BLOBHEADERNAME2="Pragma" BLOBHEADERVALUE2="no-cache"
```

To determine how long to cache a page or pagelet, these criteria are evaluated in this order:

1. A value passed in using the `CACHECONTROL` parameter.
2. If there is no value in `CACHECONTROL`, the value in the `satellite.page.cachecontrol.default` property, found in the `futuretense.ini` property file, is used.
3. If there is no value in `satellite.page.cachecontrol.default`, the value of the `expiration` property, located in the `satellite.ini` property file, is used.

FatWire recommends that you set values for the `satellite.page.cachecontrol.default` properties that make sense for the majority of your pages and pagelets. If you want to override the default expiration time, use the `CACHECONTROL` parameter. Use the `expiration` property to determine the timeout for pages that are not generated from this tag; for example, pages that are generated directly from a URL.

errno

None.

Example

This code creates an HTML `<IMG SRC>` tag. The SRC is the blob in the ImageFile table identified through the ID passed in with `BLOBWHERE="Variables.asset:id"` and both its horizontal and vertical spacing are at five pixels:

```
<RENDER.SATELLITEBLOB BLOBTABLE="ImageFile" BLOBKEY="id"
BLOBCOL="urlpicture" BLOBWHERE="Variables.asset:id" BLOBHEADER=
"Variables.asset:mimetype" SERVICE="IMG SRC" ARGS_alt=
"Variables.asset:alttext" ARGS_hspace="5" ARGS_vspace="5" />
```

See Also

[RENDER.SATELLITEPAGE](#)
[satellite.blob](#)

RENDER.SATELLITEPAGE

Retrieves a page or pagelet identified through a page name and a set of arguments and specifies its cache expiration.

Syntax

```
<RENDER.SATELLITEPAGE
  PAGENAME="nameOfPageEntry"
  [CACHECONTROL="expiration_date_and_time"]
  [ARGS_var1="value1"]/>
```

Parameters

PAGENAME (required)

Input. The name of the page to invoke.

CACHECONTROL (optional)

Input. Determines for how long to cache the pagelet. See [satellite.page](#) for details about this parameter.

ARGS_ (optional)

Input. Name/value pairs to be passed to the page entry, with the string `ARGS_` as the prefix for the variable name. For example, `ARGS_c="article"`, `ARGS_cid="Variables.oid"`, and so on. Page criteria for the page (c, cid, etc.) must also be identified in the page entry in the SiteCatalog table.

Description

This tag is the CS-Direct equivalent of the `SATELLITE.PAGE` tag with the following exceptions:

- You use CS-Direct syntax for passing it variables. That is, rather than passing them with an argument statement (`<ARGUMENT NAME= "variableName" VALUE="variableValue"/>`) you pass them with the `ARGS_` parameter (`ARGS_variableName="variableValue"`).
- `rendermode` is passed to the element automatically.

Use this tag to serve a CS-Direct page or pagelet when you are using CS-Satellite on your system. The page or pagelet must have a page entry in the SiteCatalog. If the specified

pagelet is already cached on a CS-Satellite, this tag loads it. If the specified pagelet is not already cached, this tag caches it and loads it.

If an element is coded with a `RENDER.SATTLELITEPAGE` tag but CS-Satellite is not present, `RENDER.SATELLITEPAGE` passes the call to `RENDER.CONTENTSERVER` which then serves the page or pagelet.

To determine how long to cache a page or pagelet, these criteria are evaluated in this order:

1. A value passed in using the `CACHECONTROL` parameter.
2. If there is no value in `CACHECONTROL`, the value in the `satellite.page.cachecontrol.default` property, found in the `futuretense.ini` property file, is used.
3. If there is no value in `satellite.page.cachecontrol.default`, the value of the `expiration` property, located in the `satellite.ini` property file, is used.

FatWire recommends that you set values for the `satellite.page.cachecontrol.default` properties that make sense for the majority of your pages and pagelets. If you want to override the default expiration time, use the `CACHECONTROL` parameter. Use the expiration property to determine the timeout for pages that are not generated from this tag; for example, pages that are generated directly from a URL.

errno

None.

Example

This code calls for the page entry of a template named `LeadSummary`, passing the ID of the asset that it should display, the ID of the asset's parent page, and the name of a child template:

```
<RENDER.SATELLITEPAGE PAGENAME="BurlingtonFinancial/Article/LeadSummary" ARGS_cid="theArticles.oid" ARGS_ct="Full"
ARGS_p="Variables.p"/>
```

See Also

[RENDER.CONTENTSERVER](#)
[RENDER.SATELLITEBLOB](#)
[satellite.page](#)

RENDER.STREAM

Use this tag as an alternative to [CSVAR](#) to display asset field that may have embedded links or inclusions.

Syntax

```
<RENDER.STREAM
    VALUE="variable_string"
    VARIABLE="variable name"
    SSVARIABLE="session variable name"
    PROPERTY="futuretense.ini_property"
    LIST="list_name"
    COLUMN="column_name" />
```

Parameters

VALUE (optional)

You must use at least one of the following four parameters: VARIABLE, SSVARIABLE, PROPERTY, or LIST and COLUMN (LIST and COLUMN must be used together). Name of the inline string that you want to stream to a page. For example, for a variable called *foo*, set the parameter equal to *Variables.foo*.

VARIABLE (optional)

You must use at least one of the following four parameters: VARIABLE, SSVARIABLE, PROPERTY, or LIST and COLUMN (LIST and COLUMN must be used together). Name of the Content Server variable that you want to stream to a page. For example, for a variable called *foo*, set the parameter equal to *foo* rather than to *Variables.foo*.

SSVARIABLE (optional)

You must use at least one of the following four parameters: VARIABLE, SSVARIABLE, PROPERTY, or LIST and COLUMN (LIST and COLUMN must be used together). Name of the Content Server session variable that you want to stream to a page.

PROPERTY (optional)

You must use at least one of the following four parameters: VARIABLE, SSVARIABLE, PROPERTY, or LIST and COLUMN (LIST and COLUMN must be used together). A Content Server property from the futuretense.ini property file.

LIST (optional)

You must use at least one of the following four parameters: VARIABLE, SSVARIABLE, PROPERTY, or LIST and COLUMN (LIST and COLUMN must be used together). Name of the list

COLUMN (optional)

You must use at least one of the following four parameters: VARIABLE, SSVARIABLE, PROPERTY, or LIST and COLUMN (LIST and COLUMN must be used together). Name of the field in the list.

RENDER.UNKOWNDEPS

Describes the current asset dependencies as incomplete.

Syntax

```
<RENDER . UNKOWNDEPS />
```

Parameters

None.

Description

When the `RENDER . UNKOWNDEPS` tag is used during an export publish operation, it marks the asset's dependencies as incomplete. This tag makes every publish operation republish the specified asset (rather than retrieving it from cache) to ensure the data contained in the asset is current. When a template using this tag is employed during a rendering operation on the live mirror site or during a preview operation, it instructs the cache manager not to cache the pagelet.

This tag should be used in cases when pages have undeterminable connections to dependents. For example, specify this tag when using a query in a template which yields a different resultset each time the query is generated.

errno

None.

Release Introduced

This tag was introduced in CSEE release 4.0.

Role List Tags

These tags create and manage Content Server roles. The following section describes each Role List tag, beginning with [ROLELIST.ADD](#).

ROLELIST.ADD

Add a role to the list.

Syntax

```
<ROLELIST.ADD
  NAME="rolelist_object_name"
  ROLE="role_ID" />
```

Parameters

NAME (required)
Name of the role list object.

ROLE (required)
Role ID to add.

errno

The possible values of `errno` include:

Value	Description
-100	Asset Exception
-12001	An expected table or iList field was not found

Example

The following code adds a role to the role list, then verifies whether or not the role was added correctly.

```
<!-- Add Approver role -->
<ROLELIST.ADD NAME="roleobject" ROLE="Approver"/>
<!-- Check for Approver role -->
<ROLELIST.HASROLE NAME="roleobject" ROLE="Approver"
  VARNAME="roleexists2"/>
<IF COND="Variables.roleexists2=true">
  <THEN>
    <BR/><BR/> ROLELIST.ADD: 'Approver' Role is present
    <CSVAR NAME="Variables.roleexists2"/> <BR/><BR/>
  </THEN>
  <ELSE>
    <BR/> ROLELIST.ADD: Approver role is NOT present. Value
    returned is <CSVAR NAME="Variables.roleexists2"/> <BR/><BR/>
  </ELSE>
```

</IF>

ROLELIST.CLEAR

Clears the role list.

Syntax

```
<ROLELIST.CLEAR
    NAME="rolelist_object_name" />
```

Parameters

NAME (required)

Name of the role list object from which to clear the role list.

errno

The possible values of `errno` include:

Value	Description
-12001	An expected table or iList field was not found.
-100	Asset Exception.

Example

The following code clears the current role list, then sets new attribute values for the role list.

```
<ROLELIST.CLEAR NAME="roleobject"/>
<ROLELIST.GETALL NAME="roleobject" VARNAME="roles"/>
<IF COND="IsVariable.roles=false">
<THEN>
    <SETVAR NAME="roles" VALUE="[None]"/>
    <BR/> ROLELIST.CLEAR: No roles found <BR/><BR/>
</THEN>
<ELSE>
    <BR/> ROLELIST.CLEAR: Roles found are <CSVAR
NAME="Variables.roles"/><BR/><BR/>
</ELSE>
</IF>
```

ROLELIST.DELETE

Deletes a role from the list.

Syntax

```
<ROLELIST.DELETE
  NAME="rolelist_object_name"
  ROLE="role_ID" />
```

Parameters

NAME (required)

Name of the role list object from which to delete the role.

ROLE (required)

Role ID to delete.

errno

The possible values of `errno` include:

Value	Description
-100	Asset Exception.
-12001	An expected table or iList field was not found.

Example

The following code deletes a role from a role list, then verifies whether or not the role was deleted.

```
<!-- delete a roll -->
<ROLELIST.DELETE NAME="roleobject" ROLE="GeneralAdmin"/>
<!-- show all roles -->
<ROLELIST.GETALL NAME="roleobject" VARNAME="roles3"/>
<IF COND="IsVariable.roles3=false">
  <THEN>
    <SETVAR NAME="roles3" VALUE="[None]"/>
    <BR/> ROLELIST.DELETE: No roles found <BR/>
  </THEN>
<ELSE>
  <BR/> ROLELIST.DELETE: Roles found are <CSVAR
NAME="Variables.roles3"/><BR/><BR/>
</ELSE>
</IF>
```

ROLELIST.GATHER

Gathers the values from the variable, PREFIX+roles, into a rolist object.

Syntax

```
<ROLELIST.GATHER  
    NAME="rolist_object_name"  
    PREFIX="variable_prefix" />
```

Parameters

NAME (required)

Rolist object.

PREFIX (required)

First part of the name of all variables and/or lists used by this gather operation. Variables are typically named with the prefix together with the field name. If a delimiter, such as a colon, was used in the creation of the variables that you want to gather, you must use the same delimiter in the gather operation. For example, *Variables.prefix:fieldname*.

ROLELIST.GETALL

Gets the list of roles as a comma-separated string.

Syntax

```
<ROLELIST.GETALL  
    NAME="rolist_object_name"  
    VARNAME="variable_name" />
```

Parameters

NAME (required)

Name of the role list object.

VARNAME (required)

Name of the variable to create.

Example

The following code gets all the roles in a role list object as a string, verifies whether or not the object contains any roles, and then displays the roles.

```
<!-- Show user roles -->  
<ROLELIST.GETALL NAME="roleobject" VARNAME="roles"/>  
<IF COND="IsVariable.roles=false">  
<THEN>  
    <SETVAR NAME="roles" VALUE="[None]"/>  
    <BR/> ROLELIST.GETALL: No roles found <BR/>  
</THEN>  
<ELSE>  
    <BR/> ROLELIST.GETALL: Roles found are <CSVAR  
NAME="Variables.roles"/><BR/><BR/>
```

```
</ELSE>
</IF>
```

ROLELIST.HASROLE

Verifies whether a role is present in a role list.

Syntax

```
<ROLELIST.HASROLE
  NAME="rolelist_object_name"
  ROLE="role_to_verify"
  VARNAME="variableName" />
```

Parameters

NAME (required)

Name of the role list object.

ROLE

Name of the role to verify.

VARNAME

Name of the output variable. The value is set to true if the role is in the role list.

Example

The following code checks the role list object to verify whether or not it contains a specified role.

```
<!-- check for GeneralAdmin role -->
<ROLELIST.HASROLE NAME="roleobject" ROLE="GeneralAdmin"
  VARNAME="roleexists"/>
<IF COND="Variables.roleexists=true">
  <THEN>
    <BR/><BR/> ROLELIST.HASROLE: 'GeneralAdmin' Role is present
  <BR/>
    <CSVAR NAME="Variables.roleexists"/>
  </THEN>
<ELSE>
  <BR/> ROLELIST.HASROLE: GeneralAdmin role is NOT present.
  Value returned is <CSVAR NAME="Variables.roleexists"/> <BR/><BR/>

</ELSE>
</IF>
```

ROLELIST.ISALL

Verifies whether the role list corresponds to “any” (or “all”) roles.

Syntax

```
<ROLELIST.ISALL  
    NAME="rolelist_object_name"  
    VARNAME="variable_name" />
```

Parameters

NAME (required)

Name of the rolelist object.

VARNAME (required)

Name of the output variable to create. The value is set to `true` if the object specifies "any".

Description

If the role list for a workflow process is set to “any,” then users with any role (i.e. users from all roles) can participate in that workflow process.

errno

The possible values of `errno` include:

Value	Description
-100	Asset Exception.
-12001	An expected table or iList field was not found.

Example

The following code checks to see if the status of the role is “any” or “all”.

```
<!-- check for is all -->  
<ROLELIST.ISALL NAME="roleobject" VARNAME="status"/>  
ROLELIST.ISALL: <CSVAR NAME="Variables.status"/><BR/>
```

ROLELIST.SCATTER

Writes (or “scatters”) the role list object into the variable, PREFIX+roles.

Syntax

```
<ROLELIST.SCATTER
  NAME="rolelist_object_name"
  PREFIX="variable_prefix" />
```

Parameters

NAME (required)

Name of the role list object.

PREFIX (required)

First part of the name of all variables and/or lists used by this scatter operation.

Variables are typically named with the prefix together with the field name. A delimiter is not automatically inserted after the prefix. Therefore, if you want a delimiter, such as a colon, you must specify it with the prefix. For example, Variables.prefix:fieldname.

ROLELIST.SETALL

Sets the role list to indicate "any" role.

Syntax

```
<ROLELIST.SETALL
  NAME="rolelist_object_name" />
```

Parameters

NAME (required)

Rolelist object name.

Description

If the role list for a workflow process is set to “any,” then users with any role (i.e. users from all roles) can participate in that workflow process.

Example

The following code sets all the roles into the rolelist object, then shows all the roles that have been set.

```
<!-- setall the roles -->
<ROLELIST.SETALL NAME="roleobject"/>
<!-- show all roles -->
<ROLELIST.GETALL NAME="roleobject" VARNAME="roles4"/>
<IF COND="IsVariable.roles4=false">
<THEN>
  <SETVAR NAME="roles4" VALUE="[None]"/>
  <BR/> ROLELIST.SETALL: No roles found. <BR/><BR/>
</THEN>
```

```
<ELSE>  
  <BR/> ROLELIST.SETALL: Roles found are <CSVAR  
  NAME="Variables.roles4"/><BR/><BR/>  
</ELSE>  
</IF>
```


Site List Tags

These tags enable you to manage Content Server sites. The following section describes each Site List tag, beginning with [SITELIST.ADD](#).

SITELIST.ADD

Add a site to the list.

Syntax

```
<SITELIST.ADD  
  NAME="sitelist_object_name"  
  PUBID="pub_ID" />
```

Parameters

NAME (required)
 Name of the site list object.

PUBID (required)
 Site ID to add.

SITELIST.CLEAR

Clears the site list.

Syntax

```
<SITELIST.CLEAR  
  NAME="sitelist_object_name" />
```

Parameters

NAME (required)
 Name of the site list object.

SITELIST.DELETE

Deletes a site from the list.

Syntax

```
<SITELIST.DELETE  
  NAME="sitelist_object_name"  
  PUBID="pub_ID" />
```

Parameters

NAME (required)
 Name of the site list object.

PUBID (required)
Site ID to delete.

SITELIST.GATHER

Gathers a site list object from variables that are names with the convention, PREFIX+roles.

Syntax

```
<SITELIST.GATHER  
  NAME="sitelist_object_name"  
  PREFIX="variable_prefix" />
```

Parameters

NAME (required)

Name of the site list object whose fields you want to gather.

PREFIX (required)

First part of the name of all variables and/or lists used by this gather operation. Variables are typically named with the prefix together with the field name. If a delimiter, such as a colon, was used in the creation of the variables that you want to gather, you must use the same delimiter in the gather operation. For example, Variables.*prefix:fieldname*.

SITELIST.GETALL

Gets the list of sites as a comma-separated string.

Syntax

```
<SITELIST.GETALL  
  NAME="sitelist_object_name"  
  VARNAME="variable_name" />
```

Parameters

NAME (required)

Name of the site list object.

VARNAME (required)

Name of the variable to create.

SITELIST.HASSITE

Verifies whether a site is present in site list.

Syntax

```
<SITELIST.HASSITE  
  NAME="sitelist_object_name"  
  PUBID="pub_ID"  
  VARNAME="variable_name" />
```

Parameters

NAME (required)

Name of the site list object.

PUBID

Site ID to verify.

VARNAME

Name of the output variable to create. The value is set to `true` if the site is found in the site list.

SITELIST.ISALL

Verifies whether the site list is set to be "any" site.

Syntax

```
<SITELIST.ISALL  
  NAME="sitelist_object_name"  
  VARNAME="variableName" />
```

Parameters

NAME (required)

Name of the site list object.

VARNAME

Name of the output variable to create. The value is set to `true` if the object contains all sites. This is equivalent to setting sites to "any" in the Content Server interface.

Description

If the site list for a workflow process is set to "any," then the workflow process is applicable to any site.

SITELIST.SCATTER

Writes (“scatters”) the values in a site list object into variable, PREFIX+sites.

Syntax

```
<SITELIST.SCATTER  
  NAME="sitelist_object_name"  
  PREFIX="variable_prefix" />
```

Parameters

NAME (required)

Name of the site list object.

PREFIX (required)

First part of the name of all variables and/or lists used by this scatter operation.

Variables are typically named with the prefix together with the field name. A delimiter is not automatically inserted after the prefix. Therefore, if you want a delimiter, such as a colon, you must specify it with the prefix. For example,

Variables.*prefix:fieldname*.

Description

This scatter operation sets the variable, PREFIX+sites.

SITELIST.SETALL

Sets the site list to indicate "any" site.

Syntax

```
<SITELIST.SETALL  
  NAME="sitelist_object_name" />
```

Parameters

NAME (required)

Name of the site list object.

Description

The `sitelist.setall` tag sets the site list to “any,” which indicates that a workflow process is applicable to any (or all) sites. A null entry is then registered for the site in the database.

Site Plan Tags

The CS-Direct site tree is an outline or site plan, located on the **Site Plan** tab. You use the SITEPLAN tags to extract (from the CS-Direct database) information about a page asset's position in the site tree hierarchy. You can then display that information on your online site in site maps, navigation bars, and so on.

CS-Direct stores information about page assets in several database tables:

- The Page table, which stores information about the page asset as an asset (its name, its object ID, who created it, which template it uses, and so on). Use the ASSET tags to obtain information from that table.
- The AssetRelationTree table, which stores information about the other assets that page assets have relationships with through named or unnamed associations. Use the ASSET.CHILDREN tag to obtain information from that table.
- The AssetPublication table, which specifies which management sites (publications) give you access to the page. Page assets cannot be shared between sites (publications) so there is only one row entry for a page because it can have only one **pubid**. A pubid is a unique value that identifies a site.
- The SitePlanTree table, which stores information about the page asset's hierarchy in the site structure. Use the SITEPLAN tags to obtain information from that table.

The SitePlanTree table is a standard Content Server object tree with nodes and children. It stores information such as a page asset's node ID, the node ID of its parent, whether it is placed or unplaced, and so on.

If you want to display information about a page asset as an asset, use the ASSET tags to retrieve it. (See [Asset Tags](#).) If you want to display information about a page asset's hierarchy in the site structure, use the SITEPLAN tags to retrieve it. See also .

The following section describes each site plan tag, beginning with [SITEPLAN.CHILDREN](#).

SITEPLAN.CHILDREN

Queries the SitePlanTree table and then builds a list of the child nodes (pages) for the specified node (page).

Syntax

```
<SITEPLAN.CHILDREN NAME="nodeName"
  LIST="listName"
  [OBJECTTYPE="Page"]
  [OBJECTID= "ObjectIDofSpecificChildNode"]
  [CODE="Placed" or "Unplaced"]
  [ORDER="fieldName"]/>
```

Parameters

NAME (required)

Input. Name of the parent node whose children you want to list. This parent node must be loaded and assigned a name before you can pass its name to this tag. (See [SITEPLAN.LOAD](#).)

LIST (required)

Input. The name to assign to the list of the node's children. This list holds values for all of the fields in the SitePlanTree table. If a list with this name already exists, it is replaced.

OBJECTTYPE (optional)

Input. Restricts the list to nodes of a specific type and that type should always be "Page." You can combine this parameter with OBJECTID to request one specific node.

OBJECTID (optional)

Input. The object ID of a specific child node. If you supply an OBJECTID, you must also supply an OBJECTTYPE.

CODE (optional)

Input. The name of the association that describes the relationship between the child node and the parent node. Valid codes are "Placed" and "Unplaced".

ORDER (optional)

Input. The fields to sort the list by and whether the sort on those fields is ascending (asc) or descending (desc). By default, the sort is ascending. If you specify more than one field, separate the field names with a comma.

For example, if you specify ORDER="nrank", the list is sorted by rank starting at number 1. If you want the list sorted by descending rank, use ORDER="nrank, desc".

Description

This tag queries the SitePlanTree table for a list of the child nodes of the node that you specify, listing each child with a value for all of the fields from that table (nid, nparentid, nrank, otype, oid, and ncode). You must load the parent node (SITEPLAN.LOAD) before you can invoke this tag to query for and list its children. The list is a standard Content Server list.

You can restrict the list of children nodes by association name (CODE), object type, object ID, and rank (ORDER).

If you specify OBJECTTYPE="Page", the list of children is a join of the SitePlanTree and the Page table. In other words, you do not need to perform a subsequent [SITEPLAN.LOAD](#) or [ASSET.LOAD](#) (both of which are direct database queries that take time) to obtain information about the child pages. You do need to use more [SITEPLAN.LOAD](#) tags if you want to continue traversing the hierarchy in the SitePlanTree table—that is, if you want to extract grandchildren pages, and so on.

errno

The possible values of errno include:

Value	Description
-111	The asset has no children.
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-10006	The object ID is not valid.
-10007	The version of the object is not valid.

Example

This code extracts the child pages for the page asset identified as “parentNode,” stores them in the list named “PlacedPages,” orders them by rank, and restricts the list to include only placed pages:

```
<SITEPLAN.CHILDREN NAME="parentNode" OBJECTTYPE="Page"
LIST="PlacedPages" CODE="Placed" ORDER="nrank"/>
```

See Also

[SITEPLAN.LOAD](#)

[SITEPLAN.LISTPAGES](#)

SITEPLAN.GET

Retrieves and lists the value of a single field from a loaded site plan node.

Syntax

```
<SITEPLAN.GET
    NAME="nodeName"
    FIELD="fieldName"
    [OUTPUT="variableName"]/>
```

Parameters

NAME (required)

Input. Name of the site plan node whose field value you want to retrieve. This node must be loaded and assigned a name before you can pass its name to this tag. (See [SITEPLAN.LOAD](#).)

FIELD (required)

Input. Name of the field whose value you want to retrieve.

OUTPUT (optional)

Input. Name to assign to the output variable that stores the value of the field. If you do not supply a name, CS-Direct creates a variable with the same name as that of the field.

Description

This tag obtains the value (or contents) from the specified field of a loaded node, and stores it in the specified output variable.

Use this tag to extract the parent node ID in order to trace the parentage of a page, which enables you to then display a link to the parent of the page that is currently displayed.

errno

The possible values of errno include:

Value	Description
-10004	A required parameter is missing.

Value	Description
-10005	The requested object is not in the object pool (is not loaded into memory).

Example

This code extracts the value of the “oid” field for the site plan node identified as “pageNode” and stores it in an output variable named “objectid.”

```
<SITEPLAN.GET NAME="pageNode" FIELD="oid" OUTPUT="objectid"/>
```

See Also

[SITEPLAN.LOAD](#)

SITEPLAN.GETPAGENAME

Retrieves the page asset name of the specified node. (Note that this is an asset name, not a page name from the SiteCatalog table.)

Syntax

```
<SITEPLAN.GETPAGENAME
    NAME="nodeName"
    OUTPUT="variableName" />
```

Parameters

NAME (required)

Input. The name assigned to the site plan node whose page name you want to retrieve. This node must be loaded and assigned a name before you can pass its name to this tag. (See [SITEPLAN.LOAD](#).)

OUTPUT (required)

Input. The name to assign to the output variable that stores the name of the page.

Description

This tag returns the page name of the specified node; that is, the value entered into the **Name** field on the CS-Direct **New/Edit Page** form for the page asset that corresponds to this node.

A possible use for this tag is to obtain the page name of the node so that you can display that name in a navigation bar.

errno

The possible values of `errno` include:

Value	Description
-108	Tree selector error.
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-10006	The object ID is not valid.
-10007	The version of the object is not valid.
-10008	The node ID of the object is not valid.
-10009	More than one object met the specified criteria.

Example

This code loads the site plan and then retrieves the page asset name of a specific site plan node:

```
<SITEPLAN.LOAD NAME="BusinessNewsNode" ID="Variables.nodeid"/>
<SITEPLAN.GETPAGENAME NAME="BusinessNewsNode"
OUTPUT="PageAssetName"/>
```

See Also

[SITEPLAN.GET](#)

SITEPLAN.LISTPAGES

Queries the SitePlanTree table and then creates a list of pages, starting with the page node that you specify.

Syntax

```
<SITEPLAN.LISTPAGES
    NAME="nodeName"
    PLACEDLIST="PlacedPages"
    [LEVEL="1"]/>
```

Parameters

NAME (required)

Input. The name assigned to the site plan node whose unplaced or placed pages you want to list. This node must be loaded and assigned a name before you can pass its name to this tag. (See [SITEPLAN.LOAD](#).)

PLACEDLIST (required)

Input. The name to assign to the list of placed pages.

LEVEL (optional)

Input. The number of node levels to list, starting with the node identified by the NAME parameter. The default is 10000.

Description

This tag creates a hierarchical list of the pages in the site plan. You can determine which page node to start with and how many levels in the hierarchy to list.

The list contains the following column names:

- **PageName** – Name of the page.
- **ParentName** – Name of the parent in the tree.
- **NodeLevel** – Level in the tree relative to the parent.
- **PlacedValue** – Specifies if the page is placed or unplaced.
- **AssetType** – Type of asset.
- **Id** – Asset's ID.
- **HasChildren** – Specifies if the asset has children, possible values are true or false.
- **Nid** – Node ID of SitePlan tree entry.

errno

The possible values for `errno` include:

Value	Description
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-10006	The object ID is not valid.
-10007	The version of the object is not valid.
-10008	The node ID of the object is not valid.
-10009	More than one object met the specified criteria.

Example

This code lists all the placed pages under the site plan node identified as “thePubNode” to one level in the tree; that is, it lists the child pages for “thePubNode” but not its grandchildren, or great-grandchildren, and so on.

```
<SITEPLAN.LISTPAGES NAME="thePubNode"
    PLACEDLIST="placedPages"
    LEVEL="1" />
```

See Also

[COPYLIST](#)
[FLUSH](#)
[GOTOROW](#)
[ISINLIST](#)
[LOOP](#)
[RENAMELIST](#)
[SETROW](#)
[STRINGLIST](#)

SITEPLAN.LOAD

Queries the database for the specified site plan tree node and then loads an instance of the site plan into memory as an object.

Syntax

```
<SITEPLAN.LOAD
    NAME="name"
    NODEID="nodeID" />
```

Parameters

NAME (required)

Input. Name of top-level node of the site plan to load.

NODEID (required)

Input. Node ID of the top-level node in the site plan.

Description

This tag executes a database query to retrieve an instance of the specified site plan tree node and then saves the instance into memory (session information), assigning it the name provided by the NAME parameter. That object is then available until the session is flushed or the name is overwritten.

To determine the nodeid of the site plan tree node that you want to load, you can either use the [ASSET.GETSITENODE](#), or [SITEPLAN.LISTPAGES](#) tags.

You must load the site plan before you can use any of the other SITEPLAN tags, except [SITEPLAN.ROOT](#).

errno

The possible values of errno include:

Value	Description
-108	Tree selector error.
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-10006	The object ID is not valid.
-10007	The version of the object is not valid.
-10008	The node ID of the object is not valid.
-10009	More than one object met the specified criteria.

Example

This code loads a page asset, extracts the identity of its site node, and then loads the site plan with that page asset as the top-level page:

```
<ASSET.LOAD NAME="BusinessNews" TYPE="Page"
OBJECTID="Variables.cid"/>
<ASSET.GETSITENODE NAME="BusinessNews" OUTPUT="PageNodeId"/>
<SITEPLAN.LOAD NAME="ParentNode" NODEID="Variables.PageNodeId"/>
```

See Also

[ASSET.LOAD](#)

SITEPLAN.NODEPATH

Returns a list of the loaded site node's ancestors in the SitePlanTree.

Syntax

```
<SITEPLAN.NODEPATH
  NAME="node"
  LIST="list name"/>
```

Parameters

NAME (required)
Name of the node whose children you want to get.

LIST (required)
Name of the list to put the results in.

Description

Returns a list of the loaded site node's ancestors in the SitePlanTree. The list has all the columns of the SitePlanTree and does not include the current object. The first item in the list is its immediate parent, the second is that node's parent, and so on.

errno

The possible values of errno include:.

Value	Description
-10004	A required parameter is missing.

Example

To get the top most ancestor page of a page asset:

```
<ASSET.LOAD TYPE="Page" NAME="apage" FIELD="name"
VALUE="Variables.page"/>
<ASSET.GETSITENODE NAME="apage" OUTPUT="PageNodeId"/>

<SITEPLAN.LOAD NAME="node" NODEID="Variables.PageNodeId"/>
<SITEPLAN.NODEPATH NAME="node" LIST="myAncestors"/>
<!-- display the top level page, just under the Site -->
<GOTOROW LIST="myAncestors" ROWNUM="myAncestors.#numRows"/>
<SETROW LIST="myAncestors" ACTION="PREV"/>
<if COND="myAncestors.otype=Page">
<then>
  <ASSET.LOAD NAME="ancestor" TYPE="myAncestors.otype"
OBJECTID="myAncestors.oid" DEPTYPE="exists"/>
  <ASSET.GET NAME="ancestor" FIELD="name"/>
  <csvar NAME="Variables.name"/>
</then>
</if>
```

Release Introduced

This tag was introduced in CSEE release 5.0.

SITEPLAN.PLACE

Marks the page as Placed and assigns it a place in the page hierarchy.

Syntax

```
<SITEPLAN.PLACE
  NAME="node"
  [NODEID="page node ID"]
  [CHILDDID="page ID"]
  [TYPE="asset type"]
  [RANK="page rank"] />
```

Parameters

NAME (required)

Name of the node under which to place a page.

NODEID (optional)

Node ID of the page to place under named page. You can specify either NODEID, or CHILDDID and TYPE.

CHILDDID (optional)

Used when the TYPE parameter is specified. With TYPE, this parameter identifies the page asset to place under the loaded Site Plan node.

TYPE (optional)

Used when the CHILDDID is specified. With CHILDDID, this parameter identifies the page asset to place under the loaded Site Plan node.

RANK (optional)

Specifies the rank under the parent.

errno

The possible values of errno include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-10007	Rank is not specified.

SITEPLAN.ROOT

Queries the SitePlanTree table for a root node of the site plan.

Syntax

```
<SITEPLAN.ROOT
  LIST="rootNode"
  [OBJECTID="site ID"]/>
```

Parameters

LIST (required)

Name to assign to the list that stores the root node. The columns of the list match those from the SitePlanTree table. If a list with this name already exists, it is replaced with this new list.

OBJECTID (optional)

Site (formerly publication) ID.

Description

You use this tag to extract and display information about the site structure from the top down.

If you use the **OBJECTTYPE** parameter, the result is a join of the SitePlanTree and the Publication table (only a site can be a root). The name and description fields in the Publication table are then accessible without requiring a subsequent [SITEPLAN.LOAD](#) for the site plan.

errno

The possible values of errno include:

Value	Description
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing.
-10006	The object ID is not valid.
-10007	The version of the object is not valid.
-10008	The node ID of the object is not valid.

Example

This code retrieves the root node identified by the object ID and stores it in a list named "PubRoot":

```
<SITEPLAN.ROOT LIST="PubRoot" OBJECTID="968251170475"/>
```

SITEPLAN.SETOBJECT

Sets an object in a node.

Syntax

```
<SITEPLAN.SETOBJECT  
    NAME="node name"  
    OBJECT="page asset object"/>
```

Parameters

NAME (required)
Name of a loaded Site Plan node.

OBJECT (required)
Name of a loaded page asset.

Description

The `SITEPLAN.SETOBJECT` tag sets an object in a node. The `OTYPE` and `OID` fields of the `SitePlanTree` row are updated.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

SITEPLAN.SETPARENT

Sets the parent of a node.

Syntax

```
<SITEPLAN.SETPARENT  
    NAME="node"  
    PARENT="loaded node"/>
```

Parameters

NAME (required)
Name of a loaded Site Plan node.

PARENT (required)
Name of a loaded node of the Site Plan node.

Description

The SITEPLAN.SETPARENT tag returns a list of the loaded site node's ancestors in the SitePlanTree. The NPARENTID field of the SitePlanTree row is updated.

errno

The possible values of errno include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

SITEPLAN.UNPLACE

Removes a Site Plan node from the P laced part of the tree and sets the NCODE field to UnPlaced.

Syntax

```
<SITEPLAN.UNPLACE
    NAME="node name"
    NODEID="child page nid" />
```

Parameters

NAME (required)

Name of a loaded Site Plan node.

NODEID (required)

NID (node ID) of the child page to be removed from P laced.

errno

The possible values of errno include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

SITEPLAN.UNPLACEDANCESTOR

Returns whether the specified node has any pages above it in the hierarchy that have an Unplaced ncode.

Syntax

```
<SITEPLAN.UNPLACEDANCESTOR  
    NAME="node name"  
    OUTPUT="variableName" />
```

Parameters

NAME (required)

Name of a loaded Site Plan Node.

OUTPUT (required)

Name of the output variable. This variable is set to `true` if the page asset in question has an unplaced ancestor.

Description

The `SITEPLAN.UNPLACEDANCESTOR` tag returns a list of the loaded site node's ancestors in the `SitePlanTree`.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

Type List Tags

These tags enable you to manage Content Server asset types. The following section describes each Type List tag, beginning with [TYPELIST.ADDASSETTYPE](#).

TYPELIST.ADDASSETTYPE

Adds an asset type to the type list object.

Syntax

```
<TYPELIST.ADDASSETTYPE
  NAME="typelist_object_name"
  VALUE="asset_type" />
```

Parameters

NAME (required)
Name of the type list object.

VALUE (required)
Name of the asset type to add.

TYPELIST.CLEARASSETTYPES

Clears the type list of all asset types.

Syntax

```
<TYPELIST.CLEARASSETTYPES
  NAME="typelist_object_name" />
```

Parameters

NAME (required)
Name of the type list object.

TYPELIST.DELETEASSETTYPE

Deletes an asset type from the type list object.

Syntax

```
<TYPELIST.DELETEASSETTYPE
  NAME="typelist_object_name"
  VALUE="asset_type" />
```

Parameters

NAME (required)
Name of the type list object.

VALUE (required)
Name of asset type to delete.

TYPELIST.GATHER

Gathers the variable, Prefix+types, into a type list object.

Syntax

```
<TYPELIST.GATHER  
  NAME="typelist_object_name"  
  PREFIX="variable_prefix" />
```

Parameters

NAME (required)
Name of the type list object whose fields you want to gather.

PREFIX (required)
First part of the name of all variables and/or lists used by this gather operation. Variables are typically named with the prefix together with the field name. If a delimiter, such as a colon, was used in the creation of the variables that you want to gather, you must use the same delimiter in the gather operation. For example, Variables.*prefix:fieldname*.

TYPELIST.GETALLASSETTYPES

Gets the list of types as a comma-separated string.

Syntax

```
<TYPELIST.GETALLASSETTYPES  
  NAME="list_object_name"  
  VARNAME="variable_name" />
```

Parameters

NAME (required)
Name of the type list object.

VARNAME (required)
Name of the output variable.

TYPELIST.HASTYPE

Verifies whether an asset type is present in a type list.

Syntax

```
<TYPELIST.HASTYPE  
  NAME="typelist_object_name"  
  VALUE="asset_type"  
  VARNAME="variable_name" />
```

Parameters

NAME (required)

Name of the type list object.

VALUE (required)

Name of the asset type to check.

VARNAME

Name of the output variable. The value is set to `true` if the asset type is found in the asset type list.

TYPELIST.ISALL

Verifies whether the type list describes the "all" or "any" condition.

Syntax

```
<TYPELIST.ISALL
  NAME="typelist_object_name"
  VARNAME="variable_name" />
```

Parameters

NAME (required)

Name of the type list object.

VARNAME (required)

Name of the output variable to create. The value is set to `true` if the object specifies "any".

Description

If the type list for a workflow process is set to "any," then the workflow process is applicable to an asset of any asset type.

TYPELIST.SCATTER

Writes (or "scatters") the values in a type list into variables that are named with the convention, `PREFIX+types`.

Syntax

```
<TYPELIST.SCATTER
  NAME="typelist_object_name"
  PREFIX="variable_prefix" />
```

Parameters

NAME (required)

Name of the type list object.

PREFIX (required)

First part of the name of all variables and/or lists used by this scatter operation. Variables are typically named with the prefix together with the field name. A delimiter is not automatically inserted after the prefix. Therefore, if you want a delimiter, such

as a colon, you must specify it with the prefix. For example,
`Variables.prefix:fieldname`.

Description

This scatter operation sets the variable, `Prefix+types`.

TYPELIST.SETALL

Sets the type list to indicate "any" asset type.

Syntax

```
<TYPELIST.SETALL  
  NAME="typelist_object_name"  
  VALUE="asset_type" />
```

Parameters

NAME (required)
Name of the type list object.

VALUE (required)
Name of the asset type.

User List Tags

These tags enable you to manage lists of Content Server users. The following section describes each User List tag, beginning with [USERLIST.ADD](#).

USERLIST.ADD

Adds a user to the list.

Syntax

```
<USERLIST.ADD  
  NAME="list_object_name"  
  USER="user_ID" />
```

Parameters

NAME (required)
Name of the user list object.

USER (required)
ID of the user to add.

USERLIST.CLEAR

Clears the user list.

Syntax

```
<USERLIST.CLEAR  
  NAME="list_object_name" />
```

Parameters

NAME (required)
Name of the user list object.

USERLIST.DELETE

Deletes a user from the list.

Syntax

```
<USERLIST.DELETE  
  NAME="list_object_name"  
  USER="user_ID" />
```

Parameters

NAME (required)
Name of the user list object.

USER (required)
ID of the user to delete.

USERLIST.GATHER

Gathers the variable, PREFIX+roles, to create a user list object.

Syntax

```
<USERLIST.GATHER  
  NAME="list_object_name"  
  PREFIX="variable_prefix" />
```

Parameters

NAME (required)
Name of the user list object.

PREFIX (required)
First part of the name of all variables and/or lists used by this gather operation. Variables are typically named with the prefix together with the field name. If a delimiter, such as a colon, was used in the creation of the variables that you want to gather, you must use the same delimiter in the gather operation. For example, Variables.*prefix:fieldname*.

USERLIST.GETALL

Gets the list of user IDs as a comma-separated string.

Syntax

```
<USERLIST.GETALL  
  NAME="userlist_object_name"  
  LISTVARNAME="user_ID_list_name" />
```

Parameters

NAME (required)
Name of the user list object.

LISTVARNAME
Name of the list variable to create. The user IDs are stored in the ITEM column.

USERLIST.SCATTER

Writes ("scatters") the user list object into the variable, PREFIX+roles.

Syntax

```
<USERLIST.SCATTER  
  NAME="list_object_name"  
  PREFIX="variable_prefix" />
```


Parameters

NAME (required)

Name of the user list object.

PREFIX (required)

First part of the name of all variables and/or lists used by this scatter operation.

Variables are typically named with the prefix together with the field name. A delimiter is not automatically inserted after the prefix. Therefore, if you want a delimiter, such as a colon, you must specify it with the prefix. For example,
Variables.prefix:fieldname.

User Manager Tags

This group of tags enable you to manage user information. The following section describes each User Manager tag, beginning with [USERMANAGER.DELETESITE](#).

USERMANAGER.DELETESITE

Deletes a site.

Syntax

```
<USERMANAGER.DELETESITE
  SITE="publication ID" />
```

Parameters

SITE (required)
Site (formerly publication) ID.

errno

The possible values of errno include:

Value	Description
-12051	Directory error.

USERMANAGER.DELETEUSER

Deletes a user.

Syntax

```
<USERMANAGER.DELETEUSER
  USER="user ID" />
```

Parameters

USER (required)
User ID.

errno

The possible values of errno include:

Value	Description
-100	General exception.
-12001	An expected table or Ilist field was not found.
-12051	Directory error.

USERMANAGER.GETLOGINUSER

Obtains the logged-in user ID.

Syntax

```
<USERMANAGER.GETLOGINUSER
    VARNAME="variable name" />
```

Parameters

VARNAME (required)
Name of the output variable to create.

errno

The possible values of errno include:

Value	Description
-100	General exception.
-12001	An expected table or Ilist field was not found.
-12051	Directory error.

USERMANAGER.GETLOGINUSERNAME

Obtains the logged-in user name.

Syntax

```
<USERMANAGER.GETLOGINUSERNAME
    VARNAME="variable name" />
```

Parameters

VARNAME (required)
Name of the output variable.

errno

The possible values of errno include:

Value	Description
-100	General exception.
-12001	An expected table or Ilist field was not found.
-12051	Directory error.

USERMANAGER.GETUSER

Loads or creates a user instance with the specified user ID.

Syntax

```
<USERMANAGER.GETUSER  
  USER="user ID"  
  OBJVARNAME="variable name" />
```

Parameters

USER (required)

User ID.

OBJVARNAME (required)

Name of the output variable.

errno

The possible values of errno include:

Value	Description
-100	General exception.
-12051	Directory error.
-12052	Null object.

USERMANAGER.GETUSERFROMNAME

Loads or creates a user instance based on user name.

Syntax

```
<USERMANAGER.GETUSERFROMNAME  
  USER="user name"  
  OBJVARNAME="variable name" />
```

Parameters

USER (required)

Name of the user.

OBJVARNAME (required)

Name of the output variable to create.

errno

The possible values of errno include:

Value	Description
-100	General exception.
-12001	An expected table or Ilist field was not found.
-12051	Directory error.

USERMANAGER.GETUSERNAME

Obtains the displayable name for a user when the user's ID is specified.

Syntax

```
<USERMANAGER.GETUSERNAME
  VARNAME="variable name"
  USER="user ID" />
```

Parameters

VARNAME (required)

Name of the output variable to create.

USER (required)

User ID.

errno

The possible values of errno include:

Value	Description
-100	General exception.
-12001	An expected table or Ilist field was not found.
-12051	Directory error.

USERMANAGER.GETUSERS

Gets a user with a specified role.

Syntax

```
<USERMANAGER.GETUSERS  
  PREFIX="prefix"  
  ROLE="role name"  
  SITE="site ID" />
```

Parameters

PREFIX

The first part of the name of all variables and/or lists used by this scatter operation. Variables are typically named with the prefix together with the field name. A delimiter is not automatically inserted after the prefix. Therefore, if a delimiter, such as a colon, is desired, it must be included with the prefix. For example, Variables.prefix:fieldname. The total is written into a variable called prefix:Total.

ROLE (required)

Role name.

SITE (required)

Site ID.

Description

This tag creates a family of IUser objects in the ICS object pool. They are named according to the prefix passed to the tag, plus a number which is the index of the user. If the prefix is "Hello", for example, the first IUser object is named "Hello0", etc. It uses directory services to obtain the list of users.

errno

The possible values of errno include:

Value	Description
-100	General exception.
-12001	An expected table or Ilist field was not found.
-12051	Directory error.

USERMANAGER.LOGINUSER

Logs in a user.

Syntax

```
<USERMANAGER.LOGINUSER
  USERNAME="user name"
  PASSWORD="user's password"
  VARNAME="variable name" />
```

Parameters

USERNAME (required)

User's login ID.

PASSWORD (required)

User's password.

VARNAME (required)

Returns true if the login was successful and false if the login failed.

errno

The possible values of errno include:

Value	Description
-2	Bad password.
-100	General exception.
-12001	An expected table or Ilist field was not found.

USERMANAGER.LOGOUT

Logs out the user.

Syntax

```
<USERMANAGER.LOGOUT />
```

Parameters

None.

USERMANAGER.SETUSER

Saves a user's information.

Syntax

```
<USERMANAGER.SETUSER  
  OBJECT="user object name" />
```

Parameters

OBJECT (required)
Name of Iuser object.

errno

The possible values of errno include:

Value	Description
-12051	Directory error.

Variables

When you save a template asset, CS-Direct creates a page entry in the SiteCatalog for it and sets the values of the standard CS-Direct variables (c, cid, ct, p, and rendermode) in the resargs2 column for that entry.

The following section describes each variable, beginning with [c](#).

For more information about variables, see the *CSEE Developer's Guide*.

c

Asset type that the template formats.

Syntax

```
c="asset type"
```

Parameters

None.

Description

CS-Direct sets this variable by default when you save the template asset.

Example

This example is taken from the Burlington Financial example site included with Content Server. First, the page named Home loads the home page asset, names it HomePage, and then scatters the information in its fields:

```
<ASSET.LOAD TYPE="Variables.c" OBJECTID="Variables.cid"
NAME="HomePage"/>
<ASSET.SCATTER NAME="HomePage" PREFIX="asset"/>
```

The values for c and cid are passed in from the Burlington Financial URL.

See Also

[cid](#)

[ct](#)

[eid](#)

[p](#)

[rendermode](#)

[tid](#)

cid

ID of the asset being rendered or formatted.

Syntax

`c="asset ID"`

Parameters

None.

Description

When an element either calls the templates' page entry or calls an element directly, it must pass in the asset's ID so thecode knows which asset it is supposed to format.

Example

This example is taken from the Burlington Financial example site included with Content Server. First, the page named Home loads the home page asset, names it HomePage, and then scatters the information in its fields:

```
<ASSET.LOAD TYPE="Variables.c" OBJECTID="Variables.cid"
NAME="HomePage"/>
<ASSET.SCATTER NAME="HomePage" PREFIX="asset"/>
```

The values for c and cid are passed in from the Burlington Financial URL.

See Also

[c](#)

[ct](#)

[eid](#)

[p](#)

[rendermode](#)

[tid](#)

ct

Value of a child template.

Syntax

`ct="template name"`

Parameters

None.

Description

When you want to use a template other than an asset's default template, supply the name of an alternate template with the `ct` variable.

Example

The following example specifies which template to use to render the asset.

```
<CALLELEMENT NAME="BurlingtonFinancial/Common/TextOnlyLink">
  <ARGUMENT NAME="ct" VALUE="SectionFrontText"/>
  <ARGUMENT NAME="assettype" VALUE="Page"/>
</CALLELEMENT>
```

TextOnlyLink is the element that creates the Plain Text Link. SectionFront passes it the name of the alternate template (`ct="SectionFrontText"`) and the name of the asset type (`assettype="Page"`).

See Also

[c](#)
[cid](#)
[eid](#)
[p](#)
[rendermode](#)
[tid](#)

eid

ID of a CSElement asset.

Syntax

`eid="CSElement asset ID"`

Parameters

None.

Description

`eid` is available to the CSElement's root element.

See Also

[c](#)
[cid](#)
[ct](#)
[p](#)
[rendermode](#)
[tid](#)

p

ID of the parent page.

Syntax

`p="parent page ID"`

Parameters

None.

Description

When you want to display information based on the hierarchy of your online site (a navigation bar with links to section pages, for example), you must provide a value for `p` so the code knows where to start in order to determine the hierarchical position of the current page. For more information, use Content Server Explorer to open and examine: `ElementCatalog/BurlingtonFinancial/Pagelets/Site/TopPageBreadCrumb`.

See Also

[c](#)

[cid](#)

[ct](#)

[eid](#)

[rendermode](#)

[tid](#)

rendermode

Specifies whether a page entry is to be delivered live, exported, or previewed.

Syntax

`rendermode="live|exported|previewed"`

Parameters

None.

Description

The default value of `rendermode` is `live`. When you Export to Disk, or use the Preview function, CS-Direct overrides the value of this variable with `export` or `preview`. This value is used internally and must not be modified.

See Also

[c](#)

[cid](#)

[ct](#)

[eid](#)[p](#)[tid](#)

tid

ID of the template associated with this asset.

Syntax

`tid="template ID"`

Parameters

None.

Description

CS-Direct sets this variable by default.

See Also

[c](#)[cid](#)[ct](#)[eid](#)[p](#)[rendermode](#)

Workflowable Object Tags

Workflowable object tags work for all generic object descriptions that are used by the workflow engine. The following section describes each workflowable object tag, beginning with [WORKFLOWABLEOBJECT.GETDISPLAYABLENAME](#).

WORKFLOWABLEOBJECT.GETDISPLAYABLENAME

Retrieves a description of the object that can be displayed in the user interface.

Syntax

```
<WORKFLOWABLEOBJECT.GETDISPLAYABLENAME  
  OBJECT="workflow asset object"  
  VARNAME="variable name" />
```

Parameters

OBJECT (required)
Name of the workflowable object.

VARNAME (required)
Name of the output variable.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

Example

```
<WORKFLOWABLEOBJECT.GETDISPLAYABLENAME OBJECT="workflowableobj"  
  VARNAME="description"/>
```

WORKFLOWABLEOBJECT.GETOBJECTNAME

Retrieves the unique name for the workflowable object.

Syntax

```
<WORKFLOWABLEOBJECT.GETOBJECTNAME  
  OBJECT="workflow asset object"  
  VARNAME="variable name" />
```

Parameters

OBJECT (required)
Name of the workflowable object.

VARNAME (required)

Name of the output variable.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

Example

```
<WORKFLOWABLEOBJECT.GETOBJECTNAME OBJECT="workflowablobj"  
    VARNAME="name"/>
```

Workflow Action Tags

Workflow action tags manage workflow action objects. There are six kinds of workflow action objects.

- Conditions
- Deadlock actions
- Delegation actions
- Step actions
- Timed actions
- Group actions

Workflow actions are invoked when a step is taken. The only exception to this rule is the delegate action, which is invoked when an asset is delegated to another user.

Note that workflow action objects are created, loaded, saved or deleted using workflow engine tags. See the following tag descriptions:

[WORKFLOWENGINE.CREATENEWASSIGNMENTACTION](#)
[WORKFLOWENGINE.CREATENEWCONDITION](#)
[WORKFLOWENGINE.CREATENEWDEADLOCKACTION](#)
[WORKFLOWENGINE.CREATENEWDELEGATEACTION](#)
[WORKFLOWENGINE.CREATENEWSTEPACTION](#)

The following section describes each workflow action tag, beginning with [WORKFLOWACTION.GATHER](#).

WORKFLOWACTION.GATHER

Gathers variables that have been named with the specified prefix convention to create a workflow action object.

Syntax

```
<WORKFLOWACTION.GATHER
  NAME="workflowaction object name"
  PREFIX="variable name" />
```

Parameters

NAME (required)

Name of the workflow action object whose fields you want to gather.

PREFIX (required)

First part of the name of all variables and/or lists used by this gather operation.

Variables are typically named with the prefix together with the field name. If a delimiter, such as a colon, was used in the creation of the variables that you want to gather, you must use the same delimiter in the gather operation. For example, `Variables.prefix:fieldname`.

Description

The `WORKFLOWACTION.GATHER` operation gathers the following variables into a workflow action object:

- prefix+id
- prefix+name
- prefix+description
- prefix+elementname
- prefix+arguments

WORKFLOWACTION.GETARGUMENTS

Gets the arguments of the workflow action.

Syntax

```
<WORKFLOWACTION.GETARGUMENTS
  NAME="workflowaction object name"
  VARNAME="variable name" />
```

Parameters

NAME (required)
Name of workflow action object.

VARNAME (required)
Name of the output variable to create.

WORKFLOWACTION.GETDESCRIPTION

Gets the description of the workflow action.

Syntax

```
<WORKFLOWACTION.GETDESCRIPTION
  NAME="workflowaction object name"
  VARNAME="variable name" />
```

Parameters

NAME (required)
Name of workflow action object.

VARNAME (required)
Name of output variable to create.

WORKFLOWACTION.GETELEMENTNAME

Gets the element name of the workflow action.

Syntax

```
<WORKFLOWACTION.GETELEMENTNAME
  NAME="action object name"
  VARNAME="variable name" />
```

Parameters

NAME (required)
Name of the workflow action object.

VARNAME (required)
Name of the output variable to create.

WORKFLOWACTION.GETID

Gets the ID of the workflow action object.

Syntax

```
<WORKFLOWACTION.GETID  
  NAME="workflowaction object name"  
  VARNAME="variable name" />
```

Parameters

NAME (required)
Name of the workflow action.

VARNAME (required)
Name of the output variable.

WORKFLOWACTION.GETNAME

Gets the name of the workflow action.

Syntax

```
<WORKFLOWACTION.GETNAME  
  NAME="workflowaction object name"  
  VARNAME="variable name" />
```

Parameters

NAME (required)
Name of the workflow action object.

VARNAME (required)
Name of the output variable to create.

WORKFLOWACTION.SCATTER

Writes ("scatters") the variables of the loaded workflow action object into variables with the PREFIX naming convention shown in the Description.

Syntax

```
<WORKFLOWACTION.SCATTER  
  NAME="workflowaction object name"  
  PREFIX="variable_name" />
```

Parameters

NAME (required)

The name of the workflow action object whose fields are to be scattered.

PREFIX (required)

First part of the name of all variables and/or lists used by this scatter operation.

Variables are typically named with the prefix together with the field name. A delimiter is not automatically inserted after the prefix. Therefore, if you want a delimiter, such as a colon, you must specify it with the prefix. For example, Variables.*prefix:fieldname*.

Description

This scatter operation sets the following variables:

- prefix+id
- prefix+name
- prefix+description
- prefix+elementname
- prefix+arguments

WORKFLOWACTION.SETARGUMENTS

Sets the arguments of the workflow action.

Syntax

```
<WORKFLOWACTION.SETARGUMENTS
  NAME="workflowaction object name"
  VALUE="argument" />
```

Parameters

NAME (required)

Name of the workflow action object

VALUE (required)

Workflow action argument to be set.

WORKFLOWACTION.SETDESCRIPTION

Sets the description of the workflow action.

Syntax

```
<WORKFLOWACTION.SETDESCRIPTION
  NAME="workflowaction object name"
  VALUE="description text" />
```

Parameters

NAME (required)

Name of workflow action.

VALUE (required)

Description text.

WORKFLOWACTION.SETELEMENTNAME

Sets the element name of the workflow action.

Syntax

```
<WORKFLOWACTION.SETELEMENTNAME  
  NAME="action object name"  
  VALUE="element name" />
```

Parameters

NAME (required)

Name of the workflow action object.

VALUE (required)

Name of the element.

WORKFLOWACTION.SETNAME

Sets the name of the workflow action.

Syntax

```
<WORKFLOWACTION.SETNAME  
  NAME="workflowaction object name"  
  VALUE="new workflow action name" />
```

Parameters

NAME (required)

Workflow action object name.

VALUE (required)

New workflow action name.

Workflow Action List Tags

Workflow action list tags manage the workflow action list objects. The following section describes each workflow action list tag, beginning with [WORKFLOWACTIONLIST.ADDACTION](#).

WORKFLOWACTIONLIST.ADDACTION

Adds an action to a workflow action list.

Syntax

```
<WORKFLOWACTIONLIST.ADDACTION
  NAME="workflowactionlist_object"
  VALUE="ID_of_workflow_action_to_add" />
```

Parameters

NAME (required)

Name of the workflow action list object.

VALUE (required)

ID of the workflow action to add.

WORKFLOWACTIONLIST.CLEARACTIONS

Removes all actions from a workflow action list.

Syntax

```
<WORKFLOWACTIONLIST.CLEARACTIONS
  NAME="workflowactionlist_object" />
```

Parameters

NAME (required)

Name of the workflow action list.

WORKFLOWACTIONLIST.GATHER

Gathers variables that have been named with the specified prefix convention into a workflow action list object.

Syntax

```
<WORKFLOWACTIONLSIST.GATHER
  NAME="workflowactionlist_object"
  PREFIX="variable_prefix" />
```

Parameters

NAME (required)

The name of the workflow action list object whose fields you want to gather.

PREFIX (required)

First part of the name of all variables and/or lists used by this gather operation.

Variables are typically named with the prefix together with the field name. If a delimiter, such as a colon, was used in the creation of the variables that you want to gather, you must use the same delimiter in the gather operation. For example, Variables.*prefix:fieldname*.

Description

This operation gathers the following variables into a workflow action list object:

- prefix+action+Total
- prefix+action+i (where i is a number)

WORKFLOWACTIONLIST.GETALLACTIONS

Loads all actions of a workflow action list.

Syntax

```
<WORKFLOWACTIONLIST.GETALLACTIONS
  NAME="workflowactionlist_object_name"
  VARNAME="variable_name" />
```

Parameters

NAME (required)

Name of workflow action list object.

VARNAME (required)

Name of output variable to create. This variable contains a comma-separated list of all actions.

WORKFLOWACTIONLIST.REMOVEACTION

Removes one action from the workflow action list.

Syntax

```
<WORKFLOWACTIONLIST.REMOVEACTION
  NAME="workflowactionlist_object_name"
  VALUE="action_ID" />
```

Parameters

NAME (required)

Name of the workflow action list object.

VALUE (required)

ID of the action to remove.

WORKFLOWACTIONLIST.SCATTER

Creates independent variables and lists from an in-memory workflow action list object

Syntax

```
<WORKFLOWACTIONLIST.SCATTER  
  NAME="workflowactionlist_object_name"  
  PREFIX="variable_prefix" />
```

Parameters

NAME (required)

Name of the workflow action list object whose fields are to be scattered.

PREFIX (required)

First part of the name of all variables and/or lists used by this scatter operation. Variables are typically named with the prefix together with the field name. A delimiter is not automatically inserted after the prefix. Therefore, if you want a delimiter, such as a colon, you must specify it with the prefix. For example, Variables.*prefix:fieldname*.

Description

This scatter operation sets the following variables:

- prefix+action+Total
- prefix+action+i (where i is an integer)

Workflow Asset Tags

Workflow asset tags work on asset descriptions that are used by the workflow engine. The following section describes each workflow asset tag, beginning with [WORKFLOWASSET.GETASSETID](#).

WORKFLOWASSET.GETASSETID

Retrieves the asset ID from the asset object.

Syntax

```
<WORKFLOWASSET.GETASSETID  
  OBJECT="asset name"  
  VARNAME="variable name"/>
```

Parameters

OBJECT (required)

Name of the asset object.

VARNAME (required)

Name of the output variable.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

Example

The following example retrieves the asset ID from the workflow asset description called assetobj.

```
<WORKFLOWASSET.GETASSETID OBJECT="assetobj"  
  VARNAME="assetid"/>
```

Release Introduced

This tag was introduced in CSEE release 4.0.

WORKFLOWASSET.GETASSETTYPE

Retrieves the asset type from the asset object.

Syntax

```
<WORKFLOWASSET.GETASSETTYPE
```



```
OBJECT="asset name"
VARNAME="variable name"/>
```

Parameters

OBJECT (required)

Name of the asset object.

VARNAME (required)

Name of the output variable.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

Example

```
<WORKFLOWASSET.GETASSETTYPE OBJECT="assetobj"
    VARNAME="assettype"/>
```

Release Introduced

This tag was introduced in CSEE release 4.0.

Workflow Assignment Tags

Workflow assignments represent associations between assets and users. Workflow assignment objects are created, loaded, saved and deleted using workflow engine tags, such as [WORKFLOWENGINE.CREATENEWASSIGNMENTACTION](#). Once created, these objects can be managed by the following workflow assignment tags. Note that the workflow assignment tags do not include the GATHER function found in some other tags.

The following section describes each workflow assignment tag, beginning with [WORKFLOWASSIGNMENT.GETASSIGNDATE](#).

WORKFLOWASSIGNMENT.GETASSIGNDATE

Retrieves the date the assignment was made.

Syntax

```
<WORKFLOWASSIGNMENT.GETASSIGNDATE NAME="assignment name"
    VARNAME="variable name"/>
```

Parameters

NAME (required)

Name of the assignment.

VARNAME (required)

Name of the output variable. The format of the returned date is:

yyyy-mm-dd hr:min:sec

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

Example

```
<WORKFLOWASSIGNMENT.GETASSIGNDATE NAME="assignobj"
    VARNAME="assigndate"/>
```

See Also

[WORKFLOWASSIGNMENT.GETGROUPEDCOMMENT](#)

WORKFLOWASSIGNMENT.GETASSIGNEDOBJECT

Gets the object of a workflow assignment.

Syntax

```
<WORKFLOWASSIGNMENT.GETASSIGNEDOBJECT
  NAME="object name"
  OBJVARNAME="object variable"/>
```

Parameters

NAME (required)

Name of the assignment object.

OBJVARNAME (required)

Name of the output object variable.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

Example

The following example retrieves the object called Pub .

```
<WORKFLOWASSIGNMENT.GETASSIGNEDOBJECT NAME="pub"
  OBJVARNAME="pub1"/>
```

Release Introduced

This tag was introduced in CSEE release 4.0.

See Also

[WORKFLOWASSIGNMENT.GETASSIGNERUSERID](#)

WORKFLOWASSIGNMENT.GETASSIGNEDUSERID

Retrieves the user ID for the specified assignment.

Syntax

```
<WORKFLOWASSIGNMENT.GETASSIGNEDUSERID
  NAME="assignment name"
  VARNAME="variable name"/>
```

Parameters

NAME (required)

Name of the assignment object.

VARNAME (required)

Name of the output variable.

errno

The possible values of `errno` include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

Example

```
<WORKFLOWASSIGNMENT.GETASSIGNEDUSERID NAME="assignobj"
  VARNAME="userid"/>
```

See Also

[WORKFLOWASSIGNMENT.GETASSIGNERUSERID](#)

WORKFLOWASSIGNMENT.GETASSIGNERUSERID

Retrieves the ID of the user who gave the assignment.

Syntax

```
<WORKFLOWASSIGNMENT.GETASSIGNERUSERID
  NAME="assignment name"
  VARNAME="variable name"/>
```

Parameters

NAME (required)

Name of the assignment object.

VARNAME (required)

Name of the output variable.

errno

The possible values of `errno` include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

Example

The following example retrieves and assigns a user ID from an assignment object named `assignobj`.

```
<WORKFLOWASSIGNMENT.GETASSIGNERUSERID NAME="assignobj"
  VARNAME="assignerid"/>
```

Release Introduced

This tag was introduced in CSEE release 4.0.

See Also

[WORKFLOWASSIGNMENT.GETSTATEID](#)

WORKFLOWASSIGNMENT.GETASSIGNMENTCOMMENT

Retrieves the comment for the assignment.

Syntax

```
<WORKFLOWASSIGNMENT.GETASSIGNMENTCOMMENT
  NAME="assignment name"
  VARNAME="variable name"/>
```

Parameters

NAME (required)
Name of the assignment object.

VARNAME (required)
Name of the output variable.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

Example

```
<WORKFLOWASSIGNMENT.GETASSIGNMENTCOMMENT NAME="assignobj"
  VARNAME="comment"/>
```

Release Introduced

This tag was introduced in CSEE release 4.0.

WORKFLOWASSIGNMENT.GETDEADLINE

Retrieves the due date of an assignment.

Syntax

```
<WORKFLOWASSIGNMENT.GETDEADLINE NAME="assignment name"
  VARNAME="variable name"/>
```

Parameters

NAME (required)
Name of the assignment object.

VARNAME (required)
Name of the output variable.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

Example

```
<WORKFLOWASSIGNMENT.GETDEADLINE NAME="assignobj"
    VARNAME="duedate"/>
```

See Also

[WORKFLOWASSIGNMENT.GETDISPOSITIONDATE](#)

WORKFLOWASSIGNMENT.GETDISPOSITION

Retrieves the disposition of the assignment and classifies it as cancelled, completed, delegated, etc.

Syntax

```
<WORKFLOWASSIGNMENT.GETDISPOSITION
    NAME="assignment name"
    VARNAME="variable name"/>
```

Parameters

NAME (required)
Name of the assignment object.

VARNAME (required)
Name of the output variable.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

Example

```
<WORKFLOWASSIGNMENT.GETDISPOSITION NAME="assignobj"
    VARNAME="assigndisp"/>
```

Release Introduced

This tag was introduced in CSEE release 4.0.

WORKFLOWASSIGNMENT.GETDISPOSITIONCOMMENT

Retrieves the disposition comment for the assignment.

Syntax

```
<WORKFLOWASSIGNMENT.GETDISPOSITIONCOMMENT NAME="Assignment Name"
    VARNAME="Variable name"/>
```

Parameters

NAME (required)
Name of the assignment object.

VARNAME (required)
Name of the output variable.

errno

The possible values of `errno` include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

Example

```
<WORKFLOWASSIGNMENT.GETDISPOSITIONEDCOMMENT NAME="assignobj"
    VARNAME="comment"/>
```

WORKFLOWASSIGNMENT.GETDISPOSITIONDATE

Retrieves the disposition date of the assignment.

Syntax

```
<WORKFLOWASSIGNMENT.GETDISPOSITIONDATE NAME="assignment name"
    VARNAME="variable name" />
```

Parameters

NAME (required)
Name of the assignment.

VARNAME (required)
Name of the output variable.

errno

The possible values of `errno` include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

Example

```
<WORKFLOWASSIGNMENT.GETDISPOSITIONDATE NAME="assignobj"
  VARNAME="datedispos">
```

WORKFLOWASSIGNMENT.GETDISPOSITIONUSERID

Gets the ID of workflow assignee.

Syntax

```
<WORKFLOWASSIGNMENT.GETDISPOSITIONUSERID
  NAME="assignment name"
  VARNAME="variable name" />
```

Parameters

NAME (required)
Name of the assignment.

VARNAME (required)
Name of the output variable to create. The output is a numeric value that represents the number of milliseconds between the deadline and midnight Jan 1, 1970.

Description

The disposition user is the workflow assignee, who is the person to whom the asset is assigned.

errno

The possible values of `errno` include:

Value	Description
-12020	No object by name
-12037	Required field missing

WORKFLOWASSIGNMENT.GETGROUPEDCOMMENT

Retrieves the grouped comment for the assignment.

Syntax

```
<WORKFLOWASSIGNMENT.GETGROUPEDCOMMENT
  NAME="assignment name"
  VARNAME="variable name"/>
```

Parameters

NAME (required)
Name of the assignment object.

VARNAME (required)
Name of the output variable.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

Example

```
<WORKFLOWASSIGNMENT.GETGROUPEDCOMMENT NAME="assignobj"
  VARNAME="comment"/>
```

Release Introduced

This tag was introduced in CSEE release 4.0.

WORKFLOWASSIGNMENT.GETGROUPEDDEADLINE

Retrieves the grouped due date of the assignment.

Syntax

```
<WORKFLOWASSIGNMENT.GETGROUPEDDEADLINE NAME="assignment name"
  VARNAME="variable name"/>
```

Parameters

NAME (required)
Name of the assignment object.

VARNAME (required)
Name of the output variable.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

Example

```
<WORKFLOWASSIGNMENT.GETGROUPEDDEADLINE NAME="assignobj"
    VARNAME="duedate"/>
```

Release Introduced

This tag was introduced in CSEE release 4.0.

WORKFLOWASSIGNMENT.GETGROUPID

Retrieves the group ID for the assignment.

Syntax

```
<WORKFLOWASSIGNMENT.GETGROUPID
    NAME="assignment name"
    VARNAME="variable name"/>
```

Parameters

NAME (required)
Name of the assignment object.

VARNAME (required)
Name of the output variable.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

Example

```
<WORKFLOWASSIGNMENT.GETGROUPID NAME="assignobj"
    VARNAME="groupid"/>
```

Release Introduced

This tag was introduced in CSEE release 4.0.

See Also

[WORKFLOWASSIGNMENT.GETDISPOSITIONDATE](#)

WORKFLOWASSIGNMENT.GETGROUPEDWORKFLOWEDSTEPID

Retrieves the ID for the workflow step of the assignment.

Syntax

```
<WORKFLOWASSIGNMENT.GETGROUPWORKFLOWEDSTEPID
    NAME="assignment name"
    VARNAME="variable name"/>
```

Parameters

NAME (required)

Name of the assignment.

VARNAME (required)

Name of the output variable.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

Example

```
<WORKFLOWASSIGNMENT.GETGROUPWORKFLOWEDSTEPID NAME="assignobj"
    VARNAME="stepid"/>
```

See Also

[WORKFLOWASSIGNMENT.GETDISPOSITIONDATE](#)

WORKFLOWASSIGNMENT.GETID

Gets the ID of the workflow assignment.

Syntax

```
<WORKFLOWASSIGNMENT.GETID
    NAME="assignment name"
    VARNAME="variable name" />
```

Parameters

NAME (required)
Name of the workflow assignment object.

VARNAME (required)
Name of the output variable to create.

WORKFLOWASSIGNMENT.GETQUEUEDWORKFLOWSTEPID

Retrieves the ID of the queued workflow step for an assignment.

Syntax

```
<WORKFLOWASSIGNMENT.GETQUEUEDWORKFLOWSTEPID NAME="assignment name"
  VARNAME="variable name"/>
```

Parameters

NAME (required)
Name of the assignment.

VARNAME (required)
Name of the output variable.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

Example

```
<WORKFLOWASSIGNMENT.GETQUEUEDWORKFLOWSTEPID NAME="assignobj"
  VARNAME="stepid"/>
```

See Also

[WORKFLOWASSIGNMENT.GETDISPOSITIONDATE](#)

WORKFLOWASSIGNMENT.GETSTATEID

Retrieves the ID for the workflow state of the assignment.

Syntax

```
<WORKFLOWASSIGNMENT.GETSTATEID
  NAME="assignment name"
  VARNAME="variable name"/>
```

Parameters

NAME (required)
Name of the assignment object.

VARNAME (required)
Name of the output variable.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

Example

```
<WORKFLOWASSIGNMENT.GETSTATEID NAME="assignobj"
    VARNAME="stateid"/>
```

Release Introduced

This tag was introduced in CSEE release 4.0.

See Also

[WORKFLOWASSIGNMENT.GETDISPOSITIONDATE](#)

WORKFLOWASSIGNMENT.GETSTATUS

Retrieves the status of a workflow assignment.

Syntax

```
<WORKFLOWASSIGNMENT.GETSTATUS NAME="assignment name"
    VARNAME="variable name"/>
```

Parameters

NAME (required)
Name of the assignment object.

VARNAME (required)
Name of the output variable. The possible outputs are “y” (active,) “n” (not active,) “q” (queued,) and “a” (abstain).

errno

The possible values of errno include:

Value	Description
-12020	No object by name.

Value	Description
-12037	Required field missing.

Example

```
<WORKFLOWASSIGNMENT.GETSTATUS NAME="assignobj"
    VARNAME="assingstatus"/>
```

WORKFLOWASSIGNMENT.SCATTER

Writes (“scatters”) the workflow assignment object into a group of variables that are named with the specified prefix convention.

Syntax

```
<WORKFLOWASSIGNMENT.SCATTER
    NAME="workflowassignment object name"
    PREFIX="variable prefix" />
```

Parameters

NAME (required)

Name of the workflow assignment object whose fields are to be scattered.

PREFIX (required)

The first part of the name of all variables and/or lists used by this scatter operation. Variables are typically named with the prefix together with the field name. A delimiter is not automatically inserted after the prefix. Therefore, if a delimiter, such as a colon, is desired, it must be included with the prefix. For example, Variables.*prefix:fieldname*.

Description

This scatter operation sets the following variables:

- prefix+id
- prefix+assigner
- prefix+assigned
- prefix+assigncomment
- prefix+assignedrole
- prefix+state
- prefix+group
- prefix+assigndate
- prefix+disposition
- prefix+disposedate
- prefix+disposeuser
- prefix+disposecomment

- prefix+deadline
- prefix+status

errno

The possible values of errno include:

Value	Description
-12020	No object by name
-12037	Required field missing

Workflow Engine Tags

Workflow engine tags are used to install, uninstall, administer, and use workflow. Workflow engine tags are also used to create, save and delete workflow objects. After a workflow object has been created using the appropriate workflow engine tags, it can be managed by using the corresponding workflow class for that object. For example, workflow assignment action objects are managed by workflow assignment tags.

The following section describes each workflow engine tag, beginning with [WORKFLOWENGINE.ABSTAINFROMVOTING](#).

WORKFLOWENGINE.ABSTAINFROMVOTING

Allows a user to abstain from voting.

Syntax

```
<WORKFLOWENGINE.ABSTAINFROMVOTING
  SITE="pub_ID"
  ID="assignment ID"
  OBJECT="workflow_asset_object"
  [COMMENT="comment"] />
```

Parameters

SITE (required)
Publication ID

ID (required)
Assignment ID.

OBJECT (required)
Name of the loaded workflow asset object.

COMMENT (optional)
Workflow participant's comments.

Description

The `WORKFLOWENGINE.ABSTAINFROMVOTING` tags indicates that the workflow participant/assignee is abstaining from voting.

errno

The possible values of `errno` include:

Value	Description
-12027	No asset assignment
-12035	Cannot cancel. Cannot abstain.

WORKFLOWENGINE.ADDNEWOBJECTTOGROUP

Adds a new asset to a workflow group

Syntax

```
<WORKFLOWENGINE.ADDNEWOBJECTTOGROUP
  SITE="pub_ID"
  OBJECT="workflow_asset_object"
  ID="workflow_group_id"
  [COMMENT="comment"]
  [DEADLINE="deadline"]
  [ASSIGNMENTLIST="participant_object"] />
```

Parameters

SITE (required)

Publication ID.

OBJECT (required)

Name of loaded workflow asset object.

ID (required)

Workflow group ID.

COMMENT (optional)

Workflow participant's comments.

DEADLINE (optional)

Date object. Stored as time in milliseconds since midnight of 1/1/1970.

ASSIGNMENTLIST (optional)

Name of participant object for a "Choose assignee when step is taken" assignment method. If none is entered, it uses the default set by the workflow administrator. (See Description.)

Description

The `WORKFLOWENGINE.ADDNEWOBJECTTOGROUP` tag enables you to add a new asset to a workflow group.

Note the following when you use the `ASSIGNMENTLIST` parameter; when adding a new workflow step to a workflow process, the administrator needs to decide on the assignment method, which determines to whom the asset is assigned in the next workflow state. There are four assignment methods from which to choose:

- Retain "From State" assignees
- No assignments, control actions with function privs
- Assign from list of participants
- Choose assignee when step is taken

If the administrator selects the "Choose assignee when step is take" assignment method, then the workflow participant is presented with a list of possible assignees at the time of "Finish my assignment".

For more information about setting up a workflow process, see the *CSEE Administrator's Guide*.

errno

The possible values of `errno` include:

Value	Description
-12024	Missing group. Workflow group not found.
-12030	Insufficient privileges.
-12041	Already in workflow.
-12028	Not logged in.
-12050	Incompatible process.
-12026	Missing step. No start state.
-12001	Missing field.

WORKFLOWENGINE.ADDOBJECTTOGROUP

Adds an asset to a workflow group.

Syntax

```
<WORKFLOWENGINE.ADDOBJECTTOGROUP
  SITE="pub_ID"
  OBJECT="workflow_asset_object"
  ID="workflow_group_ID" />
```

Parameters

SITE (required)

Publication ID.

OBJECT (required)

Name of the loaded workflow asset object.

ID (required)

Workflow group ID.

errno

The possible values of `errno` include:

Value	Description
-12024	Missing group.
-12030	Insufficient privileges.

WORKFLOWENGINE.CANCELASSIGNMENT

Cancels a workflow participant's workflow assignment.

Syntax

```
<WORKFLOWENGINE.CANCELASSIGNMENT
  SITE="pub_ID"
  ID="assignment ID"
  [COMMENT="comment"] />
```

Parameters

SITE (required)

Publication ID.

ID (required)

Assignment ID.

COMMENT (optional)

Workflow participant's comments

errno

The possible values of errno include:

Value	Description
-12027	No asset assignment. Assignment does not exist.
-12035	Cannot cancel

WORKFLOWENGINE.CANDELETEGROUP

Checks to see if a workflow group can be deleted by the current user.

Syntax

```
<WORKFLOWENGINE.CANDELETEGROUP
  SITE="pub_ID"
  ID="workflow_group_ID"
  VARNAME="variable_name" />
```

Parameters

SITE (required)

Publication ID.

ID (required)

Workflow group ID.

VARNAME (required)

Output variable name. Returns either true or false.

errno

The possible values of errno include:

Value	Description
-12024	Missing group.

WORKFLOWENGINE.CANEDITGROUP

Checks to see if a workflow group can be edited by the current user.

Syntax

```
<WORKFLOWENGINE.CANEDITGROUP
  SITE="pub_ID"
  ID="workflow_group_ID"
  VARNAME="variable_name" />
```

Parameters

SITE (required)

Publication ID.

ID (required)

Workflow group ID.

VARNAME (required)

Name of the output variable to create.

errno

The possible values of errno include:

Value	Description
-12024	Missing group.

WORKFLOWENGINE.CANSTARTWORKFLOW

Checks to see if the current user can start the specified workflow process.

Syntax

```
<WORKFLOWENGINE.CANSTARTWORKFLOW
  SITE="pub_ID"
  ID="process ID"
  VARNAME="variable_name"/>
```

Parameters

SITE (required)

Publication ID.

ID (required)

Workflow process ID.

VARNAME (required)

The name of the variable output to create.

errno

The possible values of errno include:

Value	Description
-12026	Missing step. No start step.
-12028	Not logged in.
-12029	Missing user.

WORKFLOWENGINE.CLOSEASSIGNMENTS

Cancels all assignments for an object and removes the object from workflow.

Syntax

```
<WORKFLOWENGINE.CLOSEASSIGNMENTS
  SITE="pub_ID"
  OBJECT="workflow_asset_object"
  [COMMENT="comment"] />
```

Parameters

SITE (required)

Publication ID.

OBJECT (required)

Workflow asset object.

COMMENT (optional)

Workflow participant's comments.

errno

The possible values of errno include:

Value	Description
-12049	Not in workflow.

WORKFLOWENGINE.COMPLETEASSIGNMENT

Allows a user to send an asset to the next state in the workflow process.

Syntax

```
<WORKFLOWENGINE.COMPLETEASSIGNMENT
  SITE="pub_ID"
  ID="assignment_ID"
  [ACTION="action_comment"]
  [COMMENT="comment"]
  NEXTSTEP="next_workflow_step_ID"
  VARNAME="assignment_name"
  [DEADLINE="deadline"]
  [ASSIGNMENTLIST="participant_object"] />
```

Parameters

SITE (required)

Publication ID.

ID (required)

Workflow assignment ID.

ACTION (optional)

Comment about what action the current user has taken.

COMMENT (optional)

Comment about what action the next user should take.

NEXTSTEP (required)

ID of the chosen workflow step.

VARNAME (required)

If COMPLETE ASSIGNMENT action fails, stores the reason for the failure.

DEADLINE (optional)

Date object. Stored as time in milliseconds since midnight of 1/1/1970.

ASSIGNMENTLIST (optional)

Required for the "Choose assignee when step is taken" step. Otherwise, optional.

Description

The WORKFLOWENGINE.COMPLETEASSIGNMENT allows a user to send an asset to the next state in the workflow process.

Note the following when you use the ASSIGNMENTLIST parameter; when adding a new workflow step to a workflow process, the administrator needs to decide on the assignment method, which determines to whom the asset is assigned in the next workflow state. There are four assignment methods from which to choose:

- Retain "From State" assignees
- No assignments, control actions with function privs
- Assign from list of participants
- Choose assignee when step is taken

If the administrator selects the “Choose assignee when step is take” assignment method, then the workflow participant is presented with a list of possible assignees at the time of "Finish my assignment".

For more information about setting up a workflow process, see the *CSEE Administrator's Guide*.

errno

The possible values of `errno` include:

Value	Description
-12027	No asset assignment.
-12036	Wrong asset state.
-12028	Not logged in
-12029	Missing user
-12030	Insufficient privileges

WORKFLOWENGINE.CREATENEWASSIGNMENTACTION

Creates a new assignment action. The assignment is an association between an asset and a user.

Syntax

```
<WORKFLOWENGINE.CREATENEWASSIGNMENTACTION
  OBJVARNAME="assignment_action_object_name" />
```

Parameters

OBJVARNAME (required)
Name of the assignment action object to create.

WORKFLOWENGINE.CREATENEWCONDITION

Creates a new condition.

Syntax

```
<WORKFLOWENGINE.CREATENEWCONDITION
  OBJVARNAME="condition_object_name" />
```

Parameters

OBJVARNAME (required)
Name of the condition object to create.

Description

The `<WORKFLOWENGINE.CREATENEWCONDITION>` tag creates a new condition. The conditions determine whether a step transition can proceed.

WORKFLOWENGINE.CREATENEWDEADLOCKACTION

Creates a new deadlock action.

Syntax

```
<WORKFLOWENGINE.CREATENEWDEADLOCKACTION  
  OBJVARNAME="deadlock_action_object_name" />
```

Parameters

OBJVARNAME (required)
Name of the deadlock action object to create.

Description

The `WORKFLOWENGINE.CREATENEWDEADLOCKACTION` tag creates a new deadlock action, which is executed when two or more workflow participants chose different steps for the asset to go to the next state. Typically, this results in an e-mail being sent to all workflow participants, informing them of the deadlock.

WORKFLOWENGINE.CREATENEWDELEGATEACTION

Creates a new delegation action.

Syntax

```
<WORKFLOWENGINE.CREATENEWDELEGATEACTION  
  OBJVARNAME="delegate_action_object_name" />
```

Parameters

OBJVARNAME (required)
Name of the delegate action object to create.

Description

The `WORKFLOWENGINE.CREATENEWDELEGATEACTION` tag creates a new delegation action, which is triggered when an asset is delegated. The old assignment is closed and a new one is given to the designated workflow participant. The state does not change.

WORKFLOWENGINE.CREATENEWFUNCTION

Creates a new workflow function.

Syntax

```
<WORKFLOWENGINE.CREATENEWFUNCTION  
  OBJVARNAME="workflow_function_name" />
```

Parameters

OBJVARNAME (required)
Name of the workflow function object to create.

Description

The WORKFLOWENGINE.CREATENEWFUNCTION tag creates a new workflow function, which is defined by a name and description.

WORKFLOWENGINE.CREATENEWGROUP

Creates a new group.

Syntax

```
<WORKFLOWENGINE.CREATENEWGROUP  
  OBJVARNAME="workflow_group_object_name" />
```

Parameters

OBJVARNAME (required)
Name of the workflow group object to create.

Description

The WORKFLOWENGINE.CREATENEWGROUP tag creates a new group, which represents a collection of individual assets that all must go through the same workflow process.

WORKFLOWENGINE.CREATENEWGROUPACTION

Creates a new group action.

Syntax

```
<WORKFLOWENGINE.CREATENEWGROUPACTION  
  OBJVARNAME="workflow_group_action_object_name" />
```

Parameters

OBJVARNAME (required)
Name of the workflow group action object to create.

Description

The `WORKFLOWENGINE.CREATENEWGROUPACTION` tag creates a new group action, which is executed when different users choose different next steps for the assets that are in the workflow group. Typically, an e-mail is sent to all participants in the workflow to inform them of the deadlock.

WORKFLOWENGINE.CREATENEWPROCESS

Creates a new workflow process.

Syntax

```
<WORKFLOWENGINE.CREATENEWPROCESS  
  OBJVARNAME="workflow_process_object_name" />
```

Parameters

OBJVARNAME (required)
Name of workflow process object to create.

Description

The `WORKFLOWENGINE.CREATENEWPROCESS` tag creates an empty workflow process, which you complete by adding a list of delegation actions; a starting step; and a list of workflow steps, which are used to link states together. A workflow process has an ID, a name, and a description and is valid for a list of sites, a list of asset types, a list of valid roles and a list of admin roles. See other workflow tags to incorporate this data into the new process created by this tag.

WORKFLOWENGINE.CREATENEWSTATE

Creates a new state.

Syntax

```
<WORKFLOWENGINE.CREATENEWSTATE  
  OBJVARNAME="workflow state object name" />
```

Parameters

OBJVARNAME (required)
Name of the workflow state object to create.

Description

The `WORKFLOWENGINE.CREATENEWSTATE` tag creates a new state, which represents the condition or status of an asset in workflow. States are defined independently of the workflow processes and can be shared by many workflow processes.

WORKFLOWENGINE.CREATENEWSTEPACTION

Creates a new step action.

Syntax

```
<WORKFLOWENGINE.CREATENEWSTEPACTION
  OBJVARNAME="step_action_object_name" />
```

Parameters

OBJVARNAME (required)
Name of the workflow step action object to create.

Description

The `WORKFLOWENGINE.CREATENEWSTEPACTION` tag creates a new step action. The execution of a workflow step action allows an asset to transition between workflow states.

WORKFLOWENGINE.DELEGATEASSIGNMENT

Allows a user to delegate a workflow assignment to a workflow participant with the same role.

Syntax

```
<WORKFLOWENGINE.DELEGATEASSIGNMENT
  SITE="pub_ID"
  ID="assignment_ID"
  OBJECT="workflow_asset_object"
  DELEGATEDTO="ID_of_workflow_participant"
  [COMMENT="comment"] />
```

Parameters

SITE (required)
Publication ID.

ID (required)
Workflow assignment ID.

OBJECT (required)
Name of the workflow asset object.

DELEGATEDTO (required)
ID of designated workflow participant who receives the assignment.

COMMENT (optional)
Assigning workflow participant's comments.

errno

The possible values of `errno` include:

Value	Description
-12035	Cannot delegate.
-12030	Insufficient privileges.

WORKFLOWENGINE.DELETEASSIGNMENTACTIONID

Deletes an assignment action based on its ID.

Syntax

```
<WORKFLOWENGINE.DELETEASSIGNMENTACTIONID
  ID="assignment_action_ID" />
```

Parameters

ID (required)
Assignment action ID.

WORKFLOWENGINE.DELETECONDITIONID

Deletes a condition based on its ID.

Syntax

```
<WORKFLOWENGINE.DELEGATECONDITIONID
  ID="condition_ID" />
```

Parameters

ID (required)
Condition ID.

WORKFLOWENGINE.DELETEDEADLOCKACTIONID

Deletes a deadlock action based on its ID.

Syntax

```
<WORKFLOWENGINE.DELETEDEADLOCKACTIONID
  ID="deadlock_action_ID" />
```

Parameters

ID (required)
Deadlock action ID.

WORKFLOWENGINE.DELETEDELEGATEACTIONID

Deletes a delegate action based on its ID.

Syntax

```
<WORKFLOWENGINE.DELETEDELEGATEACTIONID  
  ID="delegate_action_ID" />
```

Parameters

ID (required)
Delegation action ID.

WORKFLOWENGINE.DELETEFUNCTIONID

Deletes an existing function, and all its components, based on its ID.

Syntax

```
<WORKFLOWENGINE.DELETEFUNCTIONID  
  ID="workflow_function_ID" />
```

Parameters

ID (required)
Workflow function ID.

WORKFLOWENGINE.DELETEGROUPACTIONID

Deletes a group action based on its ID.

Syntax

```
<WORKFLOWENGINE.DELETEGROUPACTIONID  
  ID="workflow_group_action_ID" />
```

Parameters

ID (required)
Workflow group action ID.

WORKFLOWENGINE.DELETEGROUPID

Deletes a group based on its ID.

Syntax

```
<WORKFLOWENGINE.DELETEGROUPID  
  ID="workflow_group_ID" />
```

Parameters

ID (required)
Workflow group ID.

WORKFLOWENGINE.DELETEPROCESSID

Deletes an existing process, and all its components, based on its ID.

Syntax

```
<WORKFLOWENGINE.DELETEPROCESSID  
  ID="workflow_process_ID" />
```

Parameters

ID (required)
Workflow process ID.

WORKFLOWENGINE.DELETESTATEID

Deletes a state based on its ID.

Syntax

```
<WORKFLOWENGINE.DELETESTATEID  
  ID="workflow_state_ID" />
```

Parameters

ID (required)
Workflow state ID.

WORKFLOWENGINE.DELETESTEPACTIONID

Deletes a step action based on its ID.

Syntax

```
<WORKFLOWENGINE.DELETESTEPACTIONID  
  ID="workflow_step_action_ID" />
```

Parameters

ID (required)
Workflow step action ID.

WORKFLOWENGINE.EXECUTETIMEDACTIONS

Executes all timed actions that are due, and removes them from the queue.

Syntax

```
<WORKFLOWENGINE.EXECUTETIMEDACTIONS />
```

Parameters

None.

WORKFLOWENGINE.GETADDOBJECTTOGROUPCANDIDATEASSIGNEES

Generates a list of potential workflow participants (candidates), if the workflow process has a "Choose assignee when step is taken" assignment method for the Start Step.

Syntax

```
<WORKFLOWENGINE.GETADDOBJECTTOGROUPCANDIDATEASSIGNEES
  SITE="publication_ID"
  ID="workflow_group_ID"
  OBJECTVARNAME="participant_object" />
```

Parameters

SITE (required)

Publication ID.

ID (required)

Workflow group ID.

OBJVARNAME (required)

Name of the participants object that describes the potential workflow participants for an assignment. If no object is returned, the error message indicates that the starting process step is not a "Choose assignee when step is taken" assignment method.

Description

The WORKFLOWENGINE.GETADDOBJECTTOGROUPCANDIDATEASSIGNEES tag generates a list of workflow participants, which when modified appropriately, is passed as an ASSIGNMENTLIST object into the WORKFLOWENGINE.ADDNEWOBJECTTOGROUP tag.

Note the following when you use the ASSIGNMENTLIST parameter: when adding a new workflow step to a workflow process, the administrator needs to decide on the assignment method, which determines to whom the asset is assigned in the next workflow state. There are four assignment methods from which to choose:

- Retain "From State" assignees
- No assignments, control actions with function privs
- Assign from list of participants
- Choose assignee when step is taken

If the administrator selects the “Choose assignee when step is take” assignment method, then the workflow participant is presented with a list of possible assignees at the time of "Finish my assignment".

For more information about setting up a workflow process, see the *CSEE Administrator's Guide*.

WORKFLOWENGINE.GETALLFUNCTIONS

Loads all existing functions.

Syntax

```
<WORKFLOWENGINE.GETALLFUNCTIONS  
  PREFIX="variable_prefix" />
```

Parameters

PREFIX (required)

First part of the name of all variables and/or lists used by this operation. Variables are typically named with the prefix together with the field name. A delimiter is not automatically inserted after the prefix. Therefore, if you want a delimiter, such as a colon, you must specify it with the prefix. For example, *Variables.prefix:fieldname*.

Description

The `WORKFLOWENGINE.GETALLFUNCTIONS` tag returns the following variables:

- `PREFIX+Total` — the number of functions
- `PREFIXi` (where *i* is a number) — represents each object in the array.

WORKFLOWENGINE.GETALLPROCESSES

Loads all existing matching processes.

Syntax

```
<WORKFLOWENGINE.GETALLPROCESSES  
  PREFIX="variable_prefix"  
  [ASSETTYPE="assettype"]  
  [PUBID="pub_id"] />
```

Parameters

PREFIX (required)

First part of the name of all variables and/or lists used by this operation. Variables are typically named with the prefix together with the field name. A delimiter is not automatically inserted after the prefix. Therefore, if you want a delimiter, such as a colon, you must specify it with the prefix. For example, *Variables.prefix:fieldname*.

ASSETTYPE (optional)

Name of the asset type for which the workflow process is constructed.

PUBID (optional)
Publication ID.

Description

The `<WORKFLOWENGINE.GETALLPROCESSES>` tag returns the following variables:

- `PREFIX+Total` — the number of processes
- `PREFIXi` (where *i* is a number) — represents each object in the array.

WORKFLOWENGINE.GETALLSTATES

Loads all available states as objects named in an array of prefixed variables.

Syntax

```
<WORKFLOWENGINE.GETALLSTATES
  PREFIX="variable_prefix" />
```

Parameters

PREFIX (required)

First part of the name of all variables and/or lists used by this operation. Variables are typically named with the prefix together with the field name. A delimiter is not automatically inserted after the prefix. Therefore, if you want a delimiter, such as a colon, you must specify it with the prefix. For example, `Variables.prefix:fieldname`.

WORKFLOWENGINE.GETALLSTATESFORPROCESS

Loads all available states for a specific workflow process as objects named in an array of prefixed variables.

Syntax

```
<WORKFLOWENGINE.GETALLSTATESFORPROCESS
  PROCESS="process_ID"
  PREFIX="variable_prefix" />
```

Parameters

PROCESS (required)

Workflow process ID.

PREFIX (required)

First part of the name of all variables and/or lists used by this operation. Variables are typically named with the prefix together with the field name. A delimiter is not automatically inserted after the prefix. Therefore, if you want a delimiter, such as a colon, you must specify it with the prefix. For example, `Variables.prefix:fieldname`.

WORKFLOWENGINE.GETASSIGNMENTACTIONID

Loads an assignment action based on its ID.

Syntax

```
<WORKFLOWENGINE.GETASSIGNMENTACTIONID  
  ID="assignment_action_ID"  
  OBJVARNAME="assignment_action_object" />
```

Parameters

ID (required)
Assignment action ID.

OBJVARNAME (required)
Name of assignment action object to create.

WORKFLOWENGINE.GETASSIGNMENTACTIONNAME

Loads an assignment action based on its name.

Syntax

```
<WORKFLOWENGINE.GETASSIGNMENTACTIONNAME  
  NAME="assignment_action_name"  
  OBJVARNAME="assignment_action_object" />
```

Parameters

NAME (required)
Assignment action name.

OBJVARNAME (required)
Name of assignment action object to create.

WORKFLOWENGINE.GETASSIGNMENTID

Loads an assignment based on its ID.

Syntax

```
<WORKFLOWENGINE.GETASSIGNMENTID  
  ID="assignment_ID"  
  OBJVARNAME="assignment_object_name" />
```

Parameters

ID (required)
Workflow assignment ID.

OBJVARNAME (required)
Name of assignment object to create.

WORKFLOWENGINE.GETAVAILABLEASSIGNMENTACTIONS

Loads all available assignment actions as objects named in arrays of prefixed variables.

Syntax

```
<WORKFLOWENGINE.GETAVAILABLEASSIGNMENTACTIONS  
  PREFIX="variable_prefix" />
```

Parameters

PREFIX (required)

First part of the name of all variables and/or lists used by this operation. Variables are typically named with the prefix together with the field name. A delimiter is not automatically inserted after the prefix. Therefore, if you want a delimiter, such as a colon, you must specify it with the prefix. For example,
Variables.prefix:fieldname.

WORKFLOWENGINE.GETAVAILABLECONDITIONS

Loads all available conditions as objects named in arrays of prefixed variables.

Syntax

```
<WORKFLOWENGINE.GETAVAILABLECONDITIONS  
  PREFIX="variable_prefix" />
```

Parameters

PREFIX (required)

First part of the name of all variables and/or lists used by this operation. Variables are typically named with the prefix together with the field name. A delimiter is not automatically inserted after the prefix. Therefore, if you want a delimiter, such as a colon, you must specify it with the prefix. For example,
Variables.prefix:fieldname.

WORKFLOWENGINE.GETAVAILABLEDEADLOCKACTIONS

Loads all available deadlock actions as objects named in arrays of prefixed variables.

Syntax

```
<WORKFLOWENGINE.GETAVAILABLEDEADLOCKACTIONS  
  PREFIX="variable_prefix" />
```

Parameters

PREFIX (required)

First part of the name of all variables and/or lists used by this operation. Variables are typically named with the prefix together with the field name. A delimiter is not automatically inserted after the prefix. Therefore, if you want a delimiter, such as a colon, you must specify it with the prefix. For example,
Variables.prefix:fieldname.

WORKFLOWENGINE.GETAVAILABLEDELEGATEACTIONS

Loads all available delegation actions as objects named in arrays of prefixed variables.

Syntax

```
<WORKFLOWENGINE.GETAVAILABLEDELEGATEACTIONS  
  PREFIX="variable_prefix" />
```

Parameters

PREFIX (required)

First part of the name of all variables and/or lists used by this operation. Variables are typically named with the prefix together with the field name. A delimiter is not automatically inserted after the prefix. Therefore, if you want a delimiter, such as a colon, you must specify it with the prefix. For example, *Variables.prefix:fieldname*.

WORKFLOWENGINE.GETAVAILABLEGROUPACTIONS

Loads all available group actions as objects named in arrays of prefixed variables.

Syntax

```
<WORKFLOWENGINE.GETAVAILABLEGROUPACTIONS  
  PREFIX="variable_prefix" />
```

Parameters

PREFIX (required)

First part of the name of all variables and/or lists used by this operation. Variables are typically named with the prefix together with the field name. A delimiter is not automatically inserted after the prefix. Therefore, if you want a delimiter, such as a colon, you must specify it with the prefix. For example, *Variables.prefix:fieldname*.

WORKFLOWENGINE.GETAVAILABLEGROUPS

Loads all available groups as objects named in arrays of prefixed variables.

Syntax

```
<WORKFLOWENGINE.GETAVAILABLEGROUPS  
  PREFIX="groups_object"  
  [PUBID="site_ID"] />
```

Parameters

PREFIX (required)

First part of the name of all variables and/or lists used by this operation. Variables are typically named with the prefix together with the field name. A delimiter is not automatically inserted after the prefix. Therefore, if you want a delimiter, such as a

colon, you must specify it with the prefix. For example,
Variables.prefix:fieldname.

PUBID (optional)
 Publication ID.

WORKFLOWENGINE.GETAVAILABLESTEPACTIONS

Loads all available step actions as objects named in arrays of prefixed variables.

Syntax

```
<WORKFLOWENGINE.GETAVAILABLESTEPACTIONS
  PREFIX="step_action_object" />
```

Parameters

PREFIX (required)

First part of the name of all variables and/or lists used by this operation. Variables are typically named with the prefix together with the field name. A delimiter is not automatically inserted after the prefix. Therefore, if you want a delimiter, such as a colon, you must specify it with the prefix. For example,
Variables.prefix:fieldname.

WORKFLOWENGINE.GETBLOCKINGREASON

Gets the reason why the asset cannot move to the next state.

Syntax

```
<WORKFLOWENGINE.GETBLOCKINGREASON
  SITE="pub_ID"
  ID="assignment_ID"
  NEXTSTEP="workflow_step_object"
  VARNAME="variable_name" />
```

Parameters

SITE (required)
 Publication ID.

ID (required)
 Workflow assignment ID.

NEXTSTEP (required)
 Workflow step object.

VARNAME (required)
 Name of output variable to create.

errno

The possible values of `errno` include:

Value	Description
-12027	No asset assignment.
-12036	Wrong asset state.
-12028	Not logged in.
-12029	Missing user.
-12030	Insufficient privileges.

WORKFLOWENGINE.GETCOMPLETEASSIGNMENTCANDIDATEASSIGNEES

Generates a list of potential workflow participants if the process has a "Choose assignee when step is taken" type start step.

Syntax

```
<WORKFLOWENGINE.GETCOMPLETEASSIGNMENTCANDIDATEASSIGNEES
  SITE="pub_ID"
  NEXTSTEP="next_step_ID"
  OBJVARNAME="participants_object_name" />
```

Parameters

SITE (required)

Publication ID.

NEXTSTEP (required)

Next workflow step ID.

OBJVARNAME

Name of the participants object that describes the potential workflow participants for the assignment. If no object is returned, the starting process step does not use a "Choose assignee when step is taken" assignment method.

Description

The `WORKFLOWENGINE.GETCOMPLETEASSIGNMENTCANDIDATEASSIGNEES` tag generates a candidate list, which when modified appropriately, is passed as an `ASSIGNMENTLIST` object into [WORKFLOWENGINE.COMPLETEASSIGNMENT](#) tag.

Note the following when you use the `ASSIGNMENTLIST` parameter; when adding a new workflow step to a workflow process, the administrator needs to decide on the assignment method, which determines to whom the asset is assigned in the next workflow state. There are four assignment methods from which to choose:

- Retain "From State" assignees
- No assignments, control actions with function privs

- Assign from list of participants
- Choose assignee when step is taken

If the administrator selects the “Choose assignee when step is take” assignment method, then the workflow participant is presented with a list of possible assignees at the time of "Finish my assignment".

For more information about setting up a workflow process, see the *CSEE Administrator's Guide*.

WORKFLOWENGINE.GETCONDITIONID

Loads a condition based on its ID.

Syntax

```
<WORKFLOWENGINE.GETCONDITIONID
  ID="condition_id"
  OBJVARNAME="condition_object_name" />
```

Parameters

ID (required)
Condition ID.

OBJVARNAME (required)
Name of condition object to create.

WORKFLOWENGINE.GETCONDITIONNAME

Loads a condition based on its name.

Syntax

```
<WORKFLOWENGINE.GETCONDITIONNAME
  NAME="condition_name"
  OBJVARNAME="conditon_object_name" />
```

Parameters

NAME(required)
Name of condition.

OBJVARNAME (required)
Name of condition object to create.

WORKFLOWENGINE.GETDEADLOCKACTIONID

Loads a deadlock action based on its ID.

Syntax

```
<WORKFLOWENGINE.GETDEADLOCKACTIONID  
  ID="deadlock_action_ID"  
  OBJVARNAME="deadlock_action_object_name" />
```

Parameters

ID (required)
Deadlock action ID.

OBJVARNAME (required)
Name of deadlock action object to create.

WORKFLOWENGINE.GETDEADLOCKACTIONNAME

Loads a deadlock action based on its name.

Syntax

```
<WORKFLOWENGINE.GETDEADLOCKACTIONNAME  
  NAME="deadlock_action_name"  
  OBJVARNAME="deadlock_action_object_name" />
```

Parameters

NAME (required)
Name of deadlock action.

OBJVARNAME (required)
Name of deadlock action object to create.

WORKFLOWENGINE.GETDELEGATEACTIONID

Loads a delegation action based on its ID.

Syntax

```
<WORKFLOWENGINE.GETDELEGATEACTIONID  
  ID="delegation_action_ID"  
  OBJVARNAME="delegation_action_object_name" />
```

Parameters

ID (required)
Delegation action ID.

OBJVARNAME (required)
Name of delegation action object to create.

WORKFLOWENGINE.GETDELEGATEACTIONNAME

Loads a delegation action based on its name.

Syntax

```
<WORKFLOWENGINE.GETDELEGATEACTIONNAME  
  NAME="delegation_action_name"  
  OBJVARNAME="delegation_action_object_name" />
```

Parameters

NAME (required)

Name of delegation action.

OBJVARNAME (required)

Name of delegation action object to create.

WORKFLOWENGINE.GETDELEGATEECANDIDATES

Loads the workflow participants (candidates) who are the potential recipients of an assignment and stores the user IDs as a list.

Syntax

```
<WORKFLOWENGINE.GETDELEGATEECANDIDATES  
  SITE="pub_ID"  
  ID="assignment_ID"  
  LISTVARNAME="user_ID_list_name" />
```

Parameters

SITE (required)

Publication ID.

ID (required)

Workflow assignment ID.

LISTVARNAME (required)

Name of user ID list to create. This list contains the column ITEM.

WORKFLOWENGINE.GETFILTEREDASSIGNMENTS

Loads a list of assignments that match the specified criteria.

Syntax

```
<WORKFLOWENGINE.GETFILTEREDASSIGNMENT
  PREFIX="variable_prefix"
  [OBJECT="workflow_asset_object"]
  [SITE="pub_ID"]
  [STATUS="status1,status2"]
  [NOTSTATUS="notstatus1,notstatus2"]
  [DISPOSITION="disposition1,disposition2"]
  [STATEID="stateid1,stateid2"]
  [GROUPID="groupid1,groupid2"]
  [ASSIGNEDTO="list_name"]
  [NOTASSIGNEDTO="list_name"]
  [ASSIGNEDBY="list_name"]
  [DISPOSER="list_name"]
  [ASSIGNEDDATEBEFORE="date_object"]
  [ASSIGNEDDATEAFTER="date_object"]
  [DISPOSEDDATEBEFORE="date_object"]
  [DISPOSEDDATEAFTER="date_object"]
  [OVERDUE="true|false"]
  [OVERDUEMINS="integer1,integer2"]
  [DEADLOCKED="true|false"]
  [DESCENDING="true|false"]
  [SORTTYPE="sort_type"] />
```

Parameters

PREFIX (required)

First part of the name of all variables and/or lists used by this operation. Variables are typically named with the prefix together with the field name. A delimiter is not automatically inserted after the prefix. Therefore, if you want a delimiter, such as a colon, you must specify it with the prefix. For example, *Variables.prefix:fieldname*.

OBJECT (optional)

Workflow asset object.

SITE (optional)

Publication ID.

STATUS (optional)

Limits resultset to assignments with the specified status(es). Comma-separated list. Valid values are:

- active
- notactive
- queued
- abstain

NOTSTATUS (optional)

Limits resultset to assignments that do not have the specified status(es). Comma-separated list. Valid values are:

- active
- notactive
- queued
- abstain

DISPOSITION (optional)

Limits resultset to assignments with the specified assignment disposition. Comma-separated list. Valid values are:

- completed
- cancelled
- delegated
- unknown

STATEID (optional)

Limits resultset to assignments with the specified state ID. Comma-separated list of state IDs.

GROUPID (optional)

Limits resultset to assignments with the specified group ID. Comma-separated list of workflow group IDs.

ASSIGNEDTO (optional)

Limits resultset to assignments assigned to specified users. List name. Includes a column, ITEM, that contains user IDs.

NOTASSIGNEDTO (optional)

Limits resultset to assignments not assigned to specified users. List name. Includes a column, ITEM, that contains user IDs

ASSIGNEDBY (optional)

List name. Includes a column, ITEM, that contains user IDs

DISPOSER (optional)

Limits resultset to assignments with specified disposer. A disposer is the workflow participant who makes decisions about an assignment, such as choosing “Finishing my assignment” in the Content Server interface. List name. Includes a column, ITEM, that contains user IDs.

ASSIGNEDDATEBEFORE (optional)

Limits resultset to assignments assigned before a specified date. Date object.

ASSIGNEDDATEAFTER (optional)

Limits resultset to assignments assigned after a specified date. Date object.

DISPOSEDDATEBEFORE (optional)

Limits resultset to assignments completed (“disposed”) before a specified date. Date object.

DISPOSEDDATEAFTER (optional)

Limits resultset to assignments completed (“disposed”) after a specified date. Date object.

OVERDUE (optional)

Limits resultset to assignments that are overdue. If true, the tag returns only the assignments that have passed deadline.

OVERDUEMINS (optional)

Limits resultset to assignments that are overdue in a specified number of minutes. Comma-separated list of signed integers interpreted as "deadline between the current time and the current time plus the overdue minutes." For example, OVERDUEMINS="-10,20" means that the returned assignments should be either due within the next 10 minutes or should have already passed the deadline by up to 20 minutes.

DEADLOCKED (optional)

Limits resultset to assignments that are deadlocked. Valid values are `true` or `false`.

DESCENDING (optional)

Allows you to sort the resultset by descending order. Valid values are `true` or `false`.

SORTTYPE (optional)

Allows you to specify the sort type of the resultset. Valid values are `null` and `assigneddate`.

Description

The `WORKFLOWENGINE.GETFILTEREDASSIGNMENT` tag loads a list of assignments that match the specified criteria. Each parameter (except for `DESCENDING` and `SORTTYPE`) is one criteria by which you can limit the assignments returned in the resultset.

WORKFLOWENGINE.GETFUNCTIONID

Loads an existing workflow function based on its ID.

Syntax

```
<WORKFLOWENGINE.GETFUNCTIONID  
  ID="workflow_function_ID"  
  OBJVARNAME="workflow_function_object_name" />
```

Parameters

ID (required)

Workflow function ID.

OBJVARNAME (required)

Name of workflow function object to create.

WORKFLOWENGINE.GETFUNCTIONNAME

Loads an existing workflow function based on its name.

Syntax

```
<WORKFLOWENGINE.GETFUNCTIONNAME  
  NAME="workflow_function_name"  
  OBJVARNAME="workflow_function_object_name" />
```

Parameters

NAME (required)
Name of workflow function.

OBJVARNAME (required)
Name of workflow function object to create.

WORKFLOWENGINE.GETGROUPACTIONID

Loads a group action based on its ID.

Syntax

```
<WORKFLOWENGINE.GETGROUPACTIONID  
  ID="group_action_ID"  
  OBJVARNAME="group_action_object_name" />
```

Parameters

ID (required)
Workflow group action ID.

OBJVARNAME (required)
Name of workflow group action object to create.

WORKFLOWENGINE.GETGROUPACTIONNAME

Loads a group action based on its name.

Syntax

```
<WORKFLOWENGINE.GETGROUPACTIONNAME  
  NAME="group_action_name"  
  OBJVARNAME="group_action_object_name" />
```

Parameters

NAME (required)
Name of workflow group action.

OBJVARNAME (required)
Name of workflow group action object to create.

WORKFLOWENGINE.GETGROUPID

Loads a group based on its ID.

Syntax

```
<WORKFLOWENGINE.GETGROUPID
  ID="workflow_group_ID"
  OBJVARNAME="workflow_group_object_name" />
```

Parameters

ID (required)
Workflow group ID.

OBJVARNAME (required)
Name of workflow group object to create.

WORKFLOWENGINE.GETGROUPMEMBERS

Gets the members of the specified group.

Syntax

```
<WORKFLOWENGINE.GETGROUPMEMBERS
  GROUPID="group_ID"
  PREFIX="variable_prefix" />
```

Parameters

GROUPID
ID of the group.

PREFIX
First part of the name of all variables and/or lists used by this operation. Variables are typically named with the prefix together with the field name. A delimiter is not automatically inserted after the prefix. Therefore, if you want a delimiter, such as a colon, you must specify it with the prefix. For example, *Variables.prefix:fieldname*.

Description

The `WORKFLOWENGINE.GETGROUPMEMBERS` tag returns the following variables:

- Prefix+"Total" — the count of workflowable objects.
- Prefix+I — the name of the object itself.

WORKFLOWENGINE.GETGROUPMEMBERSHIP

Obtains the group identifier (if any) for the specified object.

Syntax

```
<WORKFLOWENGINE.GETGROUPMEMBERSHIP  
  OBJECT="workflow_asset_object"  
  VARNAME="variable_name" />
```

Parameters

OBJECT (required)

Name of the workflow asset object.

VARNAME (required)

Name of the variable to create. This variable contains the group ID.

WORKFLOWENGINE.GETGROUPNAME

Loads a group based on its name.

Syntax

```
<WORKFLOWENGINE.GETGROUPNAME  
  NAME="group_name"  
  OBJVARNAME="object_name" />
```

Parameters

NAME (required)

Name of workflow group.

OBJVARNAME (required)

Name of workflow group object to create.

WORKFLOWENGINE.GETLEGALNEXTWORKFLOWSTEPS

Gets a prefixed array of allowable steps that can be taken next.

Syntax

```
<WORKFLOWENGINE.GETLEGALNEXTWORKFLOWSTEPS  
  SITE="pub_ID"  
  OBJECT="workflow_asset_object"  
  PREFIX="prefix_name"  
  [ROLE="role_ID"] />
```

Parameters

SITE (required)

Publication ID.

OBJECT (required)

Workflow asset object.

PREFIX (required)

First part of the name of all variables and/or lists used by this operation. Variables are typically named with the prefix together with the field name. A delimiter is not automatically inserted after the prefix. Therefore, if you want a delimiter, such as a colon, you must specify it with the prefix. For example, `Variables.prefix:fieldname`.

ROLE (optional)

Role ID.

errno

The possible values of `errno` include:

Value	Description
-12028	Not logged in.
-12029	Missing user.

WORKFLOWENGINE.GETMULTISITEPROCESSES

Loads all existing processes.

Syntax

```
<WORKFLOWENGINE.GETMULTISITEPROCESSES
  [ASSETTYPE="asset_type"]
  PUBIDS="site_ID1,site_ID2"
  PREFIX="prefix_name" />
```

Parameters**ASSETTYPE (optional)**

An asset type.

PUBIDS (required)

Comma-separated list of site IDs.

PREFIX (required)

First part of the name of all variables and/or lists used by this operation. Variables are typically named with the prefix together with the field name. A delimiter is not automatically inserted after the prefix. Therefore, if you want a delimiter, such as a colon, you must specify it with the prefix. For example, `Variables.prefix:fieldname`.

Description

The `<WORKFLOWENGINE.GETMULTISITEPROCESSES>` tag returns the following variables:

- `PREFIX+Total` — the number of processes
- `PREFIXi` (where *i* is a number) — represents each object in the array.

WORKFLOWENGINE.GETOBJECTHISTORY

Retrieves the assignment history for an object as an array of prefixed variables.

Syntax

```
<WORKFLOWENGINE.GETOBJECTHISTORY  
  OBJECT="workflow_asset_object_name"  
  PREFIX="variablePrefix" />
```

Parameters

OBJECT (required)

Name of the workflow asset object.

PREFIX (required)

First part of the name of all variables and/or lists used by this operation. Variables are typically named with the prefix together with the field name. A delimiter is not automatically inserted after the prefix. Therefore, if you want a delimiter, such as a colon, you must specify it with the prefix. For example, *Variables.prefix:fieldname*.

WORKFLOWENGINE.GETOBJECTPARTICIPANTS

Retrieves the participants for a workflow asset object.

Syntax

```
<WORKFLOWENGINE.GETOBJECTPARTICIPANTS  
  OBJECT="workflow_asset_object_name"  
  OBJVARNAME="participants_object" />
```

Parameters

OBJECT (required)

Name of the workflow asset object.

OBJVARNAME (required)

Name of participants object to create.

WORKFLOWENGINE.GETOBJECTSTATEID

Gets the state ID associated with the specified object in the workflow.

Syntax

```
<WORKFLOWENGINE.GETOBJECTSTATEID  
  OBJECT="specified_workflow_object_name"  
  VARNAME="variable_name" />
```

Parameters

OBJECT (required)
Name of the specified workflow object.

VARNAME (required)
Name of output variable.

WORKFLOWENGINE.GETOBJECTWORKFLOWPROCESS

Gets the workflow process ID associated with the specified object.

Syntax

```
<WORKFLOWENGINE.GETOBJECTWORKFLOWPROCESS  
  OBJECT="specified_workflow_object_name"  
  VARNAME="variable_name" />
```

Parameters

OBJECT (required)
Name of the specified workflow object.

VARNAME (required)
Name of the output variable to create.

WORKFLOWENGINE.GETPOTENTIALPARTICIPANTS

Gets the potential participants for a specific workflow.

Syntax

```
<WORKFLOWENGINE.GETPOTENTIALPARTICIPANTS  
  SITE="pub_ID"  
  PROCESS="workflow_process_ID"  
  OBJVARNAME="participant_object_name" />
```

Parameters

SITE (required)
Publication ID.

PROCESS (required)

Workflow process ID.

OBJVARNAME (required)

Name of the participant object to create.

errno

The possible values of errno include:

Value	Description
-12025	Missing process.

WORKFLOWENGINE.GETPROCESSID

Loads an existing workflow process based on its ID.

Syntax

```
<WORKFLOWENGINE.GETPROCESSID
  OBJVARNAME="object_name"
  ID="workflow_process_id" />
```

Parameters

OBJVARNAME (required)

Name of output object variable. This variable contains the workflow process.

ID (required)

Workflow process ID.

errno

The possible values of errno include:

Value	Description

WORKFLOWENGINE.GETPROCESSNAME

Loads an existing workflow process based on its name.

Syntax

```
<WORKFLOWENGINE.GETPROCESSNAME
  NAME="workflow_process_name"
  OBJVARNAME="object_name" />
```

Parameters

OBJVARNAME (required)

Name of output object variable; in this case, a workflow process.

NAME (required)
Name of the workflow process.

errno

The possible values of errno include:

Value	Description

WORKFLOWENGINE.GETSTARTWORKFLOWCANDIDATEASSIGNEES

Gets the potential workflow participants (“candidate assignees”) for the start step.

Syntax

```
<WORKFLOWENGINE.GETPROCESSNAME
  OBJVARNAME="participant_object"
  SITE="pub_ID"
  ID="workflow_process_ID" />
```

Parameters

OBJVARNAME (required)
Name of output object variable; in this case, a participants object.

SITE (required)
Publication ID

ID (required)
Workflow process ID.

WORKFLOWENGINE.GETSTATEID

Loads a state based on its ID.

Syntax

```
<WORKFLOWENGINE.GETSTATEID
  ID="state_ID"
  OBJVARNAME="state_object_name" />
```

Parameters

ID (required)
Workflow state ID.

OBJVARNAME (required)
Name of the workflow state object to create.

WORKFLOWENGINE.GETSTATENAME

Loads a state based on its name.

Syntax

```
<WORKFLOWENGINE.GETSTATENAME  
  NAME="state_name"  
  OBJVARNAME="state_object_name" />
```

Parameters

NAME (required)

Name of workflow state.

OBJVARNAME (required)

Name of workflow state object to create.

WORKFLOWENGINE.GETSTEPACTIONID

Loads a step action based on its ID.

Syntax

```
<WORKFLOWENGINE.GETSTEPACTIONID  
  ID="step_action_ID"  
  OBJVARNAME="step_action_object_name" />
```

Parameters

ID (required)

Step action ID.

OBJVARNAME (required)

Name of step action object to create.

WORKFLOWENGINE.GETSTEPACTIONNAME

Loads a step action based on its name.

Syntax

```
<WORKFLOWENGINE.GETSTEPACTIONNAME  
  NAME="step_action_name"  
  OBJVARNAME="object_name" />
```

Parameters

NAME (required)

Name of the step action.

OBJVARNAME (required)

Name of the step action object to create.

WORKFLOWENGINE.GETSTEPID

Loads a workflow step object, based on its ID.

Syntax

```
<WORKFLOWENGINE.GETSTEPID
  ID="workflow_step_ID"
  OBJVARNAME="object_name" />
```

Parameters

ID
Workflow step ID.

OBJVARNAME
Name of the workflow step object to create.

WORKFLOWENGINE.ISFUNCTIONLEGAL

Verifies whether the current user on the specified site is allowed to execute the specified function.

Syntax

```
<WORKFLOWENGINE.ISFUNCTIONLEGAL
  SITE="pub_ID"
  OBJECT="workflow_asset_object"
  VARNAME="variable_name"
  FUNCTIONNAME="workflow_function"
  MUSTBEASSIGNED="true|false" />
```

Parameters

SITE (required)
Publication ID.

OBJECT (required)
Workflow asset object.

VARNAME (required)
Name of the output variable to create. The variable contains `true` if the function is allowed or `false` if the function is not allowed.

FUNCTIONNAME (required)
Name of function.

MUSTBEASSIGNED (optional)
If set to `true`, then unassigned objects do not return any legal functions. If set to `false`, then unassigned objects return legal values. The default value is `false`.

WORKFLOWENGINE.ISBJECTASSIGNED

Verifies whether the specified object is assigned to a specified user.

Syntax

```
<WORKFLOWENGINE.ISBJECTASSIGNED
  OBJECT="workflow_asset_object_name"
  VARNAME="variable_name" />
```

Parameters

OBJECT (required)

Name of the workflow asset object

VARNAME (required)

Name of the output variable to create. The variable contains `true` if the object is assigned or `false` if the object is not assigned.

WORKFLOWENGINE.ISBJECTASSIGNEDTOGROUP

Verifies whether the specified object is assigned to any group.

Syntax

```
<WORKFLOWENGINE.ISBJECTASSIGNEDTOGROUP
  OBJECT="workflow_asset_object_name"
  VARNAME="variable_name" />
```

Parameters

OBJECT (required)

Name of the workflow asset object.

VARNAME (required)

Name of the output variable to create. The variable contains `true` if the object is assigned to a group or `false` if the object is not assigned to a group.

WORKFLOWENGINE.ISBJECTASSIGNEDTOUSER

Verifies whether the specified object is assigned to any user.

Syntax

```
<WORKFLOWENGINE.ISBJECTASSIGNEDTOUSER
  OBJECT="workflow_asset_object"
  [ASSIGNEDTO="name_of_user"]
  VARNAME="variable_name" />
```

Parameters

OBJECT (required)

Name of the workflow asset object.

ASSIGNEDTO (optional)

Name of the user that the object is assigned to.

VARNAME (required)

Name of the output variable to create. The variable contains `true` if the object is assigned to specified user or `false` if the object is not assigned to specified user.

WORKFLOWENGINE.ISOBJECTDEADLOCKED

Checks to see if an object is deadlocked.

Syntax

```
<WORKFLOWENGINE.ISOBJECTDEADLOCKED
  OBJECT="name of workflowable object"
  VARNAME="variable name" />
```

Parameters

OBJECT (required)

Name of the object from the WORKFLOWABLEOBJECT class.

VARNAME (required)

Name of the output variable to create. The variable contains `true` if the object is deadlocked or `false` if the object is not deadlocked.

WORKFLOWENGINE.REMOVEOBJECTFROMGROUP

Removes an asset from a workflow group.

Syntax

```
<WORKFLOWENGINE.REMOVEOBJECTFROMGROUP
  SITE="pub_ID"
  OBJECT="workflow_asset_object" />
```

Parameters

SITE (required)

Publication ID.

OBJECT (required)

Name of workflow asset object.

errno

The possible values of `errno` include:

Value	Description
-12049	Not in workflow.
-12031	Wrong group state.
-12048	Not in group.

WORKFLOWENGINE.SETASSIGNMENTACTION

Saves an assignment action.

Syntax

```
<WORKFLOWENGINE.SETASSIGNMENTACTION  
  OBJECT="assignment_action_object_name" />
```

Parameters

OBJECT (required)
Name of assignment action object.

WORKFLOWENGINE.SETASSIGNMENTDEADLINE

Sets a deadline for a workflow assignment.

Syntax

```
<WORKFLOWENGINE.SETASSIGNMENTDEADLINE  
  ID="assignment_ID"  
  DEADLINE="deadline" />
```

Parameters

ID (required)
Assignment ID.
DEADLINE (required)
Date object. Stored as time in milliseconds since midnight of 1/1/1970.

WORKFLOWENGINE.SETCONDITION

Saves a condition.

Syntax

```
<WORKFLOWENGINE.SETCONDITION  
  OBJECT="condition_object_name" />
```

Parameters

OBJECT (required)
Name of condition object.

WORKFLOWENGINE.SETDEADLOCKACTION

Saves a deadlock action.

Syntax

```
<WORKFLOWENGINE.SETDEADLOCKACTION  
  OBJECT="deadlock_action_object_ID" />
```

Parameters

OBJECT (required)
Deadlock action object ID.

WORKFLOWENGINE.SETDELEGATEACTION

Saves a delegation action.

Syntax

```
<WORKFLOWENGINE.SETDELEGATEACTION  
  OBJECT="delegation_action_object_name" />
```

Parameters

OBJECT (required)
Name of delegation action object.

WORKFLOWENGINE.SETFUNCTION

Saves a newly created or edited workflow function.

Syntax

```
<WORKFLOWENGINE.SETFUNCTION  
  OBJECT="workflow_function_object" />
```

Parameters

OBJECT (required)
Name of workflow function object.

WORKFLOWENGINE.SETGROUP

Saves a group.

Syntax

```
<WORKFLOWENGINE.SETGROUP  
  OBJECT="workflow_group_object_name" />
```

Parameters

OBJECT (required)
Name of workflow group object.

WORKFLOWENGINE.SETGROUPACTION

Saves a group action.

Syntax

```
<WORKFLOWENGINE.SETGROUPACTION  
  OBJECT="workflow_group_action_object_name" />
```

Parameters

OBJECT (required)
Name of workflow group action object.

WORKFLOWENGINE.SETOBJECTPARTICIPANTS

Sets the participants for a workflow object.

Syntax

```
<WORKFLOWENGINE.SETOBJECTPARTICIPANTS  
  OBJECT="workflow_asset_object_name"  
  PARTICIPANTS="participants_object_name" />
```

Parameters

OBJECT (required)
Name of workflow asset object.
PARTICIPANTS (required)
Name of participants object.

WORKFLOWENGINE.SETPROCESS

Saves a newly created or edited workflow process.

Syntax

```
<WORKFLOWENGINE.SETPROCESS  
  OBJECT="workflow_process_object_name" />
```

Parameters

OBJECT (required)
Name of workflow process object.

WORKFLOWENGINE.SETSTATE

Saves a state.

Syntax

```
<WORKFLOWENGINE.SETSTATE  
  OBJECT="workflow_state_object_name" />
```

Parameters

OBJECT (required)
Name of workflow state object.

WORKFLOWENGINE.SETSTEPACTION

Saves a step action.

Syntax

```
<WORKFLOWENGINE.SETSTEPACTION  
  OBJECT="step_action_object_name" />
```

Parameters

OBJECT (required)
Name of step action object.

WORKFLOWENGINE.STARTOBJECTWORKFLOWPROCESS

Starts the workflow for an object.

Syntax

```
<WORKFLOWENGINE . STARTOBJECTWORKFLOWPROCESS
    SITE="pub_ID"
    OBJECT="object_ID"
    ID="workflow_process_ID"
    [COMMENT="comment"]
    [DEADLINE="assignment_deadline"]
    [ASSIGNMENTLIST="participant_object"]
    [ABSDEADLINE="workflow_process_deadline"] />
```

Parameters

SITE (required)

Publication ID.

OBJECT (required)

Name of the workflow asset object.

ID (required)

Workflow process ID.

COMMENT (optional)

User's comments.

DEADLINE (optional)

Date object containing the assignment deadline, which is the deadline for the particular step.

ASSIGNMENTLIST (optional)

Participant object for the "Choose assignee when step is taken" assignment method. Required for start state.

ABSDEADLINE (optional)

Date object which contains the absolute deadline, which is the deadline for the workflow process.

Description

The `WORKFLOWENGINE.STARTOBJECTWORKFLOWPROCESS` tag starts the workflow for an object

Note the following when you use the `ASSIGNMENTLIST` parameter: when adding a new workflow step to a workflow process, the administrator needs to decide on the assignment method, which determines to whom the asset is assigned in the next workflow state. There are four assignment methods from which to choose:

- Retain "From State" assignees
- No assignments, control actions with function privs
- Assign from list of participants
- Choose assignee when step is taken

If the administrator selects the “Choose assignee when step is taken” assignment method, then the workflow participant is presented with a list of possible assignees at the time of "Finish my assignment".

For more information about setting up a workflow process, see the *CSEE Administrator's Guide*.

Workflow Function Tags

This group of tags are used to manage the workflow function actions.

Note

You create a workflow function object by using the [WORKFLOWENGINE.CREATENEWFUNCTION](#) tag.

The following section describes each workflow function tag, beginning with [WORKFLOWFUNCTION.GATHER](#).

WORKFLOWFUNCTION.GATHER

Gathers values for fields in an object from the name/value pairs of variables defined in the environment.

Syntax

```
<WORKFLOWFUNCTION.GATHER
  NAME="workflow function object name"
  PREFIX="variablePrefix" />
```

Parameters

NAME (required)

Name of the workflow function instance object whose fields you want to gather.

PREFIX (required)

First part of the name of all variables and/or lists used by this gather operation. Variables are typically named with the prefix together with the field name. If a delimiter, such as a colon, was used in the creation of the variables that you want to gather, you must use the same delimiter in the gather operation. For example, `Variables.prefix:fieldname`.

Description

The `WORKFLOWFUNCTION.GATHER` tag gathers values for fields in an object from the name/value pairs of variables defined in the environment. The `WORKFLOWFUNCTION.GATHER` tag look for the following variables:

- `prefix+id`
- `prefix+name`
- `prefix+description`

WORKFLOWFUNCTION.GETDESCRIPTION

Gets the description of the workflow function.

Syntax

```
<WORKFLOWFUNCTION.GETDESCRIPTION  
  NAME="workflow function object name"  
  VARNAME="variable name" />
```

Parameters

NAME (required)
Name of workflow function object.

VARNAME (required)
Name of output variable to create.

WORKFLOWFUNCTION.GETID

Gets the ID of the workflow function.

Syntax

```
<WORKFLOWFUNCTION.GETID  
  NAME="workflow function object name"  
  VARNAME="variable name" />
```

Parameters

NAME (required)
Name of workflow function object.

VARNAME (required)
Name of output variable to create.

WORKFLOWFUNCTION.GETNAME

Gets the function name of the workflow function.

Syntax

```
<WORKFLOWFUNCTION.GETNAME  
  NAME="workflow function object name"  
  VARNAME="variable name" />
```

Parameters

NAME (required)
Name of workflow function object.

VARNAME (required)
Name of output variable to create.

WORKFLOWFUNCTION.SCATTER

Creates independent variables and lists from an in-memory workflow function instance.

Syntax

```
<WORKFLOWFUNCTION.SCATTER  
  NAME="workflow function object name"  
  PREFIX="variable name" />
```

Parameters

NAME (required)

Name of the workflow function instance object whose fields are to be scattered.

PREFIX (required)

First part of the name of all variables and/or lists used by this scatter operation. Variables are typically named with the prefix together with the field name. A delimiter is not automatically inserted after the prefix. Therefore, if you want a delimiter, such as a colon, you must specify it with the prefix. For example, Variables.prefix:fieldname.

Description

The WORKFLOWFUNCTION.SCATTER tag sets the following variables:

- prefix+id
- prefix+name
- prefix+description

WORKFLOWFUNCTION.SETDESCRIPTION

Sets the description of the workflow function.

Syntax

```
<WORKFLOWFUNCTION.SETDESCRIPTION  
  NAME="workflow function object name"  
  VALUE="description" />
```

Parameters

NAME (required)

Name of the workflow function object.

VALUE (required)

Text of description.

WORKFLOWFUNCTION.SETNAME

Sets the name of the workflow function.

Syntax

```
<WORKFLOWFUNCTIONACTION.SETNAME  
    NAME="workflow function object name"  
    VALUE="function name" />
```

Parameters

NAME (required)
 Name of the function object.

VALUE (required)
 Function name.

Workflow Group Tags

Workflow group tags control workflow groups. A workflow group consists of individual assets that must go through the same workflow process. Assets in a workflow group must advance to the next state together. Workflow group deadlock occurs when two assets in the same group transition through different steps.

Note

You create a workflow group object by using the [WORKFLOWENGINE.CREATENEWGROUP](#) tag.

The following section describes each workflow group tag, beginning with [WORKFLOWGROUP.GATHER](#).

WORKFLOWGROUP.GATHER

Gathers values for fields in an object from the name/value pairs of variables defined in environment.

Syntax

```
<WORKFLOWGROUP.GATHER
  NAME="workflow group object"
  PREFIX="variable name" />
```

Parameters

NAME (required)

Name of the group instance object whose fields you want to gather.

PREFIX (required)

First part of the name of all variables and/or lists used by this gather operation. Variables are typically named with the prefix together with the field name. If a delimiter, such as a colon, was used in the creation of the variables that you want to gather, you must use the same delimiter in the gather operation. For example, `Variables.prefix:fieldname`.

Description

The `WORKFLOWGROUP.GATHER` tag looks for the following variables:

- `prefix+id`
- `prefix+name`
- `prefix+description`
- `prefix+process`
- `prefix+groupactions+actionTotal`
- `prefix+ groupactions+action+i` where *i* is an integer
- `prefix+editroles+roles`
- `prefix+deleteroles+roles`

- prefix+participants
- prefix+sites

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

Release Introduced

This tag was introduced in CSEE release 4.0.

WORKFLOWGROUP.GETDELETEROLES

Retrieves the roles that can add or remove a group for the workflow group.

Syntax

```
<WORKFLOWGROUP.GETDELETEROLES
  NAME="name of workflow group"
  OBJVARNAME="object name"/>
```

Parameters

NAME (required)

Name of the workflow group object.

OBJVARNAME (required)

Name of the role list object to be returned.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

Example

```
<WORKFLOWGROUP.GETDELETEROLES NAME="groupobj"
  OBJVARNAME="ACLLIST"/>
```

Release Introduced

This tag was introduced in CSEE release 4.0.

WORKFLOWGROUP.GETDESCRIPTION

Retrieves the description of the workflow group.

Syntax

```
<WORKFLOWGROUP.GETDESCRIPTION
  NAME="workflow group object name"
  VARNAME="variable name"/>
```

Parameters

NAME (required)
Name of the workflow group object.

VARNAME (required)
Name of the output variable.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

Example

```
<WORKFLOWGROUP.GETDESCRIPTION NAME="groupobj"
  VARNAME="description"/>
```

Release Introduced

This tag was introduced in CSEE release 4.0.

WORKFLOWGROUP.GETEDITROLES

Retrieves the roles that can edit or delete a group for the workflow group.

Syntax

```
<WORKFLOWGROUP.GETEDITROLES NAME="workflow group object name"
  OBJVARNAME="object name"/>
```

Parameters

NAME (required)
Name of the workflow group object.

OBJVARNAME (required)
Name of the role list object to be returned.

errno

The possible values of `errno` include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

Example

```
<WORKFLOWGROUP.GETEDITROLES NAME="groupobj"
  OBJVARNAME="ACLLIST"/>
```

Release Introduced

This tag was introduced in CSEE release 4.0.

WORKFLOWGROUP.GETGROUPACTIONS

Gets the group deadlock actions.

Syntax

```
<WORKFLOWGROUP.GETGROUPACTIONS
  NAME="workflow group object name"
  OBJVARNAME="object name" />
```

Parameters

NAME (required)

Name of the workflow group object.

OBJVARNAME (required)

Name of the workflow action list object to be returned.

errno

The possible values of `errno` include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

Release Introduced

This tag was introduced in CSEE release 4.0.

WORKFLOWGROUP.GETID

Gets the ID of the workflow group.

Syntax

```
<WORKFLOWGROUP.GETID
  NAME="group object name"
  VARNAME="variable name" />
```

Parameters

NAME (required)
Name of the workflow group object.

VARNAME (required)
Name of the output variable.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

Release Introduced

This tag was introduced in CSEE release 4.0.

WORKFLOWGROUP.GETPARTICIPANTS

Retrieves a list of workflow group participants.

Syntax

```
<WORKFLOWGROUP.GETPARTICIPANTS
  NAME="name of workflow group"
  OBJVARNAME="object name" />
```

Parameters

NAME (required)
Name of the workflow group object.

OBJVARNAME (required)
Name of the participants object to be returned.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

Example

```
<WORKFLOWGROUP.GETPARTICIPANTS NAME="groupobj"
  OBJVARNAME="participants"/>
```

Release Introduced

This tag was introduced in CSEE release 4.0.

WORKFLOWGROUP.GETNAME

Retrieves the name of the workflow group.

Syntax

```
<WORKFLOWGROUP.GETNAME
  NAME="object name"
  VARNAME="variable name"/>
```

Parameters

NAME (required)
Name of the workflow group object.

VARNAME (required)
Name of the output variable.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

Example

The following example retrieves the name of the groupobj group object.

```
<WORKFLOWGROUP.GETNAME NAME="groupobj"
  VARNAME="groupname"/>
```


Release Introduced

This tag was introduced in CSEE release 4.0.

WORKFLOWGROUP.GETSITES

Gets the sites for the workflow group.

Syntax

```
<WORKFLOWGROUP.GETSITES
  NAME="group object name"
  OBJVARNAME="object name" />
```

Parameters

NAME (required)

Name of the workflow group object.

OBJVARNAME (required)

Name of the site list object that is returned.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

Release Introduced

This tag was introduced in CSEE release 4.0.

WORKFLOWGROUP.GETWORKFLOWPROCESSID

Retrieves the ID of the workflow process for the group.

Syntax

```
<WORKFLOWGROUP.GETWORKFLOWPROCESSID
  NAME="process name"
  VARNAME="variable name" />
```

Parameters

NAME (required)

Name of the workflow group object.

VARNAME (required)

Name of the output variable.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

Example

```
<WORKFLOWGROUP.GETWORKFLOWPROCESSID NAME="groupobj"
    VARNAME="processid"/>
```

Release Introduced

This tag was introduced in CSEE release 4.0.

WORKFLOWGROUP.SCATTER

Creates independent variables and lists from an in-memory workflow group instance.

Syntax

```
<WORKFLOWGROUP.SCATTER
    NAME="workflow group object name"
    PREFIX="variable prefix" />
```

Parameters

NAME (required)

Name of the workflow group instance object whose fields are to be scattered.

PREFIX (required)

First part of the name of all variables and/or lists used by this scatter operation. Variables are typically named with the prefix together with the field name. A delimiter is not automatically inserted after the prefix. Therefore, if you want a delimiter, such as a colon, you must specify it with the prefix. For example, Variables.prefix:fieldname.

Description

The WORKFLOWGROUP.SCATTER tag sets the following variables:

- prefix+id
- prefix+name
- prefix+description
- prefix+process
- prefix+groupactions+actionTotal
- prefix+ groupactions+action+i where i is an integer
- prefix+editroles+roles

- prefix+deleteroles+roles
- prefix+participants
- prefix+sites

Release Introduced

This tag was introduced in CSEE release 4.0.

WORKFLOWGROUP.SETDELETEROLES

Sets the roles that can add or remove a group from the workflow group.

Syntax

```
<WORKFLOWGROUP.SETDELETEROLES
  NAME="workflow group object"
  OBJECT="object name" />
```

Parameters

NAME (required)
Name of workflow group object.

OBJECT (required)
Name of role list object.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

Release Introduced

This tag was introduced in CSEE release 4.0.

WORKFLOWGROUP.SETDESCRIPTION

Sets the description of the workflow group.

Syntax

```
<WORKFLOWGROUP.SETDESCRIPTION  
  NAME="workflow group object"  
  VALUE="description" />
```

Parameters

NAME (required)
Name of the workflow group object.

VALUE (required)
Description of the workflow group.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

Release Introduced

This tag was introduced in CSEE release 4.0.

WORKFLOWGROUP.SETEDITROLES

Sets the roles that can edit or delete a group from the workflow group.

Syntax

```
<WORKFLOWGROUP.SETEDITROLES  
  NAME="workflow group object"  
  OBJECT="object name" />
```

Parameters

NAME (required)
Name of the workflow group object.

OBJECT (required)
Name of the output object variable.

errno

The possible values of `errno` include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

Release Introduced

This tag was introduced in CSEE release 4.0.

WORKFLOWGROUP.SETGROUPACTIONS

Sets the group actions for the workflow group.

Syntax

```
<WORKFLOWGROUP.SETGROUPACTIONS
  NAME="workflow group object name"
  OBJECT="object name" />
```

Parameters

NAME (required)

Name of the workflow group object.

OBJECT (required)

Name of the workflow action list object.

errno

The possible values of `errno` include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

WORKFLOWGROUP.SETNAME

Sets the name of the workflow group.

Syntax

```
<WORKFLOWGROUP.SETNAME  
    NAME="group object name"  
    VALUE="new group name" />
```

Parameters

NAME (required)
Name of the workflow group object.

VALUE (required)
Name of the workflow group.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

Release Introduced

This tag was introduced in CSEE release 4.0.

WORKFLOWGROUP.SETPARTICIPANTS

Sets the participants object, which describes the participants of the group.

Syntax

```
<WORKFLOWGROUP.SETPARTICIPANTS  
    NAME="group object name"  
    OBJECT="participant object" />
```

Parameters

NAME (required)
Name of the workflow group object.

OBJECT (required)
Name of the participants object.

errno

The possible values of `errno` include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

Release Introduced

This tag was introduced in CSEE release 4.0.

WORKFLOWGROUP.SETSITES

Sets the sites for the workflow group.

Syntax

```
<WORKFLOWGROUP.SETSITES
  NAME="group object name"
  OBJECT="sites object name" />
```

Parameters

NAME (required)

Name of the workflow group object.

OBJECT (required)

Name of the site list object.

errno

The possible values of `errno` include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

Release Introduced

This tag was introduced in CSEE release 4.0.

WORKFLOWGROUP.SETWORKFLOWPROCESSID

Sets the ID of the workflow process for the workflow group.

Syntax

```
<WORKFLOWGROUP.SETWORKFLOWPROCESSID  
    NAME="workflow group object name"  
    VALUE="process id" />
```

Parameters

NAME (required)

Name of the workflow group object.

VALUE (required)

Workflow process ID.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

Release Introduced

This tag was introduced in CSEE release 4.0.

Workflow Priv Tags

These tags manage workflowpriv objects. The following section describes each workflow priv tag, beginning with [WORKFLOWPRIV.GATHER](#).

WORKFLOWPRIV.GATHER

Gathers variables that have been named with the specified prefix convention into a workflow priv object.

Syntax

```
<WORKFLOWPRIV.GATHER
  NAME="workflowpriv_object_name"
  PREFIX="variable_prefix" />
```

Parameters

NAME (required)

Name of the priv instance object whose fields you want to gather.

PREFIX (required)

First part of the name of all variables and/or lists used by this gather operation. Variables are typically named with the prefix together with the field name. If a delimiter, such as a colon, was used in the creation of the variables that you want to gather, you must use the same delimiter in the gather operation. For example, *Variables.prefix:fieldname*.

Description

The `WORKFLOWPRIV.GATHER` tag gathers variables that have been named with the specified prefix convention into a workflow priv object.

The following variables are gathered into a workflow priv object:

- prefix+function
- prefix+state
- prefix+role
- prefix+allowed

WORKFLOWPRIV.GETALLOWED

Verifies whether the function is allowed for the privilege.

Syntax

```
<WORKFLOWPRIV.GETALLOWED  
  NAME="priv_object_name"  
  VARNAME="variable_name" />
```

Parameters

NAME (required)
 Name of the workflow priv object.

VARNAME (required)
 Name of the output variable to create.

WORKFLOWPRIV.GETFUNCTIONID

Gets the ID of the workflow function.

Syntax

```
<WORKFLOWPRIV.GETFUNCTIONID  
  NAME="workflow_priv_object"  
  VARNAME="variable_name" />
```

Parameters

NAME (required)
 Name of the workflow function.

VARNAME (required)
 Name of the output variable to create. This variable contains the workflow function ID.

WORKFLOWPRIV.GETROLE

Gets the ACL for which the workflow privilege is set.

Syntax

```
<WORKFLOWPRIV.GETROLE  
  NAME="workflowpriv_object_name"  
  VARNAME="variable_name" />
```

Parameters

NAME (required)
 Name of the workflow priv object.

VARNAME (required)
 Name of the output variable to create. This variable contains the ACL.

WORKFLOWPRIV.GETSTATEID

Gets the ID of the workflow state for which the workflow privilege is set.

Syntax

```
<WORKFLOWPRIV.GETSTATEID
  NAME="workflowpriv_object_name"
  VARNAME="variable_name" />
```

Parameters

NAME (required)

Name of the workflow priv object.

VARNAME (required)

Name of the output variable to create.

WORKFLOWPRIV.SCATTER

Writes (“scatters”) the loaded workflow priv object into variables with the following PREFIX naming convention.

Syntax

```
<WORKFLOWPRIV.SCATTER
  NAME="priv_object_name"
  PREFIX="variable_name" />
```

Parameters

NAME (required)

Name of the workflow priv object whose fields are to be scattered.

PREFIX (required)

First part of the name of all variables and/or lists used by this scatter operation.

Variables are typically named with the prefix together with the field name. A delimiter is not automatically inserted after the prefix. Therefore, if you want a delimiter, such as a colon, you must specify it with the prefix. For example,

Variables.*prefix:fieldname*.

Description

The <WORKFLOWPRIV.SCATTER> tag sets the following variables:

- prefix+function
- prefix+state
- prefix+role
- prefix+allowed

WORKFLOWPRIV.SETALLOWED

Sets whether the privilege is allowed.

Syntax

```
<WORKFLOWPRIV.SETALLOWED  
  NAME="workflowpriv_object_name"  
  ALLOWED="true|false" />
```

Parameters

NAME (required)

Name of the workflow priv object.

ALLOWED (required)

Specify true to set the privilege as “allowed” or false to disallow this privilege.
This attribute is stored as a Boolean value.

WORKFLOWPRIV.SETFUNCTIONID

Sets the ID of the workflow function for the workflow privilege.

Syntax

```
<WORKFLOWPRIV.SETFUNCTIONID  
  NAME="priv_object_name"  
  VALUE="workflow function ID" />
```

Parameters

NAME (required)

Name of the workflow priv object.

VALUE (required)

Workflow function ID.

WORKFLOWPRIV.SETROLE

Sets the role for the workflow privilege.

Syntax

```
<WORKFLOWPRIV.SETROLE  
  NAME="priv_object_name"  
  ROLE="role_name" />
```

Parameters

NAME (required)

Name of the priv object.

ROLE (required)

Name of the role to set.

WORKFLOWPRIV.SETSTATEID

Sets the ID of the workflow state for the priv.

Syntax

```
<WORKFLOWPRIV.SETSTATEID  
  NAME="priv_object_name"  
  VALUE="workflow state ID" />
```

Parameters

NAME (required)
 Name of the priv object.

VALUE (required)
 Workflow state ID.

Workflow Privs Tags

These tags manage workflow privs objects. The following section describes each workflow privs tag, beginning with [WORKFLOWPRIVS.ADD](#).

WORKFLOWPRIVS.ADD

Adds a workflow privilege to the workflow privs object.

Syntax

```
<WORKFLOWPRIVS.ADD  
  NAME="workflowprivs_object_name"  
  OBJECT="workflowpriv_object_name" />
```

Parameters

NAME (required)
 Name of the workflow privs object.

OBJECT (required)
 Name of the workflow priv object.

WORKFLOWPRIVS.CLEAR

Clears the workflow privs object.

Syntax

```
<WORKFLOWPRIVS.CLEAR  
  NAME="workflowprivs_object_name" />
```

Parameters

NAME (required)
 Name of the workflow privs object.

WORKFLOWPRIVS.CREATE

Creates a workflowpriv object.

Syntax

```
<WORKFLOWPRIVS.CREATE  
  NAME="workflowprivs_object_name"  
  OBJVARNAME="workflowpriv_object_name" />
```

Parameters

NAME (required)
 Name of the workflow privs object.

OBJVARNAME (required)
Name of workflow priv object to create

WORKFLOWPRIVS.GATHER

Gathers variables that have been named with the specified prefix convention into a workflow privs object.

Syntax

```
<WORKFLOWPRIVS.GATHER
  NAME="workflowprivs_object_name"
  PREFIX="variable_prefix" />
```

Parameters

NAME (required)
Name of the workflow privs object whose fields you want to gather.

PREFIX (required)
First part of the name of all variables and/or lists used by this gather operation. Variables are typically named with the prefix together with the field name. If a delimiter, such as a colon, was used in the creation of the variables that you want to gather, you must use the same delimiter in the gather operation. For example, Variables.*prefix:fieldname*.

Description

The <WORKFLOWPRIVS.GATHER> tag gathers the following variables into a workflow privs object:

- prefix+privTotal
- prefix+priv+i+function (where *i* is an integer)
- prefix+priv+i+state (where *i* is an integer)
- prefix+priv+i+role (where *i* is an integer)
- prefix+priv+i+allowed (where *i* is an integer)

WORKFLOWPRIVS.GETALLPRIVS

Gets all privileges.

Syntax

```
<WORKFLOWPRIVS.GETALLPRIVS
  NAME="workflow_privs_object_name"
  PREFIX="variable prefix" />
```

Parameters

NAME (required)
Name of the workflow priv object.

PREFIX (required)

The first part of the name of all variables and/or lists used by this gather operation. Variables are typically named with the prefix together with the field name. If a delimiter, such as a colon, was used in the creation of the variables that you want to gather, you must use the same delimiter in the gather operation. For example, *Variables.prefix:fieldname*.

Description

The WORKFLOWPRIVS.GETALLPRIVS tag sets the following variables:

- *prefix+Total* — # of objects
- *prefix+i* — individual object name; *i* = 0,1,2,...

WORKFLOWPRIVS.REMOVE

Removes a workflow privilege from a list of workflow privileges.

Syntax

```
<WORKFLOWPRIVS.REMOVE
  NAME="workflowprivs object name"
  OBJECT="workflowpriv object name" />
```

Parameters**NAME (required)**

Name of the workflow privs object.

OBJECT (required)

Name of the workflow priv object to remove.

WORKFLOWPRIVS.SCATTER

Writes ("scatters") the loaded workflow privs object into variables with the following PREFIX naming convention.

Syntax

```
<WORKFLOWPRIVS.SCATTER
  NAME="workflow_privs_object_name"
  PREFIX="variable_prefix" />
```

Parameters**NAME (required)**

The name of the workflow privs object whose fields are to be scattered.

PREFIX (required)

First part of the name of all variables and/or lists used by this scatter operation. Variables are typically named with the prefix together with the field name. A delimiter is not automatically inserted after the prefix. Therefore, if you want a delimiter, such as a colon, you must specify it with the prefix. For example, *Variables.prefix:fieldname*.

Description

The `WORKFLOWPRIVS.SCATTER` tag sets the following variables:

- `prefix+privTotal`
- `prefix+priv+i+function` (where *i* is an integer)
- `prefix+priv+i+state` (where *i* is an integer)
- `prefix+priv+i+role` (where *i* is an integer)
- `prefix+priv+i+allowed` (where *i* is an integer)

Workflow Process Tags

This group of tags manage the workflow processes.

Note

You create a workflow process object by using the [WORKFLOWENGINE.CREATENEWPROCESS](#) tag.

The following section describes each workflow process tag, beginning with [WORKFLOWPROCESS.ADDSTEP](#).

WORKFLOWPROCESS.ADDSTEP

Adds a step to a workflow process.

Syntax

```
<WORKFLOWPROCESS.ADDSTEP  
  NAME="workflow process object name"  
  OBJVARNAME="variable name"/>
```

Parameters

NAME (required)

Name of the workflow process object.

OBJVARNAME (required)

Name of the workflow step object.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

WORKFLOWPROCESS.CLEARSTEPS

Deletes all steps from a workflow process.

Syntax

```
<WORKFLOWPROCESS.CLEARSTEPS
  NAME="workflow process object name" />
```

Parameters

NAME (required)

Name of the workflow process object.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

WORKFLOWPROCESS.DELETESTEP

Deletes a step from a workflow process.

Syntax

```
<WORKFLOWPROCESS.DELETESTEP
  NAME="workflow process object name"
  VALUE="step ID" />
```

Parameters

NAME (required)

Name of the workflow process object.

VALUE (required)

ID of the step to be deleted.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

WORKFLOWPROCESS.GATHER

Gathers values for fields into an object from the name/value pairs of variables defined in the environment.

Syntax

```
<WORKFLOWPROCESS.GATHER  
  NAME="workflow process object name"  
  PREFIX="variable name" />
```

Parameters

NAME (required)

Name of the workflow process instance object.

PREFIX (required)

First part of the name of all variables and/or lists used by this scatter operation. Variables are typically named with the prefix together with the field name. A delimiter is not automatically inserted after the prefix. Therefore, if you want a delimiter, such as a colon, you must specify it with the prefix. For example, `Variables.prefix:fieldname`.

Description

The WORKFLOWPROCESS.GATHER tag looks for the following variables:

- prefix+id
- prefix+name
- prefix+description

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

WORKFLOWPROCESS.GETADMINROLES

Gets roles that represent administrators for the workflow process.

Syntax

```
<WORKFLOWPROCESS.GETADMINROLES
  NAME="workflow process object name"
  OBJVARNAME="variable name" />
```

Parameters

NAME (required)

Name of the workflow process object.

OBJVARNAME (required)

Name of the role list object variable.

Description

Participants with these roles are allowed to delegate assignments belonging to other people.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

WORKFLOWPROCESS.GETDELEGATIONACTIONS

Gets delegation actions from a workflow process.

Syntax

```
<WORKFLOWPROCESS.GETDELEGATIONACTIONS
  NAME="workflow process object name"
  OBJVARNAME="variable name" />
```

Parameters

NAME (required)

Name of the workflow process object.

OBJVARNAME (required)

Name of the workflow action list object variable.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

WORKFLOWPROCESS.GETFUNCTIONPRIVS

Gets the function privileges for a workflow process.

Syntax

```
<WORKFLOWPROCESS.GETFUNCTIONPRIVS
  NAME="workflow process object name"
  OBJVARNAME="variable name" />
```

Parameters

NAME (required)

Name of the workflow process object.

OBJVARNAME (required)

Name of the workflow privs object.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

WORKFLOWPROCESS.GETID

Gets the ID of a workflow process.

Syntax

```
<WORKFLOWPROCESS.GETID
  NAME="workflow process object name"
  VARNAME="variable name" />
```

Parameters

NAME (required)

Name of workflow process object.

VARNAME (required)

Name of the output variable to create.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

WORKFLOWPROCESS.GETINITIALSTEPID

Gets the start step ID of a workflow process.

Syntax

```
<WORKFLOWPROCESS.GETINITIALSTEPID
  NAME="workflow process object name"
  VARNAME="variable name" />
```

Parameters

NAME (required)

Name of the workflow process object.

VARNAME (required)

Name of the output variable to create.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

WORKFLOWPROCESS.GETPROCESSDESCRIPTION

Gets the description of the workflow process.

Syntax

```
<WORKFLOWPROCESS.GETPROCESSDESCRIPTION
  NAME="workflow process object name"
  VARNAME="variable name" />
```

Parameters

NAME (required)

Name of the workflow process object.

VARNAME (required)

Name of the output variable to create.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

WORKFLOWPROCESS.GETPROCESSNAME

Gets the name of a workflow process.

Syntax

```
<WORKFLOWPROCESS.GETPROCESSNAME
  NAME="workflow process object name"
  VARNAME="variable name" />
```

Parameters

NAME (required)

Name of workflow process object.

VARNAME (required)

Name of output variable to create.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

WORKFLOWPROCESS.GETROLES

Gets roles for which a workflow process is valid.

Syntax

```
<WORKFLOWPROCESS.GETROLES
  NAME="workflow process object name"
  OBJVARNAME="variable name" />
```

Parameters

NAME (required)

Name of workflow process object.

OBJVARNAME (required)

Name of role list object variable.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

WORKFLOWPROCESS.GETSITES

Gets sites for which a workflow process is valid.

Syntax

```
<WORKFLOWPROCESS.GETSITES
  NAME="workflow process object name"
  OBJVARNAME="variable name" />
```

Parameters

NAME (required)

Name of the workflow process object.

OBJVARNAME (required)

Name of the site list object variable.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

WORKFLOWPROCESS.GETSOURCEROLES

Returns the authorized list of roles that correspond to a particular start state.

Syntax

```
<WORKFLOWPROCESS.GETSOURCEROLES  
  NAME="workflow process object name"  
  SOURCESTATE="source state ID"  
  VARNAME="variable name" />
```

Parameters

NAME (required)

Name of the workflow process object.

SOURCESTATE (required)

Start state ID.

VARNAME (required)

Name of the output variable. This variable contains a comma-separated role list.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

WORKFLOWPROCESS.GETSTEP

Gets steps from a workflow process.

Syntax

```
<WORKFLOWPROCESS.GETSTEP  
  NAME="workflow process object name"  
  VALUE="step ID"  
  OBJVARNAME="variable name" />
```

Parameters

NAME (required)

Name of the workflow process object.

VALUE (required)

Step ID.

OBJVARNAME (required)

Name of the workflow step object variable.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

WORKFLOWPROCESS.GETSTEPS

Loads all steps of a workflow process.

Syntax

```
<WORKFLOWPROCESS.GETSTEPS
  NAME="workflow process object name"
  PREFIX="variable prefix" />
```

Parameters

NAME (required)

Name of the workflow process object.

PREFIX (required)

First part of the name of all variables and/or lists used by this operation. Variables are typically named with the prefix, a delimiter, then the field name. For example, Variables.prefix:fieldname.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

WORKFLOWPROCESS.GETTYPES

Gets asset types for which a workflow process is valid.

Syntax

```
<WORKFLOWPROCESS.GETTYPES  
  NAME="workflow process object name"  
  OBJVARNAME="variable name" />
```

Parameters

NAME (required)

Name of the workflow process object.

OBJVARNAME (required)

Name of type list object variable.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

WORKFLOWPROCESS.SCATTER

Creates independent variables and lists from an in-memory workflow process instance.

Syntax

```
<WORKFLOWPROCESS.SCATTER  
  NAME="workflow process object name"  
  PREFIX="variable prefix" />
```

Parameters

NAME (required)

Name of the workflow process instance object.

PREFIX (required)

First part of the name of all variables and/or lists used by this scatter operation. Variables are typically named with the prefix together with the field name. A delimiter is not automatically inserted after the prefix. Therefore, if you want a delimiter, such as a colon, you must specify it with the prefix. For example, Variables.prefix:fieldname.

Description

The WORKFLOWPROCESS.SCATTER tag sets the following variables:

- prefix+id

- prefix+name
- prefix+description

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

WORKFLOWPROCESS.SETADMINROLES

Sets roles that represent administrators for a workflow process.

Syntax

```
<WORKFLOWPROCESS.SETADMINROLES
  NAME="workflow process object name"
  OBJECT="object name" />
```

Parameters

NAME (required)
Name of the workflow process object.

OBJECT (required)
Name of the role list object.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

WORKFLOWPROCESS.SETDELEGATIONACTIONS

Sets delegation actions for a workflow process.

Syntax

```
<WORKFLOWPROCESS.SETDELEGATIONACTIONS  
  NAME="workflow process object name"  
  OBJECT="object name" />
```

Parameters

NAME (required)

Name of workflow process object.

OBJECT (required)

Name of workflow action list object.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

WORKFLOWPROCESS.SETFUNCTIONPRIVS

Sets the function privileges for a workflow process.

Syntax

```
<WORKFLOWPROCESS.SETFUNCTIONPRIVS  
  NAME="workflow process object name"  
  VALUE="function privilege" />
```

Parameters

NAME (required)

Name of workflow process object .

VALUE (required)

Name of the workflow privileges object.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

WORKFLOWPROCESS.SETPROCESSDESCRIPTION

Sets the description of a workflow process.

Syntax

```
<WORKFLOWPROCESS.SETPROCESSDESCRIPTION
  NAME="workflow process object name"
  VALUE="description" />
```

Parameters

NAME (required)
Name of the workflow process object.

VALUE (required)
Description text.

WORKFLOWPROCESS.SETPROCESSNAME

Sets the name of a workflow process.

Syntax

```
<WORKFLOWPROCESS.SETPROCESSNAME
  NAME="workflow process object name"
  VALUE="process name" />
```

Parameters

NAME (required)
Name of the workflow process object.

VALUE (required)
Name of the workflow process.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

WORKFLOWPROCESS.SETROLES

Sets the roles for which a workflow process is valid.

Syntax

```
<WORKFLOWPROCESS.SETROLES  
  NAME="workflow process object name"  
  OBJECT="object name" />
```

Parameters

NAME (required)

Name of the workflow process object.

OBJECT (required)

Name of the role list object.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

WORKFLOWPROCESS.SETSITES

Sets sites for which a workflow process is valid.

Syntax

```
<WORKFLOWPROCESS.SETSITES  
  NAME="workflow process object name"  
  OBJECT="object name" />
```

Parameters

NAME (required)

Name of workflow process object.

OBJECT (required)

Name of the site list object to set.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

WORKFLOWPROCESS.SETSTEPS

Sets all steps of a workflow process.

Syntax

```
<WORKFLOWPROCESS.SETSTEPS
  OBJECT="workflow process object name"
  PREFIX="prefix" />
```

Parameters

OBJECT (required)

Name of the workflow process object.

PREFIX (required)

First part of the name of all variables and/or lists used by this operation. Variables are typically named with the prefix, a delimiter, then the field name. For example, Variables.prefix:fieldname.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.
-12037	Required field missing.

WORKFLOWPROCESS.SETTYPES

Sets asset types for which a workflow process is valid.

Syntax

```
<WORKFLOWPROCESS.SETTYPES
  NAME="workflow process object name"
  OBJECT="object name" />
```

Parameters

NAME (required)

Name of the workflow process object.

OBJECT (required)

Name of asset type list object.

errno

The possible values of errno include:

Value	Description
-12020	No object by name.

Value	Description
-12037	Required field missing.

Workflow State Tags

These tags work with the workflow state. A state represents the condition or status of an asset in workflow. States are shared among more than one workflow processes, and are defined independently of the workflow processes. The following section describes each workflow state tag, beginning with [WORKFLOWSTATE.GATHER](#).

WORKFLOWSTATE.GATHER

Gathers values for fields in an object from the name/value pairs of variables defined in the environment.

Syntax

```
<WORKFLOWSTATE.GATHER
  NAME="workflow state object name"
  PREFIX="variable prefix" />
```

Parameters

NAME (required)

Name of the workflow state object whose fields you want to gather.

PREFIX (required)

First part of the name of all variables and/or lists used by this gather operation. Variables are typically named with the prefix together with the field name. If a delimiter, such as a colon, was used in the creation of the variables that you want to gather, you must use the same delimiter in the gather operation. For example, `Variables.prefix:fieldname`.

Description

The `WORKFLOWSTATE.GATHER` tag looks for the following variables:

- `prefix+id`
- `prefix+name`
- `prefix+description`
- `prefix+esttime`
- `prefix+actions+actionTotal`
- `prefix+actions+action+i` (where *i* is an integer)

WORKFLOWSTATE.GETACTIONS

Gets the actions in a workflow state.

Syntax

```
<WORKFLOWSTATE.GETACTIONS  
  NAME="workflow state object name"  
  OBJVARNAME="variable name" />
```

Parameters

NAME (required)

Name of the workflow state object.

OBJVARNAME (required)

Name of the output variable to create.

WORKFLOWSTATE.GETESTIMATEDSTATETIME

Gets the estimated time (the time between assignment and the deadline) of the state.

Syntax

```
<WORKFLOWSTATE.GETESTIMATEDSTATETIME  
  NAME="workflow state object name"  
  VARNAME="variable name" />
```

Parameters

NAME (required)

Name of the workflow state object.

VARNAME (required)

Name of the output variable to create.

WORKFLOWSTATE.GETID

Gets the ID of the workflow state.

Syntax

```
<WORKFLOWSTATE.GETID  
  NAME="workflow state object name"  
  VARNAME="variable name" />
```

Parameters

NAME (required)

Name of the workflow state object.

VARNAME (required)

Name of the output variable to create.

WORKFLOWSTATE.GETSTATEDescription

Gets the description of the workflow state.

Syntax

```
<WORKFLOWSTATE.GETSTATEDescription  
  NAME="workflow state object name"  
  VARNAME="variable name" />
```

Parameters

NAME (required)

Name of the workflow state object.

VARNAME (required)

Name of the output variable to create.

WORKFLOWSTATE.GETSTATENAME

Gets the name of the workflow state.

Syntax

```
<WORKFLOWSTATE.GETSTATENAME  
  NAME="workflow state object name"  
  VARNAME="variable name" />
```

Parameters

NAME (required)

Name of the workflow state object.

VARNAME (required)

Name of the output variable to create.

WORKFLOWSTATE.SCATTER

Creates independent variables and lists from an in-memory workflow state instance.

Syntax

```
<WORKFLOWSTATE.SCATTER  
  NAME="workflow state object name"  
  PREFIX="variable prefix" />
```

Parameters

NAME (required)

Name of the workflow process instance object whose fields are to be scattered.

PREFIX (required)

First part of the name of all variables and/or lists used by this scatter operation. Variables are typically named with the prefix together with the field name. A

delimiter is not automatically inserted after the prefix. Therefore, if you want a delimiter, such as a colon, you must specify it with the prefix. For example, `Variables.prefix:fieldname`.

Description

The `WORKFLOWSTATE.SCATTER` tag sets the following variables:

- `prefix+id`
- `prefix+name`
- `prefix+description`
- `prefix+esttime`
- `prefix+actions+actionTotal`
- `prefix+actions+action+i` (where *i* is an integer)

WORKFLOWSTATE.SETACTIONS

Sets actions in the workflow state.

Syntax

```
<WORKFLOWSTATE.SETACTIONS  
    NAME="workflow state object name"  
    OBJECT="workflow timed actions list object name" />
```

Parameters

NAME (required)

Name of the workflow state object.

OBJECT (required)

Name of the workflow timed actions list object.

WORKFLOWSTATE.SETESTIMATEDSTATETIME

Sets the estimated time (the time between assignment and the deadline) of the workflow state.

Syntax

```
<WORKFLOWSTATE.SETESTIMATEDSTATETIME  
    NAME="workflow state object name"  
    TIME="estimated time" />
```

Parameters

NAME (required)

Name of the workflow state object.

TIME (required)

Time in milliseconds between the assignment and the deadline.

WORKFLOWSTATE.SETSTATEDESCRIPTION

Sets the description of a workflow state.

Syntax

```
<WORKFLOWSTATE.SETSTATEDESCRIPTION  
  NAME="workflow state object name"  
  VALUE="text of description" />
```

Parameters

NAME (required)
Name of the workflow state object.

VALUE (required)
Text of the description.

WORKFLOWSTATE.SETSTATENAME

Sets the name of a workflow state.

Syntax

```
<WORKFLOWSTATE.SETSTATENAME  
  NAME="workflow state object name"  
  VALUE="new workflow state name" />
```

Parameters

NAME (required)
Name of the workflows state object.

VALUE (required)
New workflow state name.

Workflow Step Tags

Workflow step tags manage workflow step objects. Workflow step objects are created using the [WORKFLOWENGINE.CREATENEWSTEPACTION](#) tag. A step represents a transition between states. When an assignment is completed, the next step transition is selected and then is used to change the state. Workflow steps are defined for each workflow process. The following section describes each workflow step tag, beginning with [WORKFLOWSTEP.GATHER](#).

WORKFLOWSTEP.GATHER

Gathers variables that have been named with the specified prefix naming convention into a workflow step object.

Syntax

```
<WORKFLOWSTEP.GATHER
  NAME="stepobject"
  PREFIX="variable_prefix"/>
```

Parameters

NAME (required)

The name of the step instance object whose fields you want to gather.

PREFIX (required)

First part of the name of all variables and/or lists used by this gather operation. Variables are typically named with the prefix together with the field name. If a delimiter, such as a colon, was used in the creation of the variables that you want to gather, you must use the same delimiter in the gather operation. For example, *Variables.prefix:fieldname*.

Description

The `WORKFLOWSTEP.GATHER` tag gathers the following variables into a workflow step object:

- `prefix+id`
- `prefix+name`
- `prefix+startstate`
- `prefix+endstate`
- `prefix+override`
- `prefix+type`
- `prefix+conditions+actionTotal`
- `prefix+conditions+action+i` where *i* is an integer
- `prefix+actions+ actionTotal`
- `prefix+actions+ action+i` where *i* is an integer
- `prefix+deadlockactions+actionTotal`
- `prefix+deadlockactions+action+i` where *i* is an integer

- prefix+authroles
- prefix+notifyroles
- prefix+grouplock

WORKFLOWSTEP.GETAUTHORIZATIONROLES

Gets the authorization ACLs of the workflow step.

Syntax

```
<WORKFLOWSTEP.GETAUTHORIZATIONROLES
  NAME="step_object"
  OBJVARNAME="rolelist_object_name"/>
```

Parameters

NAME (required)

The name of the step instance object whose authorization roles are to be loaded.

OBJVARNAME (required)

Output object of the type ROLELIST.

WORKFLOWSTEP.GETCOMPLETIONACTIONS

Gets the step actions for the step.

Syntax

```
<WORKFLOWSTEP.GETCOMPLETIONACTIONS
  NAME="step_object"
  OBJVARNAME="workflowactionlist_object_name"/>
```

Parameters

NAME (required)

The name of the step instance object whose step actions are to be loaded.

OBJVARNAME (required)

Output object of type WORKFLOWACTIONLIST.

WORKFLOWSTEP.GETCONDITIONS

Gets the conditions for the step.

Syntax

```
<WORKFLOWSTEP.GETCONDITIONS
  NAME="step_object"
  OBJVARNAME="workflowactionlist_object_name"/>
```

Parameters

NAME (required)

The name of the step instance object whose step conditions are to be loaded.

OBJVARNAME (required)

Output object of type WORKFLOWACTIONLIST.

WORKFLOWSTEP.GETDEADLOCKACTIONS

Gets the deadlock actions for the step.

Syntax

```
<WORKFLOWSTEP.GETDEADLOCKACTIONS  
  NAME="step_object"  
  OBJVARNAME="workflowactionlist_object_name"/>
```

Parameters

NAME (required)

The name of the step instance object whose step deadlock actions are to be loaded.

OBJVARNAME (required)

Output object of type WORKFLOWACTIONLIST.

WORKFLOWSTEP.GETGROUPLOCKFLAG

Gets the group lock flag for the step.

Syntax

```
<WORKFLOWSTEP.GETGROUPLOCKFLAG  
  NAME="step_object"  
  VARNAME="variable_name"/>
```

Parameters

NAME (required)

The name of the step instance object whose group lock flag is to be loaded.

VARNAME (required)

Variable in which the output result is stored as true or false.

WORKFLOWSTEP.GETID

Gets the ID of the workflow step.

Syntax

```
<WORKFLOWSTEP.GETID  
  NAME="step_object"  
  VARNAME="variable_name" />
```

Parameters

NAME (required)

The name of the step instance object whose ID is to be loaded

VARNAME (required)

Variable in which the output result (step ID) is stored.

WORKFLOWSTEP.GETNAME

Gets the name of the workflow step.

Syntax

```
<WORKFLOWSTEP.GETNAME
  NAME="step_object"
  VARNAME="variable_name" />
```

Parameters

NAME (required)

The name of the step instance object whose name is to be loaded

VARNAME (required)

Variable in which output result (step name) is stored.

WORKFLOWSTEP.GETNOTIFYROLES

Gets the ACLs that are notified when a workflow step is taken.

Syntax

```
<WORKFLOWSTEP.GETNOTIFYROLES
  NAME="step_object"
  OBJVARNAME="rolelist_object_name"/>
```

Parameters

NAME (required)

The name of the workflow step object whose ACLs are to be loaded

OBJVARNAME (required)

Output object of type ROLELIST.

WORKFLOWSTEP.GETENDSTATE

Gets the end state for the step.

Syntax

```
<WORKFLOWSTEP.GETENDSTATE
  NAME="step_object"
  VARNAME="endstate_variable_name"/>
```

Parameters

NAME (required)

The name of the workflow step object whose end state is to be loaded

VARNAME (required)

Variable in which the output result (end state name) is stored

WORKFLOWSTEP.GETOVERRIDEFLAG

Gets the override flag for the step.

Syntax

```
<WorkflowStep.GetOverrideFlag  
  NAME="step_object"  
  VARNAME="flag_variable_name" />
```

Parameters

NAME (required)

The name of the step instance object whose override flag is to be loaded

VARNAME (required)

Variable in which the output result (override flag) is stored as true or false.

WORKFLOWSTEP.GETSTARTSTATE

Gets the start state for the step.

Syntax

```
<WORKFLOWSTEP.GETSTARTSTATE  
  NAME="step_object"  
  VARNAME="startstate_variable_name"/>
```

Parameters

NAME (required)

The name of the step instance object whose start state is to be loaded.

VARNAME (required)

Variable in which the output result (start state name) is stored.

WORKFLOWSTEP.GETSTEPTYPE

Gets the type of the workflow step.

Syntax

```
<WORKFLOWSTEP.GETSTEPTYPE
  NAME="step_object"
  VARNAME="steptype_variable_name" />
```

Parameters

NAME (required)

The name of the step instance object whose step type is to be loaded.

VARNAME (required)

Variable in which the output result (step type) is stored.

WORKFLOWSTEP.SCATTER

Creates independent variables and lists (or “scatters”) from an in-memory workflow step instance.

Syntax

```
<WORKFLOWSTEP.SCATTER
  NAME="workflow_step_object"
  PREFIX="variable_prefix" />
```

Parameters

NAME (required)

The name of the workflow step object whose fields are to be scattered.

PREFIX (required)

First part of the name of all variables and/or lists used by this scatter operation.

Variables are typically named with the prefix together with the field name. A delimiter is not automatically inserted after the prefix. Therefore, if you want a delimiter, such as a colon, you must specify it with the prefix. For example,

Variables.*prefix:fieldname*.

Description

The WORKFLOWSTEP.SCATTER tag sets the following variables:

- prefix+id
- prefix+name
- prefix+startstate
- prefix+endstate
- prefix+override
- prefix+type
- prefix+conditions+actionTotal

- prefix+conditions+action+i where *i* is an integer
- prefix+actions+ actionTotal
- prefix+actions+ action+i where *i* is an integer
- prefix+deadlockactions+actionTotal
- prefix+deadlockactions+action+i where *i* is an integer
- prefix+authroles
- prefix+notifyroles
- prefix+grouplock

WORKFLOWSTEP.SETAUTHORIZATIONROLES

Sets the authorization ACLs for the step.

Syntax

```
<WORKFLOWSTEP.SETAUTHORIZATIONROLES  
  NAME="step_object"  
  OBJECT="ROLELIST_object"/>
```

Parameters

NAME (required)

The name of the workflow step object whose authorization roles are to be set.

OBJECT (required)

Object of type ROLELIST containg authorization roles.

WORKFLOWSTEP.SETCOMPLETIONACTIONS

Sets the step actions for the step.

Syntax

```
<WORKFLOWSTEP.SETCOMPLETIONACTIONS  
  NAME="step_object"  
  OBJECT="workflowactionlist_object"/>
```

Parameters

NAME (required)

The name of the step instance object whose step actions are to be set

OBJECT (required)

Object of type WORKFLOWACTIONLIST containg authorization roles.

WORKFLOWSTEP.SETCONDITIONS

Sets the conditions for the step.

Syntax

```
<WORKFLOWSTEP.SETCONDITIONS  
  NAME="step_object"  
  OBJECT="conditions_list" />
```

Parameters

NAME (required)

Name of the step object

OBJECT (required)

Object of type WORKFLOWACTIONLIST that contains the conditions.

WORKFLOWSTEP.SETDEADLOCKACTIONS

Sets the deadlock actions for the step.

Syntax

```
<WORKFLOWSTEP.SETDEADLOCKACTIONS  
  NAME="step_object"  
  OBJECT="workflowactionlist_object" />
```

Parameters

NAME (required)

Name of the step object

OBJECT (required)

Object of type WORKFLOWACTIONLIST that contains the deadlock actions

WORKFLOWSTEP.SETENDSTATE

Sets the end state for the step.

Syntax

```
<WORKFLOWSTEP.SETENDSTATE  
  NAME="step_object"  
  VALUE="endstate_id"/>
```

Parameters

NAME (required)

The name of the workflow step object whose end state actions are to be set.

VALUE (required)

The ID of the end state

WORKFLOWSTEP.SETGROUPLOCKFLAG

Sets the group lock flag for the step.

Syntax

```
<WORKFLOWSTEP.SETGROUPLOCKFLAG  
  NAME="step_object"  
  FLAG="boolean_variable_name"/>
```

Parameters

NAME (required)

The name of the workflow step object whose grouplock flag is to be set.

FLAG (required)

The name of the variable whose value can be true or false.

WORKFLOWSTEP.SETNAME

Sets the name of a step in a step object.

Syntax

```
<WORKFLOWSTEP.SETNAME  
  NAME="step_object"  
  VALUE="new_step_name"/>
```

Parameters

NAME (required)

VALUE (required)

WORKFLOWSTEP.SETNOTIFYROLES

Sets the ACLs that are notified when a workflow step is taken.

Syntax

```
<WORKFLOWSTEP.SETNOTIFYROLES  
  NAME="step_object"  
  OBJECT="rolelist_object"/>
```

Parameters

NAME (required)

The name of the workflow step object whose ACLs are to be set.

OBJECT (required)

Object of type ROLELIST containing the ACLs.

WORKFLOWSTEP.SETOVERRIDEFLAG

Sets the override flag for the step.

Syntax

```
<WORKFLOWSTEP.SETOVERRIDEFLAG  
    NAME="stepobject"  
    FLAG="Variables.clearall"/>
```

Parameters

NAME (required)

The name of the step instance object whose override flag is to be set.

FLAG (required)

Variable containing the name of the override flag (true or false).

WORKFLOWSTEP.SETSTARTSTATE

Sets the start state for the step.

Syntax

```
<WORKFLOWSTEP.SETSTARTSTATE  
    NAME="step_object"  
    VALUE="startstate_id"/>
```

Parameters

NAME (required)

The name of the workflow step object whose start state is to be set,

VALUE (required)

The ID of start state,

WORKFLOWSTEP.SETSTEPTYPE

Sets the type for the step.

Syntax

```
<WORKFLOWSTEP.SETSTEPTYPE  
    NAME="step_object"  
    TYPE="steptype_variable_name" />
```

Parameters

NAME (required)

The name of the step instance object whose grouplock flag is to be set.

TYPE (required)

Variable containing the name of the step type.

Workflow Timed Action Tags

This group of tags manages the workflow timed actions. The following section describes each workflow timed action tag, beginning with [WORKFLOWTIMEDACTION.GATHER](#).

WORKFLOWTIMEDACTION.GATHER

Gathers values for fields in an object from the name/value pairs of variables defined in the environment.

Syntax

```
<WORKFLOWTIMEDACTION.GATHER
  NAME="workflow timed action object"
  PREFIX="prefix" />
```

Parameters

NAME (required)

Name of the workflow timed action instance object.

PREFIX (required)

First part of the name of all variables and/or lists used by this gather operation. Variables are typically named with the prefix together with the field name. If a delimiter, such as a colon, was used in the creation of the variables that you want to gather, you must use the same delimiter in the gather operation. For example, Variables.prefix:fieldname.

Description

The WORKFLOWTIMEDACTION.GATHER tag looks for the following variables:

- prefix+action
- prefix+time

WORKFLOWTIMEDACTION.GETASSIGNMENTACTION

Gets the assignment for a workflow timed action.

Syntax

```
<WORKFLOWTIMEDACTION.GETASSIGNMENTACTION
  NAME="workflow timed action object name"
  VARNAME="variable name" />
```

Parameters

NAME (required)

Name of the workflow timed action object.

VARNAME (required)

Name of the output variable to create. The variable contains the ID of the workflow timed action.

WORKFLOWTIMEDACTION.GETRELATIVETIME

Gets the relative time (relative to the assignment due time) for the workflow timed action.

Syntax

```
<WORKFLOWTIMEDACTION.GETRELATIVETIME
  NAME="workflow timed action object name"
  VARNAME="variable name" />
```

Parameters

NAME (required)

Name of the workflow timed action object.

VARNAME (required)

Name of the output variable to create.

WORKFLOWTIMEDACTION.SCATTER

Creates independent variables and lists from an in-memory workflow timed action instance.

Syntax

```
<WORKFLOWTIMEDACTION.SCATTER
  NAME="workflow timed action object name"
  PREFIX="prefix" />
```

Parameters

NAME (required)

Name of the workflow timed action instance object.

PREFIX (required)

First part of the name of all variables and/or lists used by this scatter operation. Variables are typically named with the prefix together with the field name. A delimiter is not automatically inserted after the prefix. Therefore, if you want a delimiter, such as a colon, you must specify it with the prefix. For example, Variables.prefix:fieldname.

Description

The WORKFLOWTIMEDACTION.SCATTER tag sets the following variables:

- prefix+action
- prefix+time

WORKFLOWTIMEDACTION.SETASSIGNMENTACTION

Sets the assignment for the workflow timed action.

Syntax

```
<WORKFLOWTIMEDACTION.SETASSIGNMENTACTION  
  NAME="workflow timed action object name"  
  VALUE="workflow timed action ID" />
```

Parameters

NAME (required)
Name of the workflow timed action object.

VALUE (required)
Workflow timed action ID.

WORKFLOWTIMEDACTION.SETRELATIVETIME

Sets the relative time for the workflow timed action.

Syntax

```
<WORKFLOWTIMEDACTION.SETRELATIVETIME  
  NAME="workflow timed action object name"  
  TIME="relative time" />
```

Parameters

NAME (required)
Name of the workflow timed action object.

TIME (required)
Relative time in milliseconds with + or -.

Workflow Timed Action List Tags

This group of tags manage the workflow timed action list object. The following section describes each workflow timed action list tag, beginning with [WORKFLOWTIMEDACTIONLIST.ADDTIMEDACTION](#).

WORKFLOWTIMEDACTIONLIST.ADDTIMEDACTION

Adds a timed action to timed action list.

Syntax

```
<WORKFLOWTIMEDACTIONLIST.ADDTIMEDACTION
  NAME="workflow timed action list object name"
  OBJECT="workflow timed action object name" />
```

Parameters

NAME (required)
Name of the timed action list object.

OBJECT (required)
Name of the timed action object.

WORKFLOWTIMEDACTIONLIST.CLEARTIMEDACTIONS

Removes all timed actions from the timed action list object.

Syntax

```
<WORKFLOWTIMEDACTIONLIST.CLEARTIMEDACTIONS
  NAME="workflow timed action list object name" />
```

Parameters

NAME (required)
Name of the workflow timed action list object.

WORKFLOWTIMEDACTIONLIST.CREATE

Creates a workflow timed action object.

Syntax

```
<WORKFLOWTIMEDACTIONLIST.CREATE
  NAME="workflow timed action list object name"
  OBJVARNAME="variable name"/>
```

Parameters

NAME (required)
Name of the workflow timed action list object.

OBJVARNAME (required)
Name of the output object variable.

WORKFLOWTIMEDACTIONLIST.GATHER

Gathers values for fields in an object from the name/value pairs of variables defined in the environment.

Syntax

```
<WORKFLOWTIMEACTIONLIST.GATHER  
  NAME="workflow timed action list object name"  
  PREFIX="prefix" />
```

Parameters

NAME (required)
Name of the workflow timed action list object.

PREFIX (required)
First part of the name of all variables and/or lists used by this gather operation. Variables are typically named with the prefix together with the field name. If a delimiter, such as a colon, was used in the creation of the variables that you want to gather, you must use the same delimiter in the gather operation. For example, Variables.prefix:fieldname.

Description

The WORKFLOWTIMEDACTIONLIST.GATHER tag looks for the following variables:

- prefix+actionTotal
- prefix+action+i (where i is an integer)

WORKFLOWTIMEDACTIONLIST.GETALLTIMEDACTIONS

Gets all timed actions in a workflow timed action list.

Syntax

```
<WORKFLOWTIMEDACTIONLIST.GETALLTIMEDACTIONS  
  NAME="workflow timed action list object name"  
  PREFIX="prefix" />
```

Parameters

NAME (required)
Name of timed action list object.

PREFIX (required)
First part of the name of all variables and/or lists used by this operation. Variables are typically named with the prefix, a delimiter, then the field name. For example, Variables.prefix:fieldname

WORKFLOWTIMEDACTIONLIST.SCATTER

Creates independent variables and lists from an in-memory workflow timed action list instance.

Syntax

```
<WORKFLOWTIMEDACTIONLIST.SCATTER
  NAME="workflow timed action list object name"
  PREFIX="prefix" />
```

Parameters

NAME (required)

Name of the workflow timed action list object whose fields are to be scattered.

PREFIX (required)

First part of the name of all variables and/or lists used by this scatter operation. Variables are typically named with the prefix together with the field name. A delimiter is not automatically inserted after the prefix. Therefore, if you want a delimiter, such as a colon, you must specify it with the prefix. For example, Variables.prefix:fieldname.

Description

The WORKFLOWTIMEDACTIONLIST.SCATTER tag sets the following variables:

- prefix+actionTotal
- prefix+action+i (where i is an integer)

WORKFLOWTIMEDACTIONLIST.REMOVETIMEDACTION

Removes a workflow timed action from a workflow timed action list object.

Syntax

```
<WORKFLOWTIMEDACTIONLIST.REMOVETIMEDACTION
  NAME="workflow timed action list object name"
  OBJECT="workflow timed action object name" />
```

Parameters

NAME (required)

Name of the workflow timed action list object.

OBJECT (required)

Name of the workflow timed action object.

Chapter 3

CS-Direct Advantage XML Tags

This chapter describes the functionality and XML syntax of the CS-Direct Advantage object methods. These methods are implemented as Content Server custom application tags, which invoke application business logic for applications built with Content Server. It serves as a reference for the tags that you use to:

- Create catalog pages
- Create shopping carts
- Implement catalog searching
- Display selected product information

Use the CS-Engage object methods, described in [Chapter 4, “CS-Engage XML Tags,”](#) to gather data about visitors to your site, and to display recommendations or promotions (merchandising information) based on that data.

Note

Although this chapter contains some examples, please refer to the sample catalog (GE Lighting), where you can search the sample code for examples of nearly every CS-Direct Advantage tag used in a consistent context.

Assetset Tags

An **assetset** is a collection of assets that can be described by either a list or a search constraint. An assetset can include different types of assets. It is not limited to assets of one type.

If you are using CS-Engage, there is an additional assetset tag available, [ASSETSET.ESTABLISHRATINGS](#).

The following section describes each assetset tag, beginning with [ASSETSET.GETASSETCOUNT](#).

ASSETSET.GETASSETCOUNT

Retrieves a count of assets, given optional sort criteria.

Syntax

```
<ASSETSET.GETASSETCOUNT
  NAME="assetsetname"
  [LIST="attriblist"]
  VARNAME="varname"/>
```

Parameters

NAME (required)

Input parameter. Name of the assetset object.

LIST (optional)

Input parameter. Name of the list that determines sort order. This list has the following columns:

- **attributetypename**
- **attributename** – name of attribute, sort by or has one of the following special values:
 - **_RATING_** – order by asset rating (CS-Engage only)
 - **_ASSETTYPE_** – order by asset type
 - **direction** – can be either ascending or descending

VARNAME (required)

Input and output parameter. As input, name of the variable that holds the assigned count on output.

Description

When called, **ASSETSET.GETASSETCOUNT** creates the specified variable with the count of table rows that results if [ASSETSET.GETASSETLIST](#) were called with similar parameters.

Example

```
<!-- ASSETSET.GETASSETCOUNT: How many Movie assets are there? -->

<SEARCHSTATE.CREATE NAME="ss"/>
```

```
<ASSETSET.SETSEARCHEDASSETS NAME="as" ASSETTYPES="Movie"
CONSTRAINT="ss"/> <ASSETSET.GETASSETCOUNT NAME="as"
VARNAME="count"/>
<CSVAR NAME="Variables.count Movie(s)"/>
```

ASSETSET.GETASSETLIST

Retrieves an ordered list of assets, given optional sort criteria.

Syntax

```
<ASSETSET.GETASSETLIST
  NAME="assetsetname"
  [LIST="attriblist"]
  [MAXCOUNT="rowcount"]
  [METHOD="random|highest"]
  LISTVARNAME="varname"/>
```

Parameters

NAME (required)

Input parameter. Name of an assetset object.

LIST (optional)

Input parameter. Name of the list that determines sort order. This list has three columns:

- `attributetypename`
- `attributename` – Name of attribute to sort by or has one of the following special values:
 - `_RATING_` – Order by asset rating (CS-Engage only)
 - `_ASSETTYPE_` – Order by asset type
 - `direction` – Either ascending or descending

MAXCOUNT (optional)

Input parameter. Maximum number of rows to return in the list. A value of 0 (zero) indicates all. If count is less than the number that otherwise is returned, then items are selected according to the specified **METHOD** parameter.

METHOD (optional)

Input parameter. Required only if the value of **MAXCOUNT** is less than the number of items described. **METHOD** can have one of the following values:

- `random` – Random weighted selection based on rating
- `highest` – Best selection based on rating

This parameter is meaningful only in the context of CS-Engage.

LISTVARNAME (required)

Input and output parameter. As input, name of the list that holds the assigned values on output. The list has the following columns:

- `assettype`
- `assetid`
- A column for each sort attribute name, prefixed by the string `SORT_`.

Description

Each row in the ordered list holds a value for each attribute that is used for sorting. For example, if an assetset represents a product that has two authors, (say Fred and Sam), and the result is specified to sort by author, then the result set has a column called `SORT_author`, with two rows (one for Fred and one for Sam).

Note

If you sort on an attribute that does not exist for some assets, those assets are filtered from the results.

Example

```
<!-- Apply a sort list to an assetset and retrieve a resultset
list -->
<ASSETSET.GETASSETLIST
  NAME="myasset"
  LIST="listout"
  LISTVARNAME="aslist"/>
```

ASSETSET.GETATTRIBUTEVALUECOUNT

Gets the count of distinct attribute values for a specific attribute associated with the products represented by an assetset.

Syntax

```
<ASSETSET.GETATTRIBUTEVALUECOUNT
  NAME="assetsetname"
  ATTRIBUTE="attribname"
  [TYPENAME="assettypename"]
  VARNAME="varname"/>
```

Parameters

NAME (required)

Input parameter. Name of the assetset object.

ATTRIBUTE (required)

Input parameter. Name of an attribute whose values are to be looked up.

TYPENAME (optional)

Input parameter. Asset type name of the attribute. The default is set in the `gator.ini` file: `mwb.defaultattributes=PAtributes`.

VARNAME (required)

Input and output parameter. As input, name of the variable that holds the count of the unique values requested on output.

Example

```
<ASSETSET.GETATTRIBUTEVALUECOUNT NAME="promotion"
ATTRIBUTE="headline" VARNAME="count"/> <CSVAR NAME= "Found
Variables.count asset(s)"/>
```

ASSETSET.GETATTRIBUTEVALUES

Gets the list of attribute values that are associated with the assets represented by an assetset.

Syntax

```
<ASSETSET.GETATTRIBUTEVALUES NAME="assetsetname"
    ATTRIBUTE="attribname"
    [TYPENAME="assettypename"]
    LISTVARNAME="varname"
    [ORDERING="ascending|descending"]/>
```

Parameters

NAME (required)

Input parameter. Name of the assetset object.

ATTRIBUTE (required)

Input parameter. Name of an attribute whose value should be looked up.

TYPENAME (optional)

Input parameter. Asset type name of the attribute. The default is set in the gator.ini file: mwb.defaultattributes=PAtributes.

LISTVARNAME (required)

Input and output parameter. As input, name of a list that holds a list of string values assigned on output and obtained from the assetset's attributes.

ORDERING (optional)

Input parameter. Indicates whether the result list should be in ascending or descending order; value can be either ascending or descending.

Example

```
<!-- Iterate through a list of attribute values. -->
<!-- First, create a blank searchstate. -->
<SEARCHSTATE.CREATE NAME="ss"/>
<!-- Associate the blank searchstate with a new asset set. -->
<ASSETSET.SETSEARCHEDASSETS NAME="as" CONSTRAINT="ss"/>
<!-- Get a set of all values for the given attribute. -->
<ASSETSET.GETATTRIBUTEVALUES ORDERING="descending" NAME="as"
    ATTRIBUTE="Price" LISTVARNAME="results"/>
```

ASSETSET.GETMULTIPLEVALUES

Scatters attribute values from several attributes (and potentially more than one product) into several specified lists.

Syntax

```
<ASSETSET.GETMULTIPLEVALUES NAME="assetsetname"
    LIST="listname"
    [BYASSET="true|false"]
    PREFIX="prefix"/>
```

Parameters

NAME (required)

Input parameter. Name of the assetset object.

LIST (required)

Input parameter (see as output, below). List of attribute names to scatter. This list has three columns:

- `attributetypename`
- `attributename`
- `direction` Possible values include: none, ascending, or descending. If the `BYASSET` parameter is set to `true` the value of this column must be set to none.

BYASSET (optional)

Input parameter. Boolean value indicating whether to scatter a different instance of each attribute for each individual asset in the assetset (`true`), or to group all the attribute values for all assets together into one list per attribute (`false`). The default is `false`.

PREFIX (required)

Input parameter. Starting part of the list names that this tag creates.

LIST

Output parameter. If `BYASSET` is `false`, creates a set of lists with the names: `prefix:attrname`. If `BYASSET` is `true`, creates a set of lists with the names: `prefix:assetid:attrname`. If any list is empty (that is, has no values), does not create, change, or clear the list of that name at all.

Description

`ASSETSET.GETMULTIPLEVALUES`, as a highly efficient tag, is a good choice when the goal is to display a fixed-format table of assets, or to obtain many attributes of a single asset (such as for a product detail page).

`ASSETSET.GETMULTIPLEVALUES` has the following limitations:

- Only non-foreign attributes can be scattered.
- Text-type attributes cannot be scattered.

Example

```
<ASSETSET.GETMULTIPLEVALUES NAME="as"
  LIST="fields"
  PREFIX="at"
  BYASSET="true"/>
```

ASSETSET.SETASSET

Builds an asset set from a single asset.

Syntax

```
<ASSETSET.SETASSET NAME="assetsetname"
  TYPE="assettype"
  ID="assetid"
  [LOCALE="localeobject"]
  [DEPTYPE="exact|exists|none"] />
```

Parameters

NAME (required)

Input and output parameter. As input, name of the assetset object that holds the specified values on output.

TYPE (required)

Input parameter. Internal asset name.

ID (required)

Input parameter. Asset identifier.

LOCALE (optional)

Input parameter. Name of a locale object, which determines sort order.

DEPTYPE (optional)

Specifies whether approval dependency is *exact*, *exists* or *greater*.

Legal values are:

- **exact** – Specifies a version dependency. The version of the dependent asset must be equal to the version of the asset when it was approved.
- **exists** – Specifies that any version of the asset satisfies the dependency condition.
- **none** (default)– Specifies no approval dependency.

Example

```
<!-- Get attributes for the specified product. -->
<ASSETSET.SETASSET
  TYPE="Products"
  NAME="myasset"
  ID="Variables.additem"/>
```

ASSETSET.SETEMPTY

Creates an empty or null assetset object.

Syntax

```
<ASSETSET.SETEMPTY NAME="assetsetname"  
  [LOCALE="localeobject"]  
  [DEPTYPE="exact|exists|none"] />
```

Parameters

NAME (required)

Input and output parameter. As input, name of the assetset object to create on output.

LOCALE (optional)

Input parameter. Name of a locale object, which determines sort order.

DEPTYPE (optional)

Specifies whether approval dependency is `exact`, `exists` or `greater`.

Legal values are:

- **exact** – Specifies a version dependency. The version of the dependent asset must be equal to the version of the asset when it was approved.
- **exists** – Specifies that any version of the asset satisfies the dependency condition.
- **none** (default) – Specifies no approval dependency.

Example

```
<ASSET.SETEMPTY NAME="promotion"/>
```

ASSETSET.SETLISTEDASSETS

Builds an assetset from a list of assets.

Syntax

```
<ASSETSET.SETLISTEDASSETS NAME="assetsetname"  
  [ASSETTYPES="assettype"]  
  ASSETS="listname"  
  [LOCALE="localeobject"]  
  [DEPTYPE="exact|exists|none"] />
```

Parameters

NAME (required)

Input and output parameter. As input, name of the assetset object to create on output, representing the list of passed assets.

ASSETTYPES (optional)

Input parameter. Name of a list of asset types to include in building the set. The list has one column called `asset type`. If null, then all assets in the assets list are considered.

ASSETS (required)

Input parameter. Name of a list containing the columns `assettype` and `assetid`. This list is used to create the assetset.

LOCALE (optional)

Input parameter. Name of a locale object, which determines sort order.

DEPTYPE (optional)

Specifies whether approval dependency is `exact`, `exists` or `greater`.

Legal values are:

- **exact** – Specifies a version dependency. The version of the dependent asset must be equal to the version of the asset when it was approved.
- **exists** – Specifies that any version of the asset satisfies the dependency condition.
- **none** (default) – Specifies no approval dependency.

Description

`ASSETSET.SETLISTEDASSETS` builds an assetset where the assets are described by their type name and asset identifier.

Example

```
<!-- ASSETSET.SETLISTEDASSETS: create an assetset from a list of
assets
-->
<SETVAR NAME="errno" VALUE="0"/>
<SETVAR NAME="tablename" VALUE="mp3"/>
<EXECSQL SQL="SELECT id AS assetid, 'Variables.tablename' AS
assettype FROM Variables.tablename ORDER BY id DESC"
LIST="lLatestmp3s" LIMIT="8"/> <ASSETSET.SETLISTEDASSETS
NAME="latest" ASSETTYPES="lLatestmp3s" ASSETS="lLatestmp3s"/>
```

ASSETSET.SETSEARCHEDASSETS

Creates an assetset object which represents all assets of specific types narrowed by specified search criteria.

Syntax

```
<ASSETSET.SETSEARCHEDASSETS
  NAME="assetsetname"
  [ASSETTYPES="assettype"]
  [CONSTRAINT="searchstateobject"]
  [LOCALE="localeobject"]
  [SITE="siteidentifier"]
  [DEPTYPE="exact|exists|none"]/>
```

Parameters

NAME (required)

Input and output parameter. As input, name of the assetset object to create on output, representing the specific asset types and search criteria.

ASSETTYPES (optional)

Input parameter. A string of asset types; for example, "Products, AArticles, AImages", to include in building the set. Identifying specific assets is recommended, since the search becomes much slower for each asset type searched. If null, then all assets in the system are considered.

CONSTRAINT (optional)

Input parameter. Name of a searchstate object describing the search constraint.

LOCALE (optional)

Input parameter. Name of a locale object, which determines sort order.

SITE (optional)

Input parameter. Site identifier to search.

DEPTYPE (optional)

Specifies whether approval dependency is exact, exists or greater.

Legal values are:

- **exact** – Specifies a version dependency. The version of the dependent asset must be equal to the version of the asset when it was approved.
- **exists** – Specifies that any version of the asset satisfies the dependency condition.
- **none** (default)– Specifies no approval dependency.

Example

```
<!-- Iterate through a list of attribute values. -->
<!-- First, create a blank searchstate. -->
<SEARCHSTATE.CREATE NAME="ss"/>
<!-- Associate the blank searchstate with a new asset set. -->
<ASSETSET.SETSEARCHEDASSETS NAME="as" CONSTRAINT="ss"
ASSETTYPES="Products"/>
<!-- Get a set of all values for the given attribute. -->
<ASSETSET.GETATTRIBUTEVALUES ORDERING="descending" NAME="as"
ATTRIBUTE="Price" LISTVARNAME="results"/>
```

Blob Service Tags

The following section describes each blob service tag, beginning with [BLOBSERVICE.GETIDCOLUMN](#).

BLOBSERVICE.GETIDCOLUMN

Returns the name of the key column to be passed to other blob management tags.

Syntax

```
<BLOBSERVICE.GETIDCOLUMN  
  VARNAME="variable name"/>
```

Parameters

VARNAME (required)
Name of the variable that contains the column.

Example

The output from the following code sample is: id.

```
<BLOBSERVICE.GETIDCOLUMN VARNAME="blobIdColumn"/>
```

BLOBSERVICE.GETTABLENAME

Returns the name of the blob table to be passed to the blob management tags.

Syntax

```
<BLOBSERVICE.GETTABLENAME  
  VARNAME="variable name"/>
```

Parameters

VARNAME (required)
Name of the variable that contains the table.

Example

The output from the following code sample is: MungoBlobs.

```
<BLOBSERVICE.GETTABLENAME VARNAME="blobTableName"/>
```

BLOBSERVICE.GETURLCOLUMN

Returns the name of a URL column to be passed to other blob management tags.

Syntax

```
<BLOBSERVICE.GETURLCOLUMN VARNAME="variable name"/>
```

Parameters

VARNAME (required)

Name of the variable that contains the name of the column.

Example

The output from the following code sample is: urldata.

```
<BLOBSERVICE.GETURLCOLUMN VARNAME="blobURLName"/>
```

BLOBSERVICE.READDATA

Uses an ID to load the file name and URL data of the specific blob.

Syntax

```
<BLOBSERVICE.READDATA ID="blob id"
    LISTVARNAME="output list"/>
```

Parameters

ID (required)

Identifies the blob to be read.

LISTVARNAME (required)

Name of the list containing the data. The list is created with one column called urldata.

Calculator Tags

The calculator tag provides basic calculator and Boolean functionality. The following section describes the calculator tag, [CALCULATOR.GO](#).

CALCULATOR.GO

Calculates the value of an expression.

Syntax

```
<CALCULATOR.GO
    VALUE="postfixstring"
    VARNAME="result" />
```

Parameters

VALUE (required)

Input parameter. Postfix expression as a string with the following format:

“opreand1 operand2 ... operandn opchar”

where operand1, operand2, ..., operandn are one or more operands, and opchar is one of the following operator characters or strings:

+	-	*	/	%	=	!=
LT	GT	LTE	GTE	AND	OR	NOT

All values are separated by spaces, and the entire VALUE input string is contained within double quotes.

VARNAME (required)

Input and output parameter. As input, name of a string that holds the output that results from evaluating the expression.

Description

The value is computed using reverse polish notation. Numbers in the expression string are pushed onto a stack. Operations pop numbers off the stack as needed and push the result. The final answer is the top element of the stack.

Example

```
<!-- Divide total price by item quantity to get the item price. -->
<CALCULATOR.GO VALUE="cartlist.price cartlist.quantity /"
    VARNAME="calcout" />
<!-- Convert the above item price to currency. -->
<CURRENCY.CREATE NAME="curval" />
<CURRENCY.GETCURRENCY NAME="curval" VALUE="Variables.calcout"
    VARNAME="curout" />
```

Cart Tags

A shopping cart object contains the following information:

- A list of product IDs, the quantity, the purchase amount, and the storeID for each product.

Note

The storeID number is assigned by your Transact administrator when you register an online store with Transact.

- A list of discount descriptions and amounts, which result from applying a campaign discount to the cart.
- A set of cart and item parameters required by the Commerce Connector.

See [Chapter 11, “Shopping Cart Parameters,”](#) for cart parameter names and values. See [Chapter 12, “Order Item Parameters,”](#) for individual item parameter names and values. Cart tags include:

CART.ADDITEM	CART.GETITEMS
CART.CHECK	CART.GETITEMTOTAL
CART.CLEAR	CART.GETLEGALVALUES
CART.CLEARDISCOUNTS	CART.GETPARAMETER
CART.COMMIT	CART.GETPARAMETERS
CART.CREATE	CART.GETPRELIMINARYTOTAL
CART.DELETEITEM	CART.GETSHIPPINGTOTAL
CART.GETCARTDISCOUNTS	CART.GETSTORES
CART.GETCARTDISCOUNTTOTAL	CART.GETTAXTOTAL
CART.GETERRORS	CART.GETTRANSACTIONID
CART.GETFINALTOTAL	CART.GETTRANSACTIONIDS
CART.FROMSTRING	CART.MERGE
CART.GETITEMDISCOUNTS	CART.SETITEMPARAMETERS
CART.GETITEMDISCOUNTTOTAL	CART.SETITEMQUANTITY
CART.GETITEMERRORS	CART.SETPARAMETER
CART.GETITEMLEGALVALUES	CART.SETPARAMETERS
CART.GETITEMPARAMETER	CART.TOSTRING
CART.GETITEMPARAMETERS	

CART.ADDITEM

Adds an order item to the shopping cart.

Syntax

```
<CART.ADDITEM
  NAME="cartname"
  STOREID="storeid"
  [TYPE="products"]
  ID="assetid"
  QUANTITY="unitspurchased"
  PRICE="totalprice"
  [LIST="orderitems"
  MERGE="true|false"
  INSERTBEFORE="rowid"
  VARNAME="rowid"]/>
```

Parameters

NAME (required)

Input parameter. Name of the cart object.

STOREID (required)

Input parameter. Positive integer, not more than eight digits, that identifies the store to Content Sever Direct Advantage. This number is assigned by your Transact administrator.

TYPE (optional)

Input parameter. Type of item to add. The default is Products.

ID (required)

Input and output parameter. As input, asset id of the order item that is added to the cart on output.

QUANTITY (required)

Input parameter. Number of units of this item that the buyer is purchasing. If items are sold in packages, then Quantity should represent the number of packages, not the total number of items in those packages.

PRICE (required)

Input parameter. Total price of this product, specified as an integer or a number with two decimal places. PRICE cannot be negative, PRICE cannot hold a currency character such as a dollar sign (\$), and PRICE cannot hold a comma (,) as used in many European countries. For example, 42.99 is an acceptable value of PRICE, but \$42.99 and 42,999 are not.

For both tangible goods and digital goods, set PRICE to be the total price of the item. If QUANTITY is greater than 1, then PRICE is the total price, not the price of an individual unit. For example, if you sell a gift set of three shirts for \$60, you set QUANTITY to 3 and PRICE to \$60. You do not set Price to \$20, the price of each individual shirt.

For digital subscriptions, set PRICE to be the price of each installment payment. For example, if a subscription costs \$100, payable in four equal installments, you set PRICE to 25.

LIST (optional)

Input parameter. List of order item parameters for the item.

MERGE (optional)

Input parameter. Boolean: whether to attempt to merge quantity, price with (possibly) existing item with the input ID (asset id).

INSERTBEFORE (optional)

Input parameter. Row identifier of the item before which to insert this item if no merge occurs (the default is to insert at end).

VARNAME (optional)

Input parameter. Row identifier for the item. This is always set by `addItem` (not by the caller) and uniquely identifies the line item in the cart.

Description

See [Chapter 12, “Order Item Parameters,”](#) for individual item parameter names and values.

CART.CHECK

Checks the cart against Transact via the Commerce Connector.

Syntax

```
<CART.CHECK NAME="cartname"
  VARNAME="varname"/>
```

Parameters**NAME (required)**

Input parameter. Name of the cart object.

VARNAME (required)

Input parameter. Name of the variable that holds one of the following values:

- `true` – the cart is acceptable and can be committed to Transact.
- `false` – the cart is not acceptable to Transact.

Description

As a result of the check, `CART.CHECK` updates the following parameters with legal values:

- `subTotal` – The total cost of items
- `shippingInfo-shippingCost` – The total cost of shipping.
`shippingInfo-shipperID` – The Transact shipper GUID.
`shippingInfo-shipperName` – The Transact shipper name.
- `tax-taxType` – Type (`USSalesTax`, `Canadian`, `VAT`)
`tax-taxAmount` – The total tax amount (if `taxType` is not `Canadian`)
`tax-GST` – Canadian GST
`tax-PST` – Canadian PST
- `shippingInfo-shippingTax-taxType` – Type (`USSalesTax`, `Canadian`, `VAT`)
`shippingInfo-shippingTax-taxAmount` – if `taxType` is not `Canadian`
`shippingInfo-shippingTax-GST` – Canadian GST
`shippingInfo-shippingTax-PST` – Canadian PST
- `totalSalesAmount` – The sum of `subTotal`, `shippingCost`, and `tax-taxAmount`.

- store – A set of parameters that come from the `transact.ini` file. (Normally, the Web developer is only concerned with `storeID` and `currencyCode`.)

See [Chapter 11, “Shopping Cart Parameters,”](#) for details on returned parameters.

The `CART.CHECK` tag also returns an error code and error message if Transact cannot resolve any other field in the cart (for example, a missing postal code, or a postal code that is not legal for a specified city).

- As a result of the check, the cart object is updated to reflect any errors or notifications retrieved from the Transact. Choices for fields that have choices also become available.

Example

```
<!-- Get the current cart. -->
<COMMERCECONTEXT.GETCURRENTCART VARNAME="thiscart"/>
<!-- Check the cart to enumerate payment methods. -->
<CART.CHECK NAME="thiscart" VARNAME="cartstatus"/>
<!-- Now get the legal values for the payment types. -->
<CART.GETLEGALVALUES NAME="thiscart" FIELD="paymentInfo-brandID"
LISTVARNAME="paylist"/>
```

CART.CLEAR

Empties the cart.

Syntax

```
<CART.CLEAR NAME="cartname"/>
```

Parameters

NAME (required)

Input parameter. Name of the cart object.

Description

Use `CART.CLEAR` after a successful commit, or whenever the contents of the cart need to be cleared. When called, `CART.CLEAR` clears the cart, including all contents, items, and discounts.

CART.CLEARDISCOUNTS

Clears the cart's current discounts.

Syntax

```
<CART.CLEARDISCOUNTS NAME="cartname"/>
```

Parameters

NAME (required)

Input parameter. Name of the cart object.

Description

Use this tag whenever the discounts in a cart need to be cleared.

CART.COMMIT

Sends a checked cart to Transact via the Commerce Connector.

Syntax

```
<CART.COMMIT NAME="cartname" VARNAME="varname"/>
```

Parameters

NAME (required)

Input parameter. Name of the cart object.

VARNAME (required)

Input and output parameter. As input, name of the variable that holds a boolean value output as a result of the attempted commit. True means that the cart has been committed to Transact.

Description

Only a successfully checked cart can be committed. A commit of an otherwise acceptable cart may fail due to unsuccessful payment authorization or other special custom conditions associated with the Content Sever Direct Advantage store configuration.

Note

If you call CART.COMMIT for a cart that has not previously been checked, then that cart is implicitly checked prior to being committed.

The cart object is updated to reflect any errors or notifications retrieved from the Commerce Connector. Choices for fields which have choices also become available.

CART.CREATE

Creates an empty cart object.

Syntax

```
<CART.CREATE NAME="cartname"/>
```

Parameters

NAME (required)

Input and output parameter. As input, the name of the cart object to create on output.

CART.DELETEITEM

Deletes an item from a cart.

Syntax

```
<CART.DELETEITEM NAME="cartname" ID="itemid"
[LISTVARNAME="listvar"]/>
```

Parameters

NAME (required)

Input parameter. Name of the shopping cart.

ID (required)

Input parameter. Row identifier that specifies the item.

LISTVARNAME (optional)

Input and output parameter. As input, a list variable that, if included, holds all the parameters of the item in the deleted row on output.

Example

```
<!-- Is there an item to delete? -->
<IF COND="IsVariable.delitem=true"><THEN>
  <!-- If yes, delete the item from the shopping cart -->
  <CART.DELETEITEM NAME="thiscart" ID="Variables.delitem"/>
  <!-- Save the cart to the commerce context -->
  <COMMERCECONTEXT.SETCURRENTCART CART="thiscart"/>
</THEN></IF>
```

See Also

[CART.GETPARAMETERS](#)

CART.GETCARTDISCOUNTS

Obtains a list of the current discounts.

Syntax

```
<CART.GETCARTDISCOUNTS NAME="cartname" LISTVARNAME="varname" />
```

Parameters

NAME (required)

Input parameter. Name of the cart object.

LISTVARNAME (required)

Input and output parameter. As input, name of a list to create on output with the current discounts. This list has the following columns:

- bucket – Identifier for the type of discount.
- storeid – Store for which to apply this discount
- description – the description of the discount.
- value – the currency value of the discount.

CART.GETCARTDISCOUNTTOTAL

Obtains a total of all discounts applied to the cart.

Syntax

```
<CART.GETCARTDISCOUNTTOTAL NAME="cartname" [STOREID="storeid"]  
VARNAME="varname" />
```

Parameters

NAME (required)

Input parameter. Name of the cart object.

STOREID (optional)

Input parameter. ID of the store for which a discount total should be retrieved, if the cart represents more than one store. If unspecified, discount totals for all stores are retrieved.

VARNAME (required)

Input and output parameter. As input, name of a variable that holds the total, which is assigned as a double value on output.

Description

The total obtained is discount only and does not include any item, shipping, or tax amounts.

CART.GETERRORS

Obtains the cart parameter errors reported the last time the cart was checked.

Syntax

```
<CART.GETERRORS NAME="cartname" [STOREID="storeid"]  
LISTVARNAME="listname" />
```

Parameters

NAME (required)

Input parameter. Name of the cart object.

STOREID (optional)

Input parameter. ID of the store for which errors should be retrieved, if the cart represents more than one store. If unspecified, errors for all stores are retrieved.

LISTVARNAME (required)

Input and output parameter. As input, name of the list to create on output with the error information. This list has three columns:

- field – the name of the parameter with which the error is associated
- description – locale-specific error text
- errorcode – an integer corresponding to the error condition

CART.GETFINALTOTAL

Obtains the final cart total.

Syntax

```
<CART.GETFINALTOTAL  
  NAME="cartname"  
  VARNAME="varname"  
  [STOREID="storeid"] />
```

Parameters

NAME (required)

Input parameter. Name of the cart object.

VARNAME (required)

Input and output parameter. As input, name of a variable that holds the total, which is assigned as a double value on output.

STOREID (optional)

Input parameter. ID of the store where the total applies, if the cart represents more than one store. If STOREID is not included, returns the item total for all stores.

Description

When called, CART.GETFINALTOTAL returns a final total that includes items and discounts, tax, and shipping. Always returns 0 (zero) until the cart has been checked or committed.

CART.FROMSTRING

Creates a cart object from a text string previously written by CART.TOSTRING.

Syntax

```
<CART.FROMSTRING NAME="objname" VARNAME="varname" />
```

Parameters

NAME (required)

Input and output parameter. As input, name of the cart object to create on output.

VARNAME (required)

Input parameter. Name of the string previously written by CART.TOSTRING.

CART.GETITEMDISCOUNTS

Obtains a list of current discounts.

Syntax

```
<CART.GETITEMDISCOUNTS NAME="cartname" ID="itemid"  
LISTVARNAME="varname" />
```

Parameters

NAME (required)

Name of the cart object.

ID (required)

Input parameter. Row identifier that specifies the item.

LISTVARNAME (required)

Input and output parameter. As input, name of a list to create on output with the current discounts. This list has the following columns:

- bucket – an identifier for the type of discount.
 - storeid – the store for which to apply this discount.
 - description – the description of the discount.
 - value – the currency value of the discount.
-

CART.GETITEMDISCOUNTTOTAL

Obtains a list of the item discount totals.

Syntax

```
<CART.GETITEMDISCOUNTTOTAL NAME="cartname" VARNAME="varname" />
```

Parameters

NAME (required)

Input parameter. Name of the cart object.

VARNAME (required)

Input and output parameter. As input, name of a variable to hold the total of the item discount on output.

CART.GETITEMERRORS

Obtains the errors for an order item.

Syntax

```
<CART.GETITEMERRORS NAME="cartname" ID="itemid"
LISTVARNAME="varlist"/>
```

Parameters

NAME (required)

Input parameter. Name of the shopping cart.

ID (required)

Input parameter. Row identifier that specifies the item.

LISTVARNAME (required)

Input and output parameter. As input, name of the list to create on output with the following columns:

- field – item parameter where the error occurred.
- description – information about the error.
- errorcode – the error encountered.

CART.GETITEMLEGALVALUES

Obtains an item parameter's legal values.

Syntax

```
<CART.GETITEMLEGALVALUES NAME="cartname" ID="itemid"
FIELD="fieldid"
LISTVARNAME="varlist"/>
```

Parameters

NAME (required)

Input parameter. Name of the shopping cart.

ID (required)

Input parameter. Row identifier that specifies the item.

FIELD (required)

Input parameter. Name of the item parameter.

LISTVARNAME (required)

Input and output parameter. As input, name of the list to create on output with the following columns:

- name – is presented to the end user
- value – is returned to set the cart parameter field.

Description

There are no fixed choices for item parameters. See [Chapter 12, “Order Item Parameters,”](#) for item parameter names and values.

CART.GETITEMPARAMETER

Obtains a single item parameter.

Syntax

```
<CART.GETITEMPARAMETER NAME="cartname" ID="itemid" FIELD="fieldid"
  VARNAME="varparam"/>
```

Parameters

NAME (required)

Input parameter. Name of the shopping cart.

ID (required)

Input parameter. Row identifier that specifies the item.

FIELD (required)

Input parameter. Name of the item parameter.

VARNAME (required)

Input and output parameter. As input, name of a variable that holds the parameter value returned on output for the specified order item.

Description

See [Chapter 12, “Order Item Parameters,”](#) for item parameter names and values.

CART.GETITEMPARAMETERS

Obtains all item parameters for the item.

Syntax

```
<CART.GETITEMPARAMETERS NAME="cartname" ID="itemid"
  LISTVARNAME="varlist"/>
```

Parameters

NAME (required)

Input parameter. Name of the shopping cart.

ID (required)

Input parameter. Row identifier that specifies the item.

LISTVARNAME (required)

Input and output parameter. As input, name of the list to create on output with the following columns:

- name – name of item parameter

- value – value of item parameter

Description

See [Chapter 12, “Order Item Parameters,”](#) for item parameter names and values.

CART.GETITEMS

Obtains all items in the cart.

Syntax

```
<CART.GETITEMS NAME="cartname" [STOREID="storeid"
LIST="inputlist"]
LISTVARNAME="varlist"/>
```

Parameters

NAME (required)

Input parameter. Name of the shopping cart.

STOREID (optional)

Input parameter. ID of the store where the items exist, if the cart represents more than one store. If STOREID is not included, returns all items for all stores.

LIST (optional)

Input parameter. Describes the minimum input parameters. By default, only the six fixed columns are returned, and any values set for items.

LISTVARNAME (required)

Input and output parameter. As input, name of the list to create on output with the following columns: rowid, assettype, assetid, storeid, quantity, price, *parametername*

Description

See [Chapter 12, “Order Item Parameters,”](#) for item parameter names and values.

Example

```
<!-- Build a resultset list that holds all item parameters. -->
<LISTOBJECT.CREATE NAME="itemlist" COLUMNS="value"/>
<LISTOBJECT.ADDROW NAME="itemlist" value="name"/>
<LISTOBJECT.ADDROW NAME="itemlist" value="uniqueid"/>
<LISTOBJECT.TOLIST NAME="itemlist" value="itemlist"/>
<!-- Get any existing items from the cart. -->
<CART.GETITEMS NAME="thiscart" LIST="itemlist"
LISTVARNAME="cartlist"/>
```

CART.GETITEMTOTAL

Obtains the item total of the cart.

Syntax

```
<CART.GETITEMTOTAL NAME="cartname" VARNAME="varname"  
  [STOREID="storeid"]/>
```

Parameters

NAME (required)

Input parameter. Name of the shopping cart.

VARNAME (required)

Input and output parameter. As input, name of a variable that holds the total, which is assigned as a double value on output.

STOREID (optional)

Input parameter. ID of the store where the total applies, if the cart represents more than one store. If STOREID is not included, returns the item total for all stores.

Description

When called, CART.GETITEMTOTAL returns the total of items only; the total does not include discounts, tax, or shipping.

Example

```
<!-- Get the cart item total -->  
<CART.GETITEMTOTAL NAME="cart" VARNAME="carttotal"/>  
<!-- Convert the total to currency. -->  
<CURRENCY.GETCURRENCY NAME="curval" VALUE="Variables.carttotal"  
  VARNAME="curout"/>
```

CART.GETLEGALVALUES

Obtains the legal values for a parameter from the cart.

Syntax

```
<CART.GETLEGALVALUES NAME="cartname" FIELD="fieldname"  
  [STOREID="storeid"] LISTVARNAME="varname"/>
```

Parameters

NAME (required)

Input parameter. Name of the shopping cart.

FIELD (required)

Input parameter. Name of the cart parameter for which to obtain legal values.

STOREID (optional)

Input parameter. ID of the store for which legal values should be retrieved, if the cart represents more than one store. Use it to retrieve shipping-related legal values. May also be used to retrieve payment-related legal values. It is ignored otherwise.

LISTVARNAME (required)

Input and output parameter. As input, name of the list to create on output with the following columns:

- name – is presented to the end user
- value – is returned to set the cart parameter field.

Description

The legal values returned by `CART.GETLEGALVALUES` may be supplied by Transact, and may be incomplete until the cart is checked. See [Chapter 11, “Shopping Cart Parameters,”](#) for cart parameter names and values. No list is created if the cart has not yet been checked, or if there are no fixed possibilities for the parameter.

Example

```
<!-- Get the current cart. -->
<COMMERCECONTEXT.GETCURRENTCART VARNAME="thiscart"/>
<!-- Check the cart to enumerate payment methods. -->
<CART.CHECK NAME="thiscart" VARNAME="cartstatus"/>
<!-- Now get the legal values for the payment types. -->
<CART.GETLEGALVALUES NAME="thiscart" FIELD="paymentInfo-brandID"
  LISTVARNAME="paylist"/>
```

CART.GETPARAMETER

Obtains the current value for a specified generic cart-level parameter.

Syntax

```
<CART.GETPARAMETER NAME="cartname" FIELD="fieldname"
  [STOREID="storeid"] VARNAME="varname"/>
```

Description

Note that the variable created to hold the value is cleared to null if the field doesn't exist in the cart. See [Chapter 11, “Shopping Cart Parameters,”](#) for cart parameter names and values.

Parameters**NAME (required)**

Input parameter. Name of the cart object.

FIELD (required)

Input parameter. Name of the cart parameter for which to the current value.

STOREID (optional)

Input parameter. ID of the store for which a parameter should be retrieved, if the cart represents more than one store. Use it to retrieve shipping-related values that are store-dependent. It is ignored otherwise.

VARNAME (required)

Input and output parameter. As input, name of the variable to hold the value assigned on output.

CART.GETPARAMETERS

Obtains the current parameter selections for the cart.

Syntax

```
<CART.GETPARAMETERS NAME="cartname" LISTVARNAME="listname"/>
```

Parameters

NAME (required)

Input parameter. Name of the cart object.

LISTVARNAME (required)

Input and output parameter. As input, name of the list to create on output with name and value columns.

Description

See [Chapter 11, “Shopping Cart Parameters,”](#) for cart parameter names and values.

CART.GETPRELIMINARYTOTAL

Obtains the preliminary cart total.

Syntax

```
<CART.GETPRELIMINARYTOTAL NAME="cartname" VARNAME="varname"  
  [STOREID="storeid"]/>
```

Parameters

NAME (required)

Input parameter. Name of the cart object.

VARNAME (required)

Input and output parameter. As input, name of a variable that holds the preliminary total, which is assigned as a double value on output.

STOREID (optional)

Input parameter. ID of the store where the total applies, if the cart represents more than one store. If STOREID is not included, returns the item total for all stores.

Description

The total returned by CART.GETPRELIMINARYTOTAL includes all items and discounts, but not tax and shipping.

CART.GETSHIPPINGTOTAL

Obtains the shipping total.

Syntax

```
<CART.GETSHIPPINGTOTAL NAME="cartname" VARNAME="varname"  
  [STOREID="storeid"]/>
```

Parameters

NAME (required)

Input parameter. Name of the cart object.

VARNAME (required)

Input and output parameter. As input, name of a variable that holds the shipping total, which is assigned as a double value on output.

STOREID (optional)

Input parameter. ID of the store where the shipping total applies, if the cart represents more than one store. If STOREID is not included, returns the shipping total for all stores.

Description

The total returned by CART.GETSHIPPINGTOTAL includes shipping, but not items and discounts and tax. Always returns 0 until the cart has been either checked or committed.

CART.GETSTORES

Obtains a list of stores for which items in the cart are assigned.

Syntax

```
<CART.GETSTORES NAME="cartname" LISTVARNAME="listname"/>
```

Parameters

NAME (required)

Input parameter. Name of the cart object.

LISTVARNAME (required)

Input and output parameter. As input, name of the list to create on output, containing store identifiers in the single column value.

Description

When called, CART.GETSTORES returns Transact store identifiers for each store for which items exist.

CART.GETTAXTOTAL

Obtains the tax total for the cart.

Syntax

```
<CART.GETTAXTOTAL NAME="cartname" VARNAME="varname"  
  [STOREID="storeid"]/>
```

Parameters

NAME (required)

Input parameter. Name of the cart object.

VARNAME (required)

Input and output parameter. As input, name of a variable that holds the tax total, which is assigned as a double value on output.

STOREID (optional)

Input parameter. ID of the store where the tax total applies, if the cart represents more than one store. If STOREID is not included, returns the tax total for all stores.

Description

When called, CART.GETTAXTOTAL returns a total that includes taxes, but not items, discounts, and shipping. Always returns zero until the cart has been checked or committed.

CART.GETTRANSACTIONID

Obtains the Transact transaction identifier from the Commerce Connector for this cart.

Syntax

```
<CART.GETTRANSACTIONID NAME="cartname" VARNAME="varname"/>  
[STOREID="storeid"]/>
```

Parameters

NAME (required)

Input parameter. Name of the cart object.

VARNAME (required)

Input and output parameter. As input, name of a variable that holds the transaction identifier assigned on output.

STOREID (optional)

Input parameter. ID of the store to which the transaction applies, if the cart represents more than one store. Ignored otherwise.

Description

The returned value is valid only if the cart has been committed.

CART.GETTRANSACTIONIDS

Obtains the transaction identifiers from the Commerce Connector for this cart.

Syntax

```
<CART.GETTRANSACTIONIDS NAME="cartname" LISTVARNAME="varname"/>
```

Parameters

NAME (required)

Input parameter. Name of the cart object.

LISTVARNAME (required)

Input and output parameter. As input, name of the list to create on output, containing transaction identifiers in the single column value.

Description

This tag is similar to [CART.GETTRANSACTIONID](#), but applies to multistore shopping carts. The returned values are valid only if the cart has been committed.

CART.MERGE

Merges an older cart into the current cart object.

Syntax

```
<CART.MERGE NAME="currentcart" CART="oldcart"/>
```

Parameters

NAME (required)

Input parameter. Name of the current cart object.

CART (required)

Input parameter. Name of the older cart object.

CART.SETITEMPARAMETERS

Sets order item parameters.

Syntax

```
<CART.SETITEMPARAMETERS NAME="cartname" ID="itemid"  
LIST="listname"/>
```

Parameters

NAME (required)

Input parameter. Name of the cart object.

ID (required)

Input parameter. Row identifier that specifies the item.

LIST (required)

Input parameter. List of name/value pairs to set on output for the specified order item.

Description

See [Chapter 12, “Order Item Parameters,”](#) for item parameter names and values.

CART.SETITEMQUANTITY

Sets an item's quantity and price information.

Syntax

```
<CART.SETITEMQUANTITY NAME="cartname" ID="itemid"  
  QUANTITY="itemquantity"  
  PRICE="itemprice" />
```

Parameters

NAME (required)

Input parameter. Name of the cart object.

ID (required)

Input parameter. Row identifier that specifies the item.

QUANTITY (required)

Input parameter. Number of the specified item to be set on output.

PRICE (required)

Input parameter. Price of the specified item to be set on output.

CART.SETPARAMETER

Sets a single shopping cart parameter.

Syntax

```
<CART.SETPARAMETER NAME="cartname" FIELD="field" [VALUE="value"  
  STOREID="storeid"] />
```

Description

The parameter set by this tag is a name/value pair whose interpretation is determined by the Commerce Connector. See [Chapter 11, “Shopping Cart Parameters,”](#) for cart parameter names and values.

Parameters

NAME (required)

Input parameter. Name of the cart object.

FIELD (required)

Input parameter. Name of the cart parameter to set.

VALUE (optional)

Input parameter. Value of the cart parameter to be set on output. If not included, the field value is undefined.

STOREID (optional)

Input parameter. Store for which the parameter is to be set on output. It is ignored otherwise.

CART.SETPARAMETERS

Sets the specified shopping cart parameters.

Syntax

```
<CART.SETPARAMETERS NAME="cartname" LIST="listname" />
```

Parameters

NAME (required)

Input parameter. Name of the cart object.

LIST (required)

Input parameter. Name of the list that holds the requested parameter values. The list has name and value columns.

Description

The parameters set by this tag are name/value pairs that are interpreted by the Commerce Connector. See [Chapter 11, “Shopping Cart Parameters,”](#) for cart parameter names and values.

CART.TOSTRING

Writes the cart object into a text string.

Syntax

```
<CART.TOSTRING NAME="objname" VARNAME="varname" />
```

Parameters

NAME (required)

Input parameter. Name of the cart object.

VARNAME (required)

Input and output parameter. As input, name of the string to which to write the cart object on output. The string is long, that is, measured in kilobytes.

Cart Set Tags

You use cart set tags to store, retrieve, delete, and list stored cart objects for a registered buyer.

The visitor (or buyer) for whom the cart set is maintained is identified by a `userID`, a string normally obtained from [VDM.GETCOMMERCEID](#) or [CSUSER.LOOKUP](#).

Note

Cart set tags cannot be used for unregistered or walk-in buyers.

The following section describes each cart set tag, beginning with [CARTSET.ADDCART](#).

CARTSET.ADDCART

Adds a new cart to the cart set.

Syntax

```
<CARTSET.ADDCART  
    NAME="cartset"  
    CARTID="cartid"  
    CART="newcart"/>
```

Parameters

NAME (required)

Input parameter. Name of the cart set to interrogate. A previous call to `CARTSET.GETSET` is required to get this parameter.

CARTID (required)

Input parameter. Identifier to use for the cart. If a cart with the same identifier already exists in the cart set, it is replaced on output.

CART (required)

Input parameter. Cart object to add to the cart set on output.

CARTSET.CARTIDS

Returns a list of identifiers for carts within the cart set.

Syntax

```
<CARTSET.CARTIDS NAME="cartset" LISTVARNAME="listname"/>
```

Parameters

NAME (required)

Input parameter. Name of the cart set to interrogate. A previous call to `CARTSET.GETSET` is required to get this parameter.

LISTVARNAME (required)

Input and output parameter. List variable that holds the identifiers for the carts in the cart set. The list has a single column, value.

CARTSET.DELETECART

Deletes a cart from the cart set.

Syntax

```
<CARTSET.DELETECART NAME="cartset" CARTID="cartid" />
```

Parameters

NAME (required)

Input parameter. Name of the cart set to interrogate. A previous call to CARTSET.GETSET is required to get this parameter.

CARTID (required)

Input parameter. Identifier of the cart to be deleted. The specified cart is removed from the cart set on output.

CARTSET.GETCART

Retrieves a stored cart from the cart set.

Syntax

```
<CARTSET.GETCART NAME="cartset" CARTID="cartid" CART="cartvar" />
```

Parameters

NAME (required)

Input parameter. Name of the cart set to interrogate. A previous call to CARTSET.GETSET is required to get this parameter.

CARTID (required)

Input parameter. Identifier of the cart to be retrieved.

CART (required)

Input and output parameter. Name of the variable to fill with the retrieved cart object on output.

CARTSET.GETSET

Retrieves the current visitor's cart set, or else creates a new empty cart set.

Syntax

```
<CARTSET.GETSET NAME="cartset" USERID="userid" />
```

Parameters

NAME (required)

Input and output parameter. Variable name in which to store the retrieved or created cart set. This NAME parameter is required for calls to all other cart set tags.

USERID (required)

Input parameter. Identifier for the user.

Commerce Context Tags

During a visitor session, CS-Direct Advantage creates and saves a **commerce context** for the visitor. Current shopping cart is maintained by the commerce context.

You use the commerce context XML tags to save cart objects to and retrieve objects from the commerce context. Shopping cart objects, when present, are persistent throughout the visitor session (that is, they are stored in the Content Server database and then flushed when the session ends).

The commerce context maintains the following information about CS-Engage assets:

- List of segments for which the visitor qualifies
- List of promotions for which the visitor qualifies
- Time object that is used for calculating time-based rules
- Utility object that gives you access to product attributes

There are additional tags available with CS-Engage for displaying recommendations and promotions on your application pages. See [Chapter 4, “CS-Engage XML Tags,”](#) in this guide for more information, as well as the *CSEE Developer’s Guide*.

COMMERCECONTEXT.GETABANDONEDCARTSESSIONS

Gets a list of sessions that have abandoned carts.

Syntax

```
<COMMERCECONTEXT.GETABANDONEDCARTSESSIONS [STARTDATE="date"]
  ENDDATE="date" LISTVARNAME="listvar"/>
```

Parameters

STARTDATE (optional)

Input parameter. Starting date, as a epoch-based timestamp. 0 is presumed if this is not provided.

ENDDATE (required)

Input parameter. Ending date, as a epoch-based timestamp.

LISTVARNAME (required)

Input and output parameter. As input, name of a list that holds the sessions that have abandoned carts. The list is created on output with two columns: odate (which is the timestamp describing the last time the cart was updated for that session) and osession.

Description

Abandoned carts are carts that were last modified for the session between a specified date range.

COMMERCECONTEXT.GETCURRENTCART

Retrieves and names a copy of the current cart from the visitor context.

Syntax

```
<COMMERCECONTEXT.GETCURRENTCART VARNAME="cartname"/>
```

Parameters

VARNAME (required)

Input or output parameter. Name of the cart object to retrieve or create.

Description

If there is already a copy of the current cart with this name, it is retrieved. If a copy of the current cart does not yet exist, the current cart object is copied and named with the variable name.

You can modify settings, data, and functions in the copy and then check it back into the visitor context with `COMMERCECONTEXT.SETCURRENTCART`.

Example

```
<!-- Get the current cart. -->
<COMMERCECONTEXT.GETCURRENTCART VARNAME="thiscart"/>
<!-- Check the cart to enumerate payment methods. -->
<CART.CHECK NAME="thiscart" VARNAME="cartstatus"/>
<!-- Now get the legal values for the payment types. -->
<CART.GETLEGALVALUES NAME="thiscart" FIELD="paymentInfo-brandID"
  LISTVARNAME="paylist"/>
```

COMMERCECONTEXT.GETSESSIONCART

Obtains a cart given a session identifier (visitor id).

Syntax

```
<COMMERCECONTEXT.GETSESSIONCART SESSION="sessionid"
  VARNAME="cartname"/>
```

Parameters

SESSION (required)

Input parameter. Session identifier.

VARNAME (required)

Input parameter. Name of the cart object to retrieve.

COMMERCECONTEXT.LOGOUT

Discards the current visitor ID and creates a new one.

Syntax

```
<COMMERCECONTEXT . LOGOUT />
```

Parameters

None.

Description

When visitors access a site implemented with CS-Engage, they are automatically assigned a visitor ID. This tag clears the current visitor ID and assigns a new one. Typical use for this method is to use it to create a logout button so a new visitor can interact with the site without having to end the session.

If the current visitor is logged into Transact or any other external database through Commerce Connector, this method automatically clears this visitor's commerce ID or access ID as well, which disconnects the visitor from Commerce Connector.

COMMERCECONTEXT.SAVEALL

Saves a copy of everything in the commerce context to the Content Server database for the duration of the session (that is, to persistent storage).

Syntax

```
<COMMERCECONTEXT . SAVEALL />
```

Parameters

None.

Description

Because the visitor objects are persistent, the only use for this tag is if you want to store other context objects (that must be explicitly saved) that you created during the session.

COMMERCECONTEXT.SETCURRENTCART

Checks the working copy of the current cart back into the visitor context, updating the current cart that is stored there.

Syntax

```
<COMMERCECONTEXT.SETCURRENTCART CART="cartname" />
```

Parameters

CART (required)

Input parameter. Name of the working copy of the cart that updates the current cart object.

Description

For more information about working with carts, see the *CSEE Developer's Guide*.

Example

```
<!-- Is there an item to delete? -->
<IF COND="IsVariable.delitem=true"><THEN>
  <!-- If yes, delete the item from the shopping cart -->
  <CART.DELETEITEM NAME="thiscart" ID="Variables.delitem"/>
  <!-- Save the cart to the commerce context -->
  <COMMERCECONTEXT.SETCURRENTCART CART="thiscart"/>
</THEN></IF>
```


CSMAC Tags

You use CSMAC tags to generate Message Authentication Codes (MACs) that can be interpreted by Transact, and to produce encrypted cookie values to identify individual visitors to Transact via the Commerce Connector.

Note

[CSMAC.STRING](#) and [CSMAC.PAYLOAD](#) produce different VARNAME values from the same input value:

- CSMAC.STRING calculates MAC based on value
- CSMAC.PAYLOAD calculates MAC based on the following:
'value + "&kid=" + storeID + "." + currentKeyNumStr + "&ss=env"'

The following section describes each CSMAC tag, beginning with [CSMAC.COOKIEVALUE](#).

CSMAC.COOKIEVALUE

Produces a cookie value that the Web designer can insert into the visitor's browser to enable Transact to recognize the user without requiring additional authentication.

Syntax

```
<CSMAC.COOKIEVALUE
  USERID="userid"
  STOREID="storeid"
  [SECONDS="duration"]
  VARNAME="varcookie"/>
```

Parameters

USERID (required)

Input parameter. Identifier of the user as known to Transact. You can get this value by calling VDM.GETCOMMERCEID.

STOREID (required)

Input parameter. Identifier of the Transact store whose secret keys are used to generate the MAC.

SECONDS (optional)

Input parameter. Duration in seconds (from now) for which Transact honors the cookie. If omitted, the duration is 600 (10 minutes).

VARNAME (required)

Input and output parameter. Name of a variable that holds the cookie string output with the following form:

```
"MAC:principal_id=userid&expiration=<now+durationSeconds>&kid=storeid.
<keyID>&ss=env"
```

Description

Successful usage of such a cookie is limited to configurations where Transact and CS-Direct Advantage reside in the same DNS domain. Typically, the value returned is used with the Content Server [SETCOOKIE](#) tag.

CSMAC.PAYLOAD

Generates a MAC query string value for a string resembling a name-value pair URL query string, with a specific store's current key.

Syntax

```
<CSMAC.PAYLOAD
  VALUE="stringval"
  STOREID="storeid"
  VARNAME="payloadvar" />
```

Parameters

VALUE (required)

Input parameter. String from which the MAC should be calculated. It should be in the name-value pair format: "name1=va l1&name2=va l2" This is typically used in URL query strings.

Note

Use the character code `&` as input instead of the ampersand character (&).

STOREID (required)

Input parameter. Identifier of the Transact store whose secret keys are used to generate the MAC.

VARNAME (required)

Input and output parameter. Name of a variable to write to with the payload string of the form "MAC:value&kid=storeid.keyID&ss=env". For example, with a value of name1=va l1 and storeID of 400003, the following could be returned (depending on current key values):

```
"Ha3857e216c5523843495550f4834908c:name1=va l1&kid=400003.4&ss=e
nv"
```

CSMAC.STRING

Generates a MAC string for any string with a specific store's current key.

Syntax

```
<CSMAC.STRING
    VALUE="stringval"
    STOREID="storeid"
    VARNAME="stringvar" />
```

Parameters

VALUE (required)

Input parameter. String from which the MAC should be calculated.

STOREID (required)

Input parameter. Identifier of the Transact store whose secret key is used to generate the MAC.

VARNAME (required)

Input and output parameter. Name of a variable to write to with the MAC string of the form "MAC:kid=storeID.keyID". For example, with a value of name1=va l1 and storeID of 400003, the following could be returned (depending on current key values):

```
"Ha3857e21f48349086c5523843495550c:kid=400003.4"
```

CSMAC.TRANSACTIONSERVER

Returns a string with the form *protocol://hostname:port* that is the URL prefix of the configured Transact.

Syntax

```
<CSMAC.TRANSACTIONSERVER
    [PROTOCOL="http|https"]
    VARNAME="stringvar" />
```

Parameters

PROTOCOL (optional)

Input parameter. Either http or https. If PROTOCOL is omitted, the default protocol for contacting Transact is taken from the `transact.ini` file.

VARNAME (required)

Input and output parameter. As input, name of a variable to write with the URL prefix string on output.

Description

The returned string contains the protocol, hostname, and possibly port number, but does not contain any pathname or query string portions of the URL. If a standard default port is configured for either http (80) or https (443), this is omitted from the returned string. For example, depending on values in the `transact.ini` file, the following URL string might be returned: `https://release.openmarket.com:8081`

CSUSER Tags

You use CSUSER tags to produce, obtain, and maintain information about Transact-registered buyers. The CSUSER tags require the Commerce Connector for communication with Transact.

You set and retrieve individual buyer parameters in a manner similar to that used for shopping cart parameters.

Note

See [Chapter 13, “Buyer Parameters,”](#) for buyer parameter names and values. See [Chapter 11, “Shopping Cart Parameters,”](#) for cart parameter names and values.

The following section describes each CSUSER tag, beginning with [CSUSER.AUTHENTICATE](#).

CSUSER.AUTHENTICATE

Retrieves user information from the Transact database.

Syntax

```
<CSUSER.AUTHENTICATE
  NAME="csuser"
  VARNAME="booleanvar"/>
```

Parameters

NAME (required)

Input parameter. Name of the csuser object.

VARNAME (required)

Input parameter. A variable that holds a string representing boolean status:

- true – Buyer has been retrieved from Transact.
- false – User could not be authenticated, or an error has occurred. Call [CSUSER.GETERRORS](#) to determine the problem.

Description

A form is typically presented to the site visitor asking for an `accessName` and `password`. Use a secure URL (`https`) with this form to guarantee the secrecy of the visitor's password.

Both the `accessName` parameter and the `password` parameter must be set within the `csuser` object to identify the user. This differs from `lookup` insofar as this tag provides a means of confirming that the buyer knows his password.

The values supplied by the visitor are set in the `csuser` object with a call to either [CSUSER.SETPARAMETER](#) or [CSUSER.SETPARAMETERS](#). If a call to [CSUSER.AUTHENTICATE](#) returns `true`, you then use the `userId` parameter in the `csuser` object to call [VDM.SETCOMMERCEID](#).

CSUSER.CLEAR

Clears the parameters and data in a csuser object.

Syntax

```
<CSUSER.CLEAR  
  NAME="csuser"/>
```

Parameters

NAME (required)

Input parameter. Name of the csuser object to clear.

CSUSER.CREATE

Creates an empty csuser object.

Syntax

```
<CSUSER.CREATE  
  NAME="csuser"/>
```

Parameters

NAME (required)

Input parameter. Name of the csuser object to create.

CSUSER.CREATEPMTACCT

Creates a new payment account in Transact for the csuser object.

Syntax

```
<CSUSER.CREATEPMTACCT  
  NAME="csuser"  
  VARNAME="booleanvar"  
  LIST="listname"/>
```

Parameters

NAME (required)

Input parameter. Name of the csuser object.

VARNAME (required)

Input parameter. A variable that holds a string representing boolean status:

- true – Payment account has been registered with Transact.
- false – An error has occurred. Call [CSUSER.GETERRORS](#) to determine the problem.

The variable is set to indicate whether the payment account was created on Transact.

LIST (required)

Input parameter. List that holds the payment method parameters. The list has two columns: name and value, where each parameter name listed below is one half of a name/value pair, and each name/value pair is a row in the list.

- name – (optional) A user-supplied name for the account.
- brandID – Credit card/switch brand ID (GUID) from `getAvailablePaymentBrands()`.
- ccn – Credit/switch card number, or equivalent.
- expirationDate – in mm/yy format.
- billToAddress – (optional) billing address for account. The default is the buyer's address. If any value is set, then all address fields must be set. The address must be specified in nine rows using the following standard fields:

```
billToAddress-name1
billToAddress-name2
billToAddress-address1
billToAddress-address2
billToAddress-city
billToAddress-state
billToAddress-country
billToAddress-postalCode
billToAddress-telephone
```

- issueNumber – (optional) Switch Card Issue Number.
- extraInstrumentInfo – (optional) Japanese brands only.
- poNumber – (optional) Purchase order number for PO account only.
- poComments – (optional) Purchase order comment for PO account only.

CSUSER.DELETEPMTACCT

Deletes a payment account on Transact for the user.

Syntax

```
<CSUSER.DELETEPMTACCT
  NAME="csuser"
  ID="accountid"
  VARNAME="listvar"/>
```

Parameters**NAME (required)**

Input parameter. Name of the csuser object.

ID (required)

Input parameter. The accountID within Transact of the payment account to be deleted.

VARNAME (required)

Input parameter. Variable that holds a string representing boolean status:

- true – Payment account has been deleted from Transact.
- false – An error has occurred. Call [CSUSER.GETERRORS](#) to determine the problem.

CSUSER.FROMSTRING

Creates a csuser object from a text string previously written by [CSUSER.TOSTRING](#).

Syntax

```
<CSUSER.FROMSTRING
    NAME="objname"
    VARNAME="varname"/>
```

Parameters

NAME (required)

Input and output parameter. As input, name of the csuser object to create on output.

VARNAME (required)

Input parameter. Name of the string previously written by [CSUSER.TOSTRING](#).

Example

```
<CSUSER.CREATE NAME="csuser"/>

<CSUSER.SETPARAMETER NAME="csuser" FIELD="accessName"
VALUE="Test"/>

<CSUSER.GETPARAMETER NAME="csuser" FIELD="accessName"
VARNAME="gan"/>

<CSUSER.TOSTRING NAME="csuser" VARNAME="csuserToString"/>

<CSUSER.FROMSTRING NAME="csusergetfrom"
VARNAME="Variables.csuserToString"/>
<CSUSER.GETPARAMETER NAME="csusergetfrom" FIELD="accessName"
VARNAME="outputValue"/>
CSUSER.GETPARAMETER Access Name: <CSVAR
NAME="Variables.outputValue"/><br/>
```

CSUSER.GETALLPMTACCTS

Obtains the current payment parameter selections for the csuser object.

Syntax

```
<CSUSER.GETALLPMTACCTS
    NAME="csuser"
    LISTVARNAME="listvar"/>
```

Parameters

NAME (required)

Input parameter. Name of the csuser object.

LISTVARNAME (required)

Input and output parameter. As input, name of a list to create on output with the following columns:

name, accountID, billToAddress-name, billToAddress-address1, expirationDate, brandID.

CSUSER.GETERRORS

Obtains the errors reported the last time any of the following tags were called for the specified csuser object:

- CSUSER.REGISTER
- CSUSER.MODIFY
- CSUSER.LOOKUP
- CSUSER.AUTHENTICATE
- CSUSER.CREATEPMTACCT
- CSUSER.MODIFYPMTACCT
- CSUSER.DELETEPMTACCT

Syntax

```
<CSUSER.GETERRORS  
    NAME="csuser"  
    LISTVARNAME="listname"/>
```

Parameters

NAME (required)

Input parameter. Name of the csuser object.

LISTVARNAME (required)

Input and output parameter. As input, name of the list to create on output that holds the error information. The list has three columns:

- field – Name of the parameter with which the error is associated.
- description – Locale-specific error text.
- errorcode – Integer corresponding to the error condition.

CSUSER.GETLEGALPMTBRANDS

Obtains the current parameter selections for the csuser object.

Syntax

```
<CSUSER.GETLEGALPMTBRANDS  
    NAME="csuser"  
    LISTVARNAME="listvar"/>
```

Parameters

NAME (required)

Input parameter. Name of the csuser object.

LISTVARNAME (required)

Input and output parameter. As input, name of the list to create on output that holds the error information. The list has two columns:

- name – As is displayed on the screen.
- value – Returned to set the parameter paymentInfo-brandID.

CSUSER.GETPARAMETER

Obtain the current value for a specified generic user parameter.

Syntax

```
<CSUSER.GETPARAMETER
  NAME="csuser"
  FIELD="paramname"
  VARNAME="varname"/>
```

Parameters

NAME (required)

Input parameter. Name of the csuser object.

FIELD (required)

Input parameter. Name of the parameter.

VARNAME (required)

Input and output parameter. As input, name of the variable to create on output with the result. The variable is cleared to null if the field does not exist.

Description

See [Chapter 13, “Buyer Parameters,”](#) for parameter names and values.

CSUSER.GETPARAMETERS

Obtains the current parameter selections for the user.

Syntax

```
<CSUSER.GETPARAMETERS
  NAME="csuser"
  LISTVARNAME="listname"/>
```

Parameters

NAME (required)

Input parameter. Name of the csuser object.

LISTVARNAME (required)

Input and output parameter. As input, name of the list to create on output. The list has two columns: name and value.

Description

See [Chapter 13, “Buyer Parameters,”](#) for parameter names and values.

CSUSER.LOOKUP

Verifies if the user exists in the Transact system.

Syntax

```
<CSUSER.LOOKUP  
    NAME="csuser"  
    VARNAME="booleanvar"/>
```

Parameters

NAME (required)

Input parameter. Name of the csuser object.

VARNAME (required)

Input parameter. Variable that holds a string representing boolean status:

- true – Buyer exists in Transact.
- false – Buyer does not exist in Transact, or an error has occurred. Use [CSUSER.GETERRORS](#) to determine the problem.

Description

Either the csuser object's `accessName` or the `userID` parameter must be set to identify the buyer. If both are set, `userID` is used. The csuser object is updated with the corresponding data from Transact. After lookup, use [VDM.SETCOMMERCEID](#) with the `userID` parameter as input.

See Also

[CSUSER.AUTHENTICATE](#)

CSUSER.MODIFY

Modifies buyer data on Transact.

Syntax

```
<CSUSER.MODIFY  
    NAME="csuser"  
    VARNAME="booleanvar"/>
```

Parameters

NAME (required)

Input parameter. Name of the csuser object.

VARNAME (required)

Input parameter. Variable that holds a string representing boolean status:

- true – Buyer has been modified with Transact.
- false – An error has occurred. Use [CSUSER.GETERRORS](#) to determine the problem.

Description

The userID parameter must be set for the csuser object. Any other buyer parameters set in the csuser object replace the corresponding parameters in the buyer data record on Transact, with the following exceptions:

- The accessName parameter cannot be changed.
- If set, the password parameter must differ from the buyer's current password on Transact.
- If any address field is modified, then all address fields must be modified. The address must be specified in nine rows using the following standard fields:
 - billToAddress-name1
 - billToAddress-name2
 - billToAddress-address1
 - billToAddress-address2
 - billToAddress-city
 - billToAddress-state
 - billToAddress-country
 - billToAddress-postalCode
 - billToAddress-telephone

CSUSER.MODIFYPMTACCT

Changes payment account data on Transact for the specified buyer.

Syntax

```
<CSUSER.MODIFYPMTACCT
  NAME="csuser"
  LIST="listname"
  VARNAME="booleanvar"/>
```

Parameters

NAME (required)

Input parameter. Name of the csuser object.

LIST (required)

Input parameter. List containing the payment method input parameters. The list has two columns: name and value, where each parameter name listed below is one half of a name/value pair, and each name/value pair is a row in the list.

- name – (optional) A user-supplied name for the account.
- accountID – Credit card/switch brand ID (GUID) from getAvailablePaymentBrands().
- ccn – Credit/switch card number, or equivalent.
- expirationDate – in mm/yy format.
- billToAddress – (optional) billing address for account. The default is the buyer's address. If any value is set, then all address fields must be set. The address must be specified in nine rows using the following standard fields:

```
billToAddress-name1
billToAddress-name2
```

billToAddress-address1
 billToAddress-address2
 billToAddress-city
 billToAddress-state
 billToAddress-country
 billToAddress-postalCode
 billToAddress-telephone

- issueNumber – (optional) Switch Card Issue Number.
- extraInstrumentInfo – (optional) Japanese brands only.
- poNumber – (optional) Purchase order number for PO account only.
- poComments – (optional) Purchase order comment for PO account only.

Note

The ccn and accountID parameters must not be changed. To change the ccn value, you must delete the old payment account (with [CSUSER.DELETEPMTACCT](#)) and create a new one (with [CSUSER.CREATEPMTACCT](#)).

VARNAME (required)

Input parameter. Variable that holds a string representing Boolean status:

- true – Buyer has been modified with Transact.
- false – An error has occurred. Call [CSUSER.GETERRORS](#) to determine the problem.

CSUSER.REGISTER

Registers a new buyer as a csuser object with Transact.

Syntax

```
<CSUSER.REGISTER
  NAME="csuser"
  VARNAME="Booleanvar"/>
```

Parameters

NAME (required)

Input parameter. Name of the csuser object.

VARNAME (required)

Input parameter. Variable that holds a string representing boolean status:

- true – Buyer has been registered with Transact.
- false – An error has occurred. Use [CSUSER.GETERRORS](#) to determine the problem.

Description

As a result of a call to [CSUSER.REGISTER](#), the csuser object is updated with a userID assigned by Transact. After registering, call [VDM.SETCOMMERCEID](#) with the userID parameter as input.

See Also

[CSUSER.AUTHENTICATE](#)

CSUSER.SETPARAMETER

Sets a single buyer parameter.

Syntax

```
<CSUSER.SETPARAMETER
  NAME="csuser"
  FIELD="paramname"
  VALUE="paramvalue"/>
```

Parameters

NAME (required)

Input parameter. Name of the csuser object.

FIELD (required)

Input parameter. Name of the buyer parameter to set.

VALUE (optional)

Input parameter. Value of the buyer parameter. If not included, field becomes unset.

Description

The name/value pair is subject to Commerce Connector interpretation.

CSUSER.SETPARAMETERS

Sets the specified buyer parameters.

Syntax

```
<CSUSER.SETPARAMETERS
  NAME="csuser"
  LIST="paramlist"/>
```

Parameters

NAME (required)

Input parameter. Name of the csuser object.

LIST (required)

Input parameter. Name of the parameter list. This list has two columns: name and value.

Description

The name/value pairs are subject to Commerce Connector interpretation. See [Chapter 13, "Buyer Parameters,"](#) for parameter names and values.

CSUSER.TOSTRING

Writes the csuser object into a text string.

Syntax

```
<CSUSER.TOSTRING  
  NAME="objname"  
  VARNAME="varname" />
```

Parameters

NAME (required)

Input parameter. Name of the csuser object.

VARNAME (required)

Input and output parameter. As input, name of the string to which to write the csuser object on output. The string is long, that is, measured in kilobytes.

Currency Tags

Currency tags perform conversions between floating point values and currency strings that can be displayed. Currency tags also perform formatting and rounding operations on currency strings.

The following section describes each currency tag, beginning with [CURRENCY.CREATE](#).

CURRENCY.CREATE

Creates a currency object for a specific currency ISO code.

Syntax

```
<CURRENCY.CREATE
  NAME="curname"
  [ISOCURRENCY="isosymbol"]
  [SEPARATOR="dot|comma|default"]/>
```

Parameters

NAME (required)

Input and output parameter. As input, name of the currency object to create on output.

ISOCURRENCY (optional)

Input parameter. The 3-character ISO currency symbol. If not specified, the default currency symbol (USD) is used.

SEPARATOR (optional)

Input parameter. Specifies the currency separator character. It can have one of the following values:

- dot
- comma
- default

Example

```
<!-- Get the item price, which is stored as (price * quantity). -->
<CALCULATOR.GO VALUE="cartlist.price cartlist.quantity /"
  VARNAME="calcout"/>
<!-- Convert the above item price to currency. -->
<CURRENCY.CREATE NAME="curval"/>
<CURRENCY.GETCURRENCY NAME="curval" VALUE="Variables.calcout"
  VARNAME="curout"/>
```

CURRENCY.GETCURRENCY

Converts a double to a displayable currency string.

Syntax

```
<CURRENCY.GETCURRENCY  
    NAME="curname"  
    VALUE="fpvalue"  
    VARNAME="varname" />
```

Parameters

NAME (required)

Input parameter. Name of the currency object.

VALUE (required)

Input parameter. The floating-point monetary value.

VARNAME (required)

Input and output parameter. As input, the name of the variable that holds the displayable monetary string assigned on output.

Example

```
<!-- Get the item price, which is stored as (price * quantity). --  
>  
<CALCULATOR.GO VALUE="cartlist.price cartlist.quantity /"  
    VARNAME="calcout"/>  
<!-- Convert the above item price to currency. -->  
<CURRENCY.CREATE NAME="curval"/>  
<CURRENCY.GETCURRENCY NAME="curval" VALUE="Variables.calcout"  
    VARNAME="curout"/>
```

CURRENCY.GETISOCODE

Returns the ISO code for the currency instance.

Syntax

```
<CURRENCY.GETISOCODE  
    NAME="curname"  
    VARNAME="varname" />
```

Parameters

NAME (required)

Input parameter. Name of the currency object.

VARNAME (required)

Input and output parameter. As input, the name of the variable that holds the ISO code value assigned on output.

CURRENCY.GETSYMBOL

Returns the symbol typically used to represent currencies of this type.

Syntax

```
<CURRENCY.GETSYMBOL  
    NAME="curname"  
    VARNAME="varname" />
```

Parameters

NAME (required)

Input parameter. Name of the currency object.

VARNAME (required)

Input and output parameter. As input, name of the currency object that holds currency symbol assigned on output.

CURRENCY.READCURRENCY

Converts a number previously rendered as a monetary string back to a decimal value.

Syntax

```
<CURRENCY.READCURRENCY  
    NAME="curname"  
    VALUE="string"  
    VARNAME="varname" />
```

Parameters

NAME (required)

Input parameter. Name of the currency object.

VALUE (required)

Input parameter. Display string.

VARNAME (required)

Input and output parameter. As input, name of the variable that holds floating point value assigned on output.

CURRENCY.ROUND

Rounds a currency value to the nearest legal absolute currency value.

Syntax

```
<CURRENCY.ROUND  
    NAME="curname"  
    VALUE="string"  
    VARNAME="varname" />
```

Parameters

NAME (required)

Input parameter. Name of the currency object.

VALUE (required)

Input parameter. Value to be rounded.

VARNAME (required)

Input and output parameter. As input, name of the variable that holds the rounded floating point value assigned on output.

CURRENCY.ROUNDDOWN

Rounds a currency value downward to the next lower legal absolute currency value.

Syntax

```
<CURRENCY.ROUNDDOWN  
  NAME="currency"  
  VALUE="string"  
  VARNAME="varname" />
```

Parameters

NAME (required)

Input parameter. Name of the currency object.

VALUE (required)

Input parameter. Value to be rounded down.

VARNAME (required)

Input and output parameter. As input, name of the variable that holds the rounded down floating point value assigned on output.

CURRENCY.ROUNDUP

Rounds a currency value upward to the next higher legal absolute currency value.

Syntax

```
<CURRENCY.ROUNDUP  
  NAME="currency"  
  VALUE="string"  
  VARNAME="varname" />
```

Parameters

NAME (required)

Input parameter. Name of the currency object.

VALUE (required)

Input parameter. Value to be rounded up.

VARNAME (required)

Input and output parameter. As input, name of the variable that holds the rounded up floating point value assigned on output.

Description

Here are some examples of how this tag rounds up:

- 10.011 is rounded to 10.02
- -10.011 is rounded to -10.02.

Date Converter Tags

Date tags convert dates with year, month, day, and optional hour, minute, and am/pm fields into epoch format long integers representing milliseconds since Jan 1, 1970, 0:00 GMT. Date tags also convert long integers into dates.

The following section describes each date converter tag, beginning with [DATE.CLOCKLIST](#).

DATE.CLOCKLIST

Converts an epoch format long integer representing milliseconds since Jan 1, 1970, 0:00 GMT, into a single-row list with columns representing various date/time fields and formats.

Syntax

```
<DATE.CLOCKLIST
    LISTVARNAME="listvarname"
    [CLOCK="longinteger"]
    [TIMEZONE="timezone"]/>
```

Note

If you take the result of `DATE.CLOCKLIST` and use it as an input value to [DATE.CONVERT](#), the epoch date returned is different from the originally converted date because `DATE.CLOCKLIST` is 1-based for month and day values, while [DATE.CONVERT](#) is 0-based for month and day values.

Parameters

LISTVARNAME (required)

Input parameter (see as output, below). Name of a list to create with broken out date formats.

CLOCK (optional)

Input parameter. Long integer value of milliseconds since Jan 1, 1970 0:00 GMT. The default is now, or the current clock value.

TIMEZONE (optional)

Input parameter. Time zone name. The default is system default.

Note

Only North American time zones are recognized. For time zones elsewhere in the world, use a plus or minus offset to GMT; for example,

```
TIMEZONE="-3:00"
TIMEZONE="+6:00".
```

LISTVARNAME

Output parameter. A list is created with the following columns:

- `fulldate` – for example, "Saturday, April 6, 2001"

- `fulltime` – for example, "1:05:00 o'clock AM EST"
- `longdate` – for example, "April 6, 2001"
- `longtime` – for example, "1:05:00 AM EST"
- `year` – four-digit year
- `month` – integer month, January=1, December=11
- `dayofmonth` – integer day of month, 1–31
- `dayofweek` – integer day of week 1–7, Sunday=1
- `hour` – integer hour 0–11
- `minute` – integer minute 0–59
- `second` – integer second 0–59
- `millisecond` – integer millisecond 0–999
- `ampm` – integer, am=0, pm=1
- `clock` – integer, milliseconds since Jan 1, 1970 0:00 GMT

DATE.CONVERT

Converts a specific date to an epoch format integer.

Note

If you take the result of `DATE.CLOCKLIST` and use it as an input value to [DATE.CONVERT](#), the epoch date returned is different from the originally converted date because `DATE.CLOCKLIST` is 1-based for month and day values, while [DATE.CONVERT](#) is 0-based for month and day values.

Syntax

```
<DATE.CONVERT
  VARNAME="varname"
  YEAR="year"
  MONTH="month"
  DAY="day"
  [ HOUR="hour" ]
  [ AMPM="ampm" ]
  [ MINUTE="minute" ]
  [ TIMEZONE="timezone" ] />
```

Parameters

VARNAME (required)

Input and output parameter. As input, name of a variable to create with the integer date format on output.

YEAR (required)

Input parameter. The year number.

MONTH (required)

Input parameter. The month number (0-based; that is, 0=January, 11=December).

DAY (required)

Input parameter. The day number (0-based; that is, Sunday=0; Saturday=6.)

HOURL (optional)

Input parameter. The hour number (default is 0).

AMPM (optional)

Input parameter. The am/pm specifier (default is am).

MINUTE (optional)

Input parameter. The minute number (default is 0).

TIMEZONE (optional)

Input parameter. A specified time zone.

Note

Only North American time zones are recognized. For time zones elsewhere in the world, use a plus or minus offset to GMT; for example,

TIMEZONE="-3:00"

TIMEZONE="+6:00"

DATE.DEFAULTTZ

Returns the current time zone in standard international format.

Syntax

```
<DATE.DEFAULTTZ  
  VARNAME="varname"/>
```

Parameters

VARNAME (required)

Input and output parameter. As input, name of a variable to hold the current time zone on output.

Decimal Tags

The decimal tags are used for formatting floating point values as decimal objects in different locales.

The following section describes each decimal tag, beginning with [DECIMAL.CREATE](#).

DECIMAL.CREATE

Creates a decimal mapping object, which you can use to convert between a floating point value and a displayable string.

Syntax

```
<DECIMAL.CREATE
    NAME="decimalobject"
    [LOCALE="localeobject"]/>
```

Parameters

NAME (required)

Input and output parameter. As input, name of the decimal object to create on output.

LOCALE (optional)

Input parameter. The locale object to use for this decimal value. If unspecified, the system default applies.

DECIMAL.GETDECIMAL

Converts a floating point value to a displayable string.

Syntax

```
<DECIMAL.GETDECIMAL
    NAME="decimalobject"
    VALUE="floatvalue"
    VARNAME="decimalvalue"/>
```

Parameters

NAME (required)

Input parameter. Name of the decimal object to convert.

VALUE (required)

Input parameter. Floating-point value to convert.

VARNAME (required)

Input and output parameter. As input, name of a string to hold the formatted decimal value on output.

Description

This tag uses the locale and place count information present in the provided decimal mapping object.

DECIMAL.GETMAXPLACES

Gets the previously set maximum number of places to the right of the decimal point (or comma).

Syntax

```
<DECIMAL.GETMAXPLACES  
    NAME="decimalobject"  
    VARNAME="decimalplaces"/>
```

Parameters

NAME (required)

Input parameter. Name of the decimal object.

VARNAME (required)

Input and output parameter. As input, name of a variable to hold maximum number of decimal places.

DECIMAL.GETMINPLACES

Gets the previously set minimum number of places to the right of the decimal point (or comma).

Syntax

```
<DECIMAL.GETMINPLACES  
    NAME="decimalobject"  
    VARNAME="decimalplaces"/>
```

Parameters

NAME (required)

Input parameter. Name of the decimal object.

VARNAME (required)

Input and output parameter. As input, name of the variable to hold minimum number of decimal places on output.

DECIMAL.GETTHOUSANDS

Gets the value set by [DECIMAL.SETTHOUSANDS](#) for this decimal converter object.

Syntax

```
<DECIMAL.GETTHOUSANDS  
    NAME="decimalobject"  
    VARNAME="true|false"/>
```

Parameters

NAME (required)

Input parameter. Name of the decimal object.

VARNAME (required)

Input and output parameter. As input, name of the variable to hold the true or false value set by [DECIMAL.SETTHOUSANDS](#).

DECIMAL.READDECIMAL

Converts a displayable string (created by [DECIMAL.GETDECIMAL](#)) back into a floating point value.

Syntax

```
<DECIMAL . READDECIMAL
    NAME="decimalobject"
    VALUE="decimalvalue"
    VARNAME="floatvalue"/>
```

Parameters

NAME (required)

Input parameter. Name of the decimal object.

VALUE (required)

Input parameter. The formatted decimal value produced by [DECIMAL.GETDECIMAL](#).

VARNAME (required)

Input and output parameter. As input, name of a variable to hold the floating-point value on output.

DECIMAL.SETMAXPLACES

Sets the maximum number of places to the right of the decimal point (or comma).

Syntax

```
<DECIMAL . SETMAXPLACES
    NAME="decimalobject"
    PLACES="decimalplaces"/>
```

Parameters

NAME (required)

Input parameter. Name of the decimal object.

PLACES (required)

Input parameter. Maximum number of decimal places to set.

DECIMAL.SETMINPLACES

Sets the minimum number of places to the right of the decimal point (or comma).

Syntax

```
<DECIMAL.SETMINPLACES  
    NAME="decimalobject"  
    PLACES="decimalplaces"/>
```

Parameters

NAME (required)

Input parameter. Name of the decimal object.

PLACES (required)

Input parameter. Minimum number of decimal places to set.

DECIMAL.SETTHOUSANDS

Sets whether to include a thousands separator, where appropriate, in displayable strings.

Syntax

```
<DECIMAL.SETTHOUSANDS  
    NAME="decimalobject"  
    VALUE="true|false"/>
```

Parameters

NAME (required)

Input parameter. Name of the decimal object.

VALUE (required)

Input parameter. Either true or false.

Hash Table Tags

Hash table tags help improve CS-Direct Advantage user interface performance. These tags are particularly useful when you are trying quickly and repeatedly to determine whether a list column holds a value.

The following section describes each hash table tag, beginning with [HASH.ADD](#).

HASH.ADD

Adds a value to the hash table object.

Syntax

```
<HASH.ADD
  NAME="objname"
  VALUE="stringval" />
```

Parameters

NAME (required)
Input parameter. Name of the hash table object.

VALUE (required)
Input parameter. Value to add to the hash table.

HASH.CONTAINS

Checks for a specified value in the hash table object.

Syntax

```
<HASH.CONTAINS
  NAME="objname"
  VALUE="stringval"
  VARNAME="Booleanvar" />
```

Parameters

NAME (required)
Input parameter. Name of the hash table object.

VALUE (required)
Input parameter. Value to check for in the hash table.

VARNAME (required)
Input parameter. Variable that holds a string representing boolean status: `true` or `false`, the value is or is not present in the hash table.

HASH.CREATE

Creates a hash table object.

Syntax

```
<HASH.CREATE  
  NAME="objname"  
  [LIST="listname"]  
  [COLUMN="colname"]/>
```

Parameters

NAME (required)

Input parameter. Name of the hash table object.

LIST (optional)

Input parameter. Name of a list from which to read to initialize the hash table. Table is created empty, otherwise.

COLUMN (optional)

Input parameter. Name of column within the list, if specified, from which to read to initialize the hash table.

HASH.HASDATA

Checks for the existence of data in hash table object.

Syntax

```
<HASH.HASDATA  
  NAME="objname"  
  VARNAME="varname">
```

Parameters

NAME (required)

Input parameter. Name of the hash table object.

VARNAME (required)

Input and output parameter. As input, name of the variable that holds a boolean value on output: `true` if the hash table object has data; `false` if it is empty.

HASH.TOSTRING

Writes out hash table values to a string.

Syntax

```
<HASH.TOSTRING  
  NAME="objname"  
  DELIM="delimiter"  
  VARNAME="varname" />
```

Parameters

NAME (required)

Input parameter. Name of the hash table object.

DELIM (required)

Input parameter. Delimiter used to separate hash values in the output string.

VARNAME (required)

Input and output parameter. As input, name of the variable to which the delimited values are assigned on output.

List Object Tags

You use the list object tags to construct Content Server resultset lists, which are used throughout your elements as arguments for other CS-Direct Advantage tags. For example, you call list object tags and use the subsequent resultset lists as input for shopping cart tags, and assetset tags. You use all three list object tags at a time to perform the following tasks:

- Create an empty list object.
- Add rows to the list object.
- Convert the list object to a Content Server resultset list.

The following section describes each list object tag, beginning with [LISTOBJECT.ADDROW](#).

LISTOBJECT.ADDROW

Adds a new row to the end of a list.

Syntax

```
<LISTOBJECT.ADDROW
  NAME="listname"
  colx="valx" ... colz="valz"/>
```

Parameters

NAME (required)

Input parameter. Name of the list object to which to add a new row.

colx="valx" ... colz="valz" (required)

Input parameter. The remaining arguments are column name/value pairs that constitute the row.

Note

The list object name and value inputs are case-sensitive.

Example

```
<!-- Create a list for a call to ASSETSET.GETASSETLIST -->
<LISTOBJECT.CREATE NAME="mylist"
  COLUMNS="attributename,direction"/>
<LISTOBJECT.ADDROW NAME="mylist" attributename="name"
  direction="descending"/>
<LISTOBJECT.ADDROW NAME="mylist" attributename="price"
  direction="descending"/>
<LISTOBJECT.TOLIST NAME="mylist" LISTVARNAME="listout"/>
```

LISTOBJECT.CREATE

Creates a new list with specific columns but no rows.

Syntax

```
<LISTOBJECT.CREATE
    NAME="listname"
    COLUMNS="col1,col2,...,coln"/>
```

Parameters

NAME (required)

Input and output parameter. As input, name of the list object to create on output as an empty list with the specified columns.

COLUMNS (required)

Input parameter. A comma-separated string of column names.

Example

```
<!-- Create a list for a call to ASSETSET.GETASSETLIST -->
<LISTOBJECT.CREATE NAME="mylist"
COLUMNS="attributename,direction"/>
<LISTOBJECT.ADDROW NAME="mylist" attributename="Name"
    direction="descending"/>
<LISTOBJECT.ADDROW NAME="mylist" attributename="Price"
    direction="descending"/>
<LISTOBJECT.TOLIST NAME="mylist" LISTVARNAME="listout"/>
```

LISTOBJECT.TOLIST

Converts a list object to a resultset list.

Syntax

```
<LISTOBJECT.TOLIST
    NAME="listname"
    LISTVARNAME="varname"/>
```

Parameters

NAME (required)

Input parameter. Name of the list object to convert.

LISTVARNAME (required)

Input and output parameter. As input, name of the list variable into which to write the resultset list on output.

Example

```
<!-- Create a list for a call to ASSETSET.GETASSETLIST -->
<LISTOBJECT.CREATE NAME="mylist"
COLUMNS="attributename,direction"/>
<LISTOBJECT.ADDROW NAME="mylist" attributename="name"
```

```
    direction="descending"/>
<LISTOBJECT.ADDROW NAME="mylist" attributename="price"
    direction="descending"/>
<LISTOBJECT.TOLIST NAME="mylist" LISTVARNAME="listout"/>
```


Locale Tags

The following section describes the locale tag, [LOCALE.CREATE](#).

LOCALE.CREATE

Creates a locale descriptor for use in other tags.

Syntax

```
<LOCALE.CREATE
  VARNAME="localeobject"
  COUNTRY="isocountry"
  LANGUAGE="isolanguage"
  LOCALENAME="locale name"/>
```

Parameters

VARNAME (required)

Input and output parameter. As input, name of a variable to which to assign the locale object on output.

COUNTRY (required)

Input parameter. ISO-standard country name.

LANGUAGE (required)

Input parameter. ISO-standard language name.

LOCALENAME (required)

Input parameter. Name of the locale, for example, en_US for the United States, ja_JP for Japan, or fr_FR for France.

Description

You use the LOCALE.CREATE tag to generate a locale object, which is used to describe the desired locale for various other tags in the system.

Miscellaneous Tags

The following section describes [MISC.GETLISTCOLUMNNNAMES](#).

MISC.GETLISTCOLUMNNNAMES

Returns the names of all the columns in an input list.

Syntax

```
<MISC.GETLISTCOLUMNNNAMES  
  LIST="listobject"  
  LISTVARNAME="varname" />
```

Parameters

LIST (required)

Input parameter. Name of the list object to interrogate.

LISTVARNAME (required)

Input and output parameter. As input, variable list name to write with the list of column names. The column is called name.

Searchstate Tags

A searchstate represents a set of constraints that apply to a pool of products. A **constraint** can be either a restriction based on a single attribute, or another searchstate (called a **nested constraint**). Searchstate objects contain:

- a list of attribute names and their values
- a list of index names and their rich-text search criteria.

The following section describes the searchstate tags, beginning with [ASSETSET.GETASSETCOUNT](#).

SEARCHSTATE.ADDLIKECONSTRAINT

Adds an attribute name/value constraint into a new or existing searchstate object.

Syntax

```
<SEARCHSTATE.ADDLIKECONSTRAINT
  NAME="ssname"
  [BUCKET="bucket name"]
  [TYPENAME="asset type"]
  ATTRIBUTE="attribname"
  [LIST="list name"]
  [IMMEDIATEONLY="true|false"]
  [CASEINSENSITIVE="true|false"]/>
```

Parameters

NAME (required)

Input parameter. Name of an object representing a searchstate.

BUCKET (optional)

Input parameter. Bucket name. If not specified, the attribute name is used. Use this parameter to indicate which constraint you need when the same attribute is used in multiple constraints. A bucket is an alias and functions similar to an inner join in SQL; when you have multiple tables that have columns of the same name, you must specify the table name to indicate which column you are requesting.

TYPENAME (optional)

Input parameter. The internal asset name for the attribute (either CAttributes for content attribute, or PAttributes for product attribute). If you do not specify TYPENAME, a value is supplied from a property in the `gator.ini` property file: `mwb.defaultattributes=PAtributes`. The default is PAttributes and the value may be changed.

ATTRIBUTE (required)

Input parameter. Name of the attribute to constrain.

LIST (optional)

Input parameter. A list of the constrained values for the attribute. If specified, one or more of the values must match the attribute for a product to meet the constraint. The default is that all assets that have any value for the attribute match the constraint. The column is called value.

IMMEDIATEONLY (optional)

Input parameter. A Boolean value: `true` indicates that the search is limited to values directly associated with the specified attribute; `false` (the default) extends the search to include values inherited from a parent.

CASEINSENSITIVE (optional)

Input parameter. A Boolean value: `true` indicates that the comparison is case-insensitive; `false` (the default) considers case in the comparison.

Description

The object referred to by `NAME` is updated to include the new constraint, and is similar to a database `LIKE` operation and accepts wild cards `"%"`. If the attribute name is already in the searchstate, then the new constraint replaces the old constraint.

SEARCHSTATE.ADDNESTEDCONSTRAINT

Adds a nested searchstate as part of the constraint.

Syntax

```
<SEARCHSTATE.ADDNESTEDCONSTRAINT
  NAME="newssname"
  BUCKET="bucketname"
  SEARCHSTATE="ssname"/>
```

Parameters**NAME (required)**

Input parameter. Name of the searchstate object to add.

BUCKET(required)

Input parameter. An invented name defined by [SEARCHSTATE.FINDCONSTRAINT](#). Use this parameter to indicate which constraint you need when the same attribute is used in multiple constraints. A bucket is an alias and functions similar to an inner join in SQL; when you have multiple tables that have columns of the same name, you must specify the table name to indicate which column you are requesting.

SEARCHSTATE (required)

Input parameter. Name of the searchstate object to nest inside of the object specified by `NAME`.

SEARCHSTATE.ADDRANGECONSTRAINT

Adds a range constraint for a specific attribute name.

Syntax

```
<SEARCHSTATE.ADDRANGECONSTRAINT
  NAME="ssname"
  [BUCKET="bucketname"]
  [TYPENAME="assettype"]
  ATTRIBUTE="attribname"
  LOWER="lowrange"
  UPPER="uprange"
  [CASEINSENSITIVE="true|false"] />
```

Parameters

NAME (required)

Input parameter. Name of an object representing a searchstate.

BUCKET (optional)

Input parameter. Bucket name. If not specified, the attribute name is used. Use this parameter to indicate which constraint you need when the same attribute is used in multiple constraints. A bucket is an alias and functions similar to an inner join in SQL; when you have multiple tables that have columns of the same name, you must specify the table name to indicate which column you are requesting.

TYPENAME (optional)

Input parameter. The internal asset name for the attribute (either CAttributes for content attribute, or PAttributes for product attribute). If you do not specify TYPENAME, a value is supplied from a property in the gator.ini property file: mwb.defaultattributes=PAtributes. The default is PAttributes and the value may be changed.

ATTRIBUTE (required)

Input parameter. Name of the attribute to constrain.

LOWER | LOWEREQUAL (required)

Input parameter. The bottom end of the range.

UPPER | UPPEREQUAL (required)

Input parameter. The top end of the range.

CASEINSENSITIVE (optional)

Input parameter. A Boolean value: true indicates that the comparison is case-insensitive; false (the default) considers case in the comparison.

Example

```
<!-- if there is a rich text search parameter,
  use that instead of any parametric search parameters -->
<IF COND="IsVariable.gdSearch=true"><THEN>
  <SEARCHSTATE.ADDRICHTEXTCONSTRAINT NAME="mysearch"
    ATTRIBUTE="Name"
    VALUE="Variables.gdSearch" CONFIDENCE=".1"/>
</THEN><ELSE>
  <SEARCHSTATE.ADDSIMPLESTANDARDCONSTRAINT NAME="mysearch"
    ATTRIBUTE="Variables.p1" VALUE="Variables.p2"/>
```

```

<SEARCHSTATE.ADDRANGECONSTRAINT NAME="mysearch"
  ATTRIBUTE="Price"
  LOWER="Variables.low" UPPER="Variables.high"/>
</ELSE></IF>

```

SEARCHSTATE.ADDRICHTEXTCONSTRAINT

Adds an index name and rich-text expression to the list of rich-text criteria for items.

Syntax

```

<SEARCHSTATE.ADDRICHTEXTCONSTRAINT
  NAME="ssname"
  [BUCKET="bucketname"]
  [TYPENAME="assettype"]
  ATTRIBUTE="attribname"
  VALUE="criteria"
  [PARSER="parsername"]
  CONFIDENCE="minlevel"
  [MAXCOUNT="number"] />

```

Parameters

NAME (required)

Input parameter. Name of an object representing a searchstate.

BUCKET (optional)

Input parameter. Bucket name. If not specified, the attribute name is used. Use this parameter to indicate which constraint you need when the same attribute is used in multiple constraints. A bucket is an alias and functions similar to an inner join in SQL; when you have multiple tables that have columns of the same name, you must specify the table name to indicate which column you are requesting.

TYPENAME (optional)

Input parameter. The internal asset name for the attribute (either CAttributes for content attribute, or PAttributes for product attribute). If you do not specify TYPENAME, a value is supplied from a property in the `gator.ini` property file: `mwb.defaultattributes=PAtributes`. The default is PAttributes and the value may be changed.

ATTRIBUTE (required)

Input parameter. Name of the attribute to constrain.

VALUE (required)

Input parameter. The rich-text search criteria, which should apply to the attribute.

PARSER (optional)

Input parameter. The search-engine-dependent rich text parser to use.

CONFIDENCE (required)

Input parameter. The minimum confidence level for the match. This parameter is search engine dependent; adjust the value lower if you are not getting the desired results.

MAXCOUNT (optional)

Input parameter. The maximum number of answers desired for the match. If this parameter is not specified, the number of results is limited only by the confidence and the number of products.

Description

The object described by the NAME parameter is updated to reflect the additional search criteria. If the attribute name is already mentioned as part of a rich-text constraint in the searchstate, then the existing constraint is removed first.

Note

This tag requires that the attribute was created as indexed.

Example

```
<!-- if there is a rich text search parameter,
      use that instead of any parametric search parameters -->
<IF COND="IsVariable.gdSearch=true"><THEN>
  <SEARCHSTATE.ADDRICHTEXTCONSTRAINT NAME="mysearch"
    ATTRIBUTE="Name"
    VALUE="Variables.gdSearch" CONFIDENCE=".1"/>
</THEN><ELSE>
  <SEARCHSTATE.ADDSIMPLESTANDARDCONSTRAINT NAME="mysearch"
    ATTRIBUTE="Variables.p1" VALUE="Variables.p2"/>
  <SEARCHSTATE.ADDRANGECONSTRAINT NAME="mysearch"
    ATTRIBUTE="Price"
    LOWER="Variables.low" UPPER="Variables.high"/>
</ELSE></IF>
```

SEARCHSTATE.ADDSIMPLELIKECONSTRAINT

Adds an attribute name/single value constraint to an existing searchstate.

Syntax

```
<SEARCHSTATE.ADDSIMPLELIKECONSTRAINT
  NAME="ssname"
  [BUCKET="bucketname"]
  [TYPENAME="assettype"]
  ATTRIBUTE="attribname"
  VALUE="value"
  [IMMEDIATEONLY="true|false"]/>
```

Parameters

NAME (required)

Input parameter. Name of an object representing a searchstate.

BUCKET (optional)

Input parameter. Bucket name. If not specified, the attribute name is used. Use this parameter to indicate which constraint you need when the same attribute is used in multiple constraints. A bucket is an alias and functions similar to an inner join in SQL; when you have multiple tables that have columns of the same name, you must specify the table name to indicate which column you are requesting.

TYPENAME (optional)

Input parameter. The internal asset name for the attribute (either CAttributes for content attribute, or PAttributes for product attribute). If you do not specify TYPENAME, a value is supplied from a property in the gator.ini property file: mwb.defaultattributes=PAtributes. The default is PAtributes and the value may be changed.

ATTRIBUTE (required)

Input parameter. Name of the attribute to constrain.

VALUE (required)

Input parameter. The single value that constrains the attribute.

IMMEDIATEONLY (optional)

Input parameter. A boolean: true indicates that the search is limited to values directly associated with the specified attribute; false (the default) extends the search to include values inherited from a parent.

Description

This tag is the simple version of [SEARCHSTATE.ADDLIKECONSTRAINT](#). This tag is similar to a database LIKE operation and accepts wild cards (%). The object referred to by NAME is updated to reflect the new constraint. If the attribute name is already in the searchstate, then the new constraint replaces the old constraint.

SEARCHSTATE.ADDSIMPLESTANDARDCONSTRAINT

Adds an attribute name/single value constraint to an existing searchstate.

Syntax

```
<SEARCHSTATE.ADDSIMPLESTANDARDCONSTRAINT
  NAME="ssname"
  [BUCKET="bucketname"]
  [TYPENAME="assettype"]
  ATTRIBUTE="attribname"
  VALUE="value"
  [IMMEDIATEONLY="true|false"]/>
```

Parameters

NAME (required)

Input parameter. Name of an object representing a searchstate.

BUCKET (optional)

Input parameter. Bucket name. If not specified, the attribute name is used. Use this parameter to indicate which constraint you need when the same attribute is used in multiple constraints. A bucket is an alias and functions similar to an inner join in SQL; when you have multiple tables that have columns of the same name, you must specify the table name to indicate which column you are requesting.

TYPENAME (optional)

Input parameter. The internal asset name for the attribute (either CAttributes for content attribute, or PAttributes for product attribute). If you do not specify TYPENAME, a value is supplied from a property in the gator.ini property file: mwb.defaultattributes=PAtributes. The default is PAtributes and the value may be changed.

ATTRIBUTE (required)

Input parameter. Name of the attribute to constrain.

VALUE (required)

Input parameter. The single value that constrains the attribute.

IMMEDIATEONLY (optional)

Input parameter. A Boolean value: `true` indicates that the search is limited to values directly associated with the specified attribute; `false` (the default) extends the search to include values inherited from a parent.

Description

This tag is the simple version of [SEARCHSTATE.ADDSTANDARDCONSTRAINT](#). The object referred to by `NAME` is updated to reflect the new constraint. If the attribute name is already in the searchstate, then the new constraint replaces the old constraint.

Example

```
<!-- if there is a rich text search parameter,
      use that instead of any parametric search parameters -->
<IF COND="IsVariable.gdSearch=true"><THEN>
  <SEARCHSTATE.ADDRICHTEXTCONSTRAINT NAME="mysearch"
    ATTRIBUTE="Name"
    VALUE="Variables.gdSearch" CONFIDENCE=".1"/>
</THEN><ELSE>
  <SEARCHSTATE.ADDSIMPLESTANDARDCONSTRAINT NAME="mysearch"
    ATTRIBUTE="Variables.p1" VALUE="Variables.p2"/>
  <SEARCHSTATE.ADDRANGECONSTRAINT NAME="mysearch"
    ATTRIBUTE="Price" '
    LOWER="Variables.low" UPPER="Variables.high"/>
</ELSE></IF>
```

SEARCHSTATE.ADDSTANDARDCONSTRAINT

Adds an attribute name/value constraint into a new or existing searchstate object.

Syntax

```
<SEARCHSTATE.ADDSTANDARDCONSTRAINT
  NAME="ssname"
  [BUCKET="bucketname"]
  [TYPENAME="assettype"]
  ATTRIBUTE="attribname"
  [LIST="listname"]
  [IMMEDIATEONLY="true|false"]
  [CASEINSENSITIVE="true|false"]/>
```

Parameters

NAME (required)

Input parameter. Name of an object representing a searchstate.

BUCKET (optional)

Input parameter. Bucket name. If not specified, the attribute name is used. Use this parameter to indicate which constraint you need when the same attribute is used in multiple constraints. A bucket is an alias and functions similar to an inner join in SQL; when you have multiple tables that have columns of the same name, you must specify the table name to indicate which column you are requesting.

TYPENAME (optional)

Input parameter. The internal asset name for the attribute (either `CAttributes` for content attribute, or `PAttributes` for product attribute). If you do not specify `TYPENAME`, a value is supplied from a property in the `gator.ini` property file: `mwb.defaultattributes=PAttributes`. The default is `PAttributes` and the value may be changed.

ATTRIBUTE (required)

Input parameter. Name of the attribute to constrain.

LIST (optional)

Input parameter. A list of the constrained values for the attribute. If specified, one or more of the values must match the attribute for a product to meet the constraint. The default is that all assets that have any value for the attribute match the constraint.

IMMEDIATEONLY (optional)

Input parameter. A Boolean value: `true` indicates that the search is limited to values directly associated with the specified attribute; `false` (the default) extends the search to include values inherited from a parent.

CASEINSENSITIVE (optional)

Input parameter. A Boolean value: `true` indicates that the comparison is case-insensitive; `false` (the default) considers case in the comparison.

Description

The object referred to by `NAME` is updated to include the new constraint. If the attribute name is already in the searchstate, then the new constraint replaces the old constraint.

SEARCHSTATE.CREATE

Builds an empty searchstate object.

Syntax

```
<SEARCHSTATE.CREATE
  NAME="ssname"
  [OP="and|or"]/>
```

Parameters

NAME (required)

Input and output parameter. As input, name of the searchstate object to create on output.

OP (optional)

Input parameter. Searchstate type, either `and` or `or`. The default is `and`.

Example

```
<!-- Iterate through a list of attribute values. -->
<!-- First, create a blank searchstate. -->
<SEARCHSTATE.CREATE NAME="ss"/>
<!-- Associate the blank searchstate with a new asset set. -->
<ASSETSET.SETSEARCHEDASSETS NAME="as" CONSTRAINT="ss"/>
<!-- Get a set of all values for the given attribute. -->
<ASSETSET.GETATTRIBUTEVALUES ORDERING="descending" NAME="as"
  ATTRIBUTE="Price" LISTVARNAME="results"/>
```

SEARCHSTATE.DELETECONSTRAINT

Deletes a specific attribute constraint from a searchstate.

Syntax

```
<SEARCHSTATE.DELETECONSTRAINT
  NAME="ssname"
  [BUCKET="bucketname"]
  [TYPENAME="asset type"]
  ATTRIBUTE="attribname"/>
```

Parameters

NAME (required)

Input parameter. Name of an object representing a searchstate.

BUCKET (optional)

Input parameter. Bucket name. If not specified, the attribute name is used. Use this parameter to indicate which constraint you need when the same attribute is used in multiple constraints. A bucket is an alias and functions similar to an inner join in SQL; when you have multiple tables that have columns of the same name, you must specify the table name to indicate which column you are requesting.

TYPENAME (optional)

Input parameter. The internal asset name for the attribute (either CAttributes for content attribute, or PAttributes for product attribute). If you do not specify TYPENAME, a value is supplied from a property in the gator.ini property file: mwb.defaultattributes=PAtributes. The default is PAttributes and the value may be changed.

ATTRIBUTE (required)

Input parameter. Name of the attribute whose constraint should be deleted.

SEARCHSTATE.DUPLICATE

Duplicates a searchstate object.

Syntax

```
<SEARCHSTATE.DUPLICATE  
  NAME="ssname"  
  VARNAME="varname" />
```

Parameters

NAME (required)

Input parameter. Name of the searchstate object.

VARNAME (required)

Input and output parameter. As input, name of the new searchstate object to hold the duplicate on output.

SEARCHSTATE.FINDCONSTRAINT

Checks a searchstate for a constraint to a specific attribute.

Syntax

```
<SEARCHSTATE.FINDCONSTRAINT  
  NAME="ssname"  
  [BUCKET="bucketname"]  
  ATTRIBUTE="attribname"  
  VARNAME="varname" />
```

Parameters

NAME (required)

Input parameter. Name of an object representing a searchstate.

BUCKET (optional)

Input parameter. Bucket name. If not specified, the attribute name is used. Use this parameter to indicate which constraint you need when the same attribute is used in multiple constraints. A bucket is an alias and functions similar to an inner join in SQL; when you have multiple tables that have columns of the same name, you must specify the table name to indicate which column you are requesting.

ATTRIBUTE (required)

Input parameter. Name of the attribute to constrain.

VARNAME (required)

Input and output parameter. As input, name of a variable into which to write the constraint type on output. Note the variants below.

OBJVARNAME (required)

Name of an object to create with the contents of the nested constraint.

ATTRTYPEVARNAME (optional)

Name of a variable into which to write the attribute asset type name for the constraint.

ATTRVARNAME (optional)

Name of a variable into which to write the attribute name for the constraint.

LISTVARNAME (required)

Name of a variable into which to write the list of values for the attribute, if found. If not found, the variable is set to null. The list has a single column named value.

IMMEDIATEONLYVARNAME (optional)

Name of a variable into which to write the IMMEDIATEONLY constraint value.

CASEINSENSITIVEVARNAME (optional)

Name of a variable into which to write the CASEINSENSITIVE constraint value.

ATTRTYPEVARNAME (optional)

Name of a variable into which to write the attribute asset type name for the constraint.

ATTRVARNAME (optional)

Name of a variable into which to write the attribute name for the constraint.

EXPRESSIONVARNAME (optional)

Name of the variable into which to write the rich text expression.

PARSERVARNAME (optional)

Name of the variable into which to write the rich text parser name.

CONFIDENCEVARNAME (optional)

Name of the variable into which to write the confidence.

MAXCOUNTVARNAME (optional)

Name of the variable into which to write the max answers.

ATTRTYPEVARNAME (optional)

Name of a variable into which to write the attribute asset type name for the constraint.

ATTRVARNAME (optional)

Name of a variable into which to write the attribute name for the constraint.

LOWERVERNAME | LOWEREQUALVARNAME (required)

Name of the variable into which to write the lower constraint.

UPPERVARNAME | UPPEREQUALVARNAME (required)

Name of the variable into which to write the upper constraint.

IMMEDIATEONLYVARNAME (optional)

Name of the variable into which to write the IMMEDIATEONLY value constraint.

CASEINSENSITIVEVARNAME (optional)

Name of a variable into which to write the CASEINSENSITIVE constraint value.

Description

If found, data values about the constraint are returned.

SEARCHSTATE.FROMSTRING

Provides the ability for a searchstate object to be initialized from its string representation.

Syntax

```
<SEARCHSTATE.FROMSTRING  
  NAME="objname"  
  VALUE="stringval" />
```

Parameters

NAME (required)

Input and output parameter. As input, name of the output object to be reinitialized with the string passed in the VALUE argument.

VALUE (required)

Input parameter. The string value to be used to initialize the searchstate object.

SEARCHSTATE.TOSTRING

Converts a searchstate object into its string representation that is suitable for various uses, such as saving in a session variable or packing into a URL.

Syntax

```
<SEARCHSTATE.TOSTRING  
  NAME="objname"  
  VARNAME="varname" />
```

Parameters

NAME (required)

Input parameter. Name of the object.

VARNAME (required)

Input and output parameter. As input, name of the string that is assigned the string representation on output.

Session Tag

The following section describes the session objects tag, [SESSIONOBJECTS.FLUSH](#).

SESSIONOBJECTS.FLUSH

Flushes object storage for all expired sessions.

Syntax

```
<SESSIONOBJECTS.FLUSH  
  [TIMESTAMP="cutofftime"]/>
```

Parameters

TIMESTAMP (optional)

Input parameter. The cutoff timestamp for objects. All objects older than the timestamp are removed.

String tags

These tags perform various string manipulations. The following section describes the string tags, beginning with [STRING.FROMOBJECT](#).

STRING.FROMOBJECT

Converts a named Content Server object back to a string.

Syntax

```
<STRING.FROMOBJECT  
    VALUE="inputstring"  
    VARNAME="varname"/>
```

Parameters

VALUE (required)

Input parameter. Name of the Content Server object.

VARNAME (required)

Input and output parameter. As input, name of the variable to hold the string on output.

STRING.LENGTH

Returns the length of an input string in the specified variable.

Syntax

```
<STRING.LENGTH  
    VALUE="inputstring"  
    VARNAME="varname"/>
```

Parameters

VALUE (required)

Input parameter. The string to size.

VARNAME (required)

Input and output parameter. As input, the name of variable to hold the length of the string on output.

STRING.STREAM

Streams text in an HTML context.

Syntax

```
<STRING.STREAM  
    VALUE="inputstring"  
    [VARIABLE="variable name"]  
    [SSVARIABLE="session variable name"]  
    [PROPERTY="property name"]  
    [LIST="list name"]  
    [COLUMN="column name"] />
```

Parameters

VALUE (required)

Input parameter. The string to stream.

VARIABLE (optional)

Name of the Content Server variable.

SSVARIABLE (optional)

Name of the Content Server session variable.

PROPERTY (optional)

Name of the property from the `futuretense.ini` property file.

LIST (optional)

Name of the list.

COLUMN (optional)

Name of the field in the list. You must use COLUMN when you use LIST.

Description

Use this tag to display text literally as stored in Content Server database, that is, with no browser interpretation or Content Server evaluation of variable names that are present in the text. This feature is useful for Edit or Inspect forms. This tag streams character text after HTML-escaping special characters, as well as all characters greater than 127. Any strings that are special to Content Server evaluation engine (for example, `Variable.id`) are not evaluated.

If you want to retrieve a blob, use `BlobServer`. If you want to render a binary document, use [RENDER.STREAM](#).

STRING.TOOBJECT

Converts string to a named Content Server object.

Syntax

```
<STRING.TOOBJECT  
    VALUE="inputstring"  
    VARNAME="varname" />
```

Parameters

VALUE (required)

Input parameter. The string to convert.

VARNAME (required)

Input and output parameter. As input, name of the object to create on output.

Text Format Tags

The following section describes each text format tag, beginning with [TEXTFORMAT.LENGTH \(Deprecated\)](#).

TEXTFORMAT.LENGTH (Deprecated)

As of Content Server version 4.0, use [STRING.LENGTH](#).

Returns the length of an input string in the specified variable.

Syntax

```
<TEXTFORMAT.LENGTH
    VALUE="inputstring"
    VARNAME="varname"/>
```

Parameters

VALUE (required)

Input parameter. The string to size.

VARNAME (required)

Input and output parameter. As input, the name of variable to hold the length of the string on output.

TEXTFORMAT.TOSTRING

Converts a list to a displayable string.

Syntax

```
<TEXTFORMAT.TOSTRING
    LIST="listname"
    [FIELD="fieldname"]
    VARNAME="varname"
    [MAXCOUNT="number"]
    [ELLIPSIS="true|false"]/>
```

Parameters

LIST (required)

Input parameter. Name of the list to convert.

FIELD (optional)

Input parameter. A column name in the list.

VARNAME (required)

Input and output parameter. As input, the name of variable to hold the string on output.

MAXCOUNT (optional)

Input parameter. Maximum number of entries to display.

ELLIPSIS (optional)

Input parameter. determines if an ellipsis string (...) is appended if more than the maximum number of entries exist in this list. Valid options are `true` (default value) and `false`.

URL Tags (Deprecated)

*As of Content Server version 4.0, replaced by **RENDER** tags.*

URL methods provide basic packing and unpacking for URLs.

URL.PACK (Deprecated)

*As of Content Server version 4.0, replaced by **RENDER** tags.*

Converts a string into a form suitable for passage through a URL.

Syntax

```
<URL.PACK
    VALUE="inputstring"
    VARNAME="varname"/>
```

Parameters

VALUE (required)

Input parameter. The input string.

VARNAME (required)

Input and output parameter. As input, name of the variable to hold the URL-encoded string on output.

Description

The string held by the output variable is a packed string that is compatible with both iAS URL and WebLogic URL conventions. No unpacking is necessary if the packed string is used as part of a URL because the application server does it automatically.

URL.UNPACK (Deprecated)

*As of Content Server version 4.0, replaced by **RENDER** tags.*

Converts a string created with [URL.PACK \(Deprecated\)](#) back to an unpacked form.

Syntax

```
<URL.UNPACK
    VALUE="inputstring"
    VARNAME="varname"/>
```

Parameters

VALUE (required)

Input parameter. The packed input string.

VARNAME (required)

Input and output parameter. As input, name of the variable to hold the unpacked string on output.

Description

You use this tag for form elements that have been packed because they contain questionable XML characters. You do not need to explicitly unpack URL values because the application server does it automatically.

Visitor Data Manager Tags

The visitor data manager is the data access method that CS-Engage uses when processing rules. The visitor data manager enables you to gather, store, and retrieve visitor information from both the local, Content Server database as well as various external databases — Transact, for example.

When you create visitor attributes, history attributes, and history types for defining segments, the visitor data manager stores them in the local CS-Engage database. If you are using Transact, you can select Transact visitor attributes when defining segments as well. Then, when a method that determines which segments a visitor qualifies for is invoked, the visitor data manager checks the appropriate database for the needed visitor data. It provides a single access method to all of your visitor data. For information about creating visitor attributes, history attributes, and history types, see the *CSEE Developer's Guide*.

To maintain visitor session information after visitors end their sessions, you use persistent cookies. These cookies serve two purposes: to store the data and to mark the session with an identifier.

A visitor's Transact ID is also an identifier that can mark a visitor session. If a visitor performs an action on the site that requires him to log into Transact, his Transact ID is passed to his hard drive and identifies him as a valid user.

The object methods in the preceding list enable you to link visitor sessions marked by different identifiers by assigning the same alias to those sessions.

Use the [VDM.RECORDHISTORY](#) and [VDM.SETSCALAR](#) tags to write visitor data to the CS-Engage database. Use the [VDM.RECORDHISTORY](#) tag to write a time-stamped record of the data defined in a history type to the database. This data could include information from several history attributes, depending on the history type. Use the [VDM.SETSCALAR](#) tag to write data for individual visitor attributes to the database.

Use the GET tags to retrieve visitor data from either the local or external databases. (Whenever one of the COMMERCECONTEXT tags invokes a SEGMENT tag, they retrieve this data automatically — you do not have to use the VDM.GET tags in those cases.)

Note

Because the visitor data manager does not track revisions, you cannot create segments based on changes in visitor data. For example, two sessions ago a visitor changed his job description from “technical” to “financial.” When that change was written to the database, the value “financial” replaced the value “technical” and there is no record of his previous job description. Now, when segments based on job description are calculated, he qualifies only for the ones that include “financial” as an option.

The following section describes the visitor data manager tags, beginning with [VDM.FLUSHINACTIVE](#).

VDM.FLUSHINACTIVE

Removes unused data recorded for visitors that have not returned to the site within a specified period of time.

Syntax

```
<VDM.FLUSHINACTIVE  
  STARTDATE="timestamp" />
```

Parameters

STARTDATE (required)

Input parameter. Cutoff timestamp in milliseconds since Jan 1, 1970.

Description

This tag removes all data accumulated for visitors who have not been to the site since the specified cutoff date. All visitors who have been to the site since the cutoff date are considered current, and their data is not removed.

Note

This tag is for maintenance purposes only, and should not be used as part of site delivery. It can take a significant time to run.

VDM.GETACCESSID

Retrieves a copy of the visitor's access identifier, if there is one.

Syntax

```
<VDM.GETACCESSID  
  KEY="pluginname"  
  VARNAME="varname" />
```

Parameters

KEY (required)

Input parameter. Name of the IConnection plug-in. There is one valid key name: tx for Transact.

VARNAME (required)

Input and output parameter. As input, name of a variable to create on output to hold the access identifier.

Description

Use this tag when you have an IConnection plug-in for Commerce Connector that is not for Transact.

The visitor data manager passes this access identifier to Commerce Connector in order to obtain visitor data from an external database. Having an access identifier is equivalent to

being logged into that external database. If the visitor does not have an access ID, there is no external visitor data available for this person.

Note

Setting this tag to `<VDM.GETACCESSID KEY="tx"/>` is the equivalent of using `<VDM.GETCOMMERCEID/>`

VDM.GETALIAS

Retrieves the alias string for the current visitor, if there is one.

Syntax

```
<VDM.GETALIAS
    KEY="keyvalue"
    VARNAME="varname"/>
```

Parameters

KEY (required)

Input parameter. Identifying key value for the current visitor (for example, a site cookie).

VARNAME (required)

Input and output parameter. As input, name of the variable to assign the value of the alias on output.

Description

You use this tag for debugging during the development phase.

VDM.GETCOMMERCEID

Retrieves a copy of the visitor's commerce identifier, that is, the visitor's Transact ID.

Syntax

```
<VDM.GETCOMMERCEID
    VARNAME="varname"/>
```

Parameters

VARNAME (required)

Input and output parameter. As input, name of a variable to which the commerce identifier is assigned on output.

Description

The visitor data manager passes retrieved identifier to Commerce Connector in order to obtain Transact-based visitor data. Therefore, having a commerce identifier is equivalent to being logged into Transact. If the visitor does not have a commerce identifier, there is no Transact-based visitor data available for this person.

Note

Using the <VDM.GETCOMMERCEID/> tag is the equivalent of setting <VDM.GETACCESSID KEY= "tx"/>.

VDM.SETACCESSID

Sets or clears the visitor's access identifier.

Syntax

```
<VDM.SETACCESSID  
    KEY="pluginname"  
    [ID="userid"]/>
```

Parameters

KEY (required)

Input parameter. Name of the IConnection plug-in. There is one valid key name: tx for Transact.

ID (optional)

Input parameter. The visitor's commerce identifier, user ID. Omit the ID to clear the commerce identifier for the current visitor.

Description

Having an access ID set for a visitor is the equivalent of logging into an external database (other than Transact's). Use this tag when you need to check whether or not the visitor is logged in to an external database.

Additionally, the visitor data manager passes this ID to Commerce Connector in order to obtain visitor data from an external database. If the visitor does not have an access identifier, there is no external visitor data available for this person.

Note

Setting this tag to <VDM.SETACCESSID KEY="tx"/> is the equivalent of using <VDM.SETCOMMERCEID/>.

VDM.SETALIAS

Sets the alias string for the current visitor using the specified alias key.

Syntax

```
<VDM.SETALIAS  
    KEY="keyvalue"  
    VALUE="aliasvalue"/>
```

Parameters

KEY (required)

Input parameter. Kind of alias, that is, an arbitrary string that describes the kind of link being set up (for example, a cookie).

VALUE (required)

Input parameter. Value of the alias.

Description

VDM.SETALIAS associates an alias with the current session. You use this tag to link sessions together so that session data from multiple sessions can be linked to the same (current) visitor.

For more information about working with aliases, see the *CSEE Developer's Guide*.

VDM.SETCOMMERCEID

Sets or clears the visitor's commerce identifier; that is, the visitor's Transact ID.

Syntax

```
<VDM.SETCOMMERCEID  
    [ID="userid"]/>
```

Parameters

ID (optional)

Input parameter. The visitor's commerce identifier, user ID. Omit the ID to clear the commerce identifier for the current visitor.

Description

Having a commerce ID set for a visitor is the equivalent of logging into Transact. Use this tag when you need to check whether or not the visitor is logged in to Transact.

Additionally, the visitor data manager passes this ID to Commerce Connector in order to obtain Transact-based visitor data. If the visitor does not have a commerce identifier, there is no Transact-based visitor data available for this person.

For information about additional tags and parameters that are typically used with this one, see the descriptions of [CSUSER.AUTHENTICATE](#) and the user ID buyer parameter.

Note

Using the `<VDM.SETCOMMERCEID/>` tag is the equivalent of using `<VDM.SETACCESSID ID= "tx"/>`.

Chapter 4

CS-Engage XML Tags

This chapter describes the functionality and XML syntax of the CS-Engage object methods. These methods are implemented as Content Server custom application tags, which invoke application business logic for applications built with Content Server.

This chapter serves as your reference for the tags that you use to design application pages that gather visitor data and display recommendations or promotions (merchandising information) based on that data. Use the CS-Direct Advantage object methods to design the catalog pages. Then use the CS-Engage object methods to display merchandising information on those pages, information that changes based on both the identity of the visitor, and, in some cases, the product or content that is currently displayed.

This chapter contains the following sections:

- [Assetset Tags](#)
- [Commerce Context tags](#)
- [Visitor Data Manager Tags](#)

Note

Although this chapter contains some examples, please refer to the GE Lighting sample catalog where you can search the sample code for examples of nearly every CS-Engage tag used in a consistent context.

Assetset Tags

Assetset tags represent a group of assets that can be described either by a search constraint or by a list. An **assetset** can include different types of assets; it is not limited to assets of one type.

There are other assetset tags available with CS-Direct Advantage. (See [Chapter 3, “CS-Direct Advantage XML Tags,”](#) for more information.)

ASSETSET.ESTABLISHRATINGS

Enumerates the assets in the named set and recalculates the rating for each one according to the rules that apply to it.

Syntax

```
<ASSETSET. ESTABLISHRATINGS
    NAME="assetsetname"
    [DEFAULTRATING="rating"]/>
```

Parameters

NAME (required)

Input parameter. Name of the assetset object.

DEFAULTRATING (optional)

Input parameter. Default rating for assets that do not have one. If you do not declare a value, unrated assets are assigned a default rating of 50 on a scale of 0-100.

Description

This tag can be used to refresh the ratings of the assets in the set when visitor data that could effect their ratings has changed since the first time the assets' ratings were requested.

If the assetset on which it is invoked is not sufficiently constrained, this tag might take considerable time and memory to return a result. Web designers should take this into account when designing the site, and try to limit the size of assetsets on which this operation is performed.

Ratings are established (refreshed) and stored for all the assets in the set. Assets that return with a value of 0 are deleted from the set.

Note

If all the assets return with a 0 rating, the assetset is empty. Check the assetset after you use this tag to determine whether the set has assets. For example, use `<IF COND="list.numRows=0"/>` and then code an appropriate action if the assetset is empty.

This tag also invokes the [COMMERCECONTEXT.GETRATINGS](#) tag. For information about ratings, see the *CSEE Developer's Guide*.

Commerce Context tags

During a visitor's session, a visitor context is created for that visitor. Five types of objects are placed there:

- Current cart (see [Chapter 3, "CS-Direct Advantage XML Tags,"](#) for descriptions of the cart XML tags)
- List of segments for which the visitor qualifies
- List of promotions for which the visitor qualifies
- Time object that is used for calculating time-based rules
- Utility object that gives you access to product attributes

All of these objects, when present, are persistent throughout the visitor session (that is, they are stored in the Content Server database and then flushed when the session ends). The visitor context is the rules engine plus any objects that have been placed in this context. There is one visitor context per session.

The rules engine is a CS-Engage component that uses a chaining algorithm to calculate the rules that define segments, recommendations, and promotions. It imposes the conditions, rules, and actions set up for segments, recommendations, and promotions on the objects placed in the visitor context.

The commerce context encompasses the visitor context and gives you access to it. The commerce context object tags are how you place objects in the visitor context. In other words, these tags are your interface to the visitor context. The commerce context tags available with CS-Direct Advantage are described in [Chapter 3, "CS-Direct Advantage XML Tags."](#)

Use these object tags to display recommendations and promotions on your application pages.

The following section describes each commerce context tag, beginning with [COMMERCECONTEXT.CALCULATEPROMOTIONS](#).

COMMERCECONTEXT.CALCULATEPROMOTIONS

Creates a list of the promotions for which the current visitor is eligible.

Syntax

```
<COMMERCECONTEXT.CALCULATEPROMOTIONS/>
```

Parameters

None.

Description

This tag examines the available visitor data for the visitor, compares it to the visitor data that defines the segments associated with all the possible promotions (in other words, it invokes [COMMERCECONTEXT.CALCULATESEGMENTS](#)), and then lists only the promotions that are a match. It lets you recalculate promotion membership.

This tag is invoked whenever you use any of these tags:

```
COMMERCECONTEXT.DISCOUNTCART  
COMMERCECONTEXT.DISCOUNTTEPCART  
COMMERCECONTEXT.GETPROMOTIONS  
COMMERCECONTEXT.GETRECOMMENDATIONS  
COMMERCECONTEXT.GETSINGLERECOMMENDATION
```

You use this tag to recalculate or refresh the promotion list when visitor data that could effect the promotion list for this visitor has changed since the promotion list was last requested during this session. This action is done the first time promotion information is required in a session.

COMMERCECONTEXT.CALCULATESEGMENTS

Lists the segments to which the visitor belongs.

Syntax

```
<COMMERCECONTEXT.CALCULATESEGMENTS/>
```

Parameters

None.

Description

This tag examines the available visitor data, compares it to the data types that define the segments, and then lists the segments that are a match. It lets you recalculate segment membership based on current information.

This tag is invoked whenever you use any of these tags:

```
COMMERCECONTEXT.CALCULATEPROMOTIONS  
COMMERCECONTEXT.DISCOUNTCART  
COMMERCECONTEXT.DISCOUNTTEPCART  
COMMERCECONTEXT.GETPROMOTIONS  
COMMERCECONTEXT.GETRATINGS  
COMMERCECONTEXT.GETRECOMMENDATIONS  
COMMERCECONTEXT.GETSEGMENTS  
COMMERCECONTEXT.GETSINGLERECOMMENDATION
```

You use this tag during the debugging phase of development. The segment list object is updated to contain only the segments for which the visitor qualifies. This action is done the first time segment information is required in a session.

COMMERCECONTEXT.DISCOUNTCART

Examines the current promotion list to determine whether there are any appropriate discount rules.

Syntax

```
<COMMERCECONTEXT.DISCOUNTCART/>
```

Parameters

None.

Description

If there are discount rules, this tag applies the discount to the current shopping cart. If the promotion list does not yet exist, this tag invokes

[COMMERCECONTEXT.GETPROMOTIONS](#) to obtain the list.

COMMERCECONTEXT.DISCOUNTTEMPCART

Examines the current promotion list to determine whether there are any appropriate discount rules.

Syntax

```
<COMMERCECONTEXT.DISCOUNTTEMPCART  
  CART="cartname"/>
```

Parameters

CART (required)

Input parameter. Name of the cart to discount (not the current cart).

Description

If there are discount rules, this tag applies the discount to the shopping cart that is passed into the argument. If the promotion list does not yet exist, this tag invokes

[COMMERCECONTEXT.GETPROMOTIONS](#) to obtain one.

The current cart is not discounted.

You use this tag to display a discounted price on a page. For example, one way to show that an item is reduced in price (that is, on sale) is to create a temporary cart, put that item in that cart, use this tag to apply a special discount to the temporary cart, and then display the resulting price on the page next to text that announces the discount.

COMMERCECONTEXT.GETPROMOTIONS

Creates the list of promotions for which the current visitor is eligible.

Syntax

```
<COMMERCECONTEXT.GETPROMOTIONS  
  LISTVARNAME="promotionlist"/>
```

Parameters

LISTVARNAME (required)

Input and output parameter. As input, name of the list to create on output of the promotions for which the current visitor is eligible.

Description

The COMMERCECONTEXT.GETPROMOTIONS tag is invoked whenever a tag that needs to consult the promotion list is invoked and then finds that the promotion list does not yet exist. It is invoked by the following tags if they do not find the promotion list:

```
COMMERCECONTEXT.GETRECOMMENDATIONS  
COMMERCECONTEXT.GETSINGLERECOMMENDATION  
COMMERCECONTEXT.DISCOUNTCART  
COMMERCECONTEXT.DISCOUNTTEMPCART
```

To create the list of promotions, COMMERCECONTEXT.GETPROMOTIONS invokes the [COMMERCECONTEXT.CALCULATEPROMOTIONS](#) tag.

The list is created unsorted; it can be used to display appropriate promotions for the current visitor. You use this tag during the debugging phase of your application design.

COMMERCECONTEXT.GETRATINGS

Calculates the ratings of the assets in a named list.

Syntax

```
<COMMERCECONTEXT.GETRATINGS  
  ASSETS="assetlist"  
  LISTVARNAME="ratinglist"  
  DEFAULTRATING="defaultrating"/>
```

Parameters

ASSETS (required)

Input parameter. Name of the list of assets whose ratings you want to calculate. Its columns are assettype and assetid.

LISTVARNAME (required)

Input and output parameter. As input, name of the list to create on output that stores the assets with their ratings. Its columns are assettype, assetid, and rating. The list of assets is created and ordered according to the segment ratings associated with the assets.

DEFAULTRATING (required)

Input parameter. Default rating to use for assets that do not have a rating assigned to them yet.

Description

This tag calculates ratings according to how important the asset is to this visitor, based on the segments to which the visitor belongs. This tag is invoked whenever you use any of these tags:

COMMERCECONTEXT.GETRECOMMENDATIONS
COMMERCECONTEXT.GETSINGLERECOMMENDATION
ASSETSET.ESTABLISHRATINGS

You use this tag during a debugging phase of your application development.

COMMERCECONTEXT.GETRECOMMENDATIONS

Retrieves and lists the assets that match the recommendation constraints passed to the tag.

Syntax

```
<COMMERCECONTEXT.GETRECOMMENDATION
  COLLECTION="recommendationname"
  [LIST="inputlist"]
  [VALUE="rating"]
  [MAXCOUNT="assetcount"]
  LISTVARNAME="assetlist"
  [FILTER="true|false"]/>
```

Parameters**COLLECTION (required)**

Input parameter. Name of the recommendation. The sort and selection criteria defined in the recommendation are used to create the list of possible assets. You can constrain this list by using the MAXCOUNT argument, defined below. If there are any promotions in place that override this recommendation, it substitutes the name of the promotion, instead.

LIST (optional)

Input parameter. name of the list of assets you want to be used as the input for the calculation. This argument is applicable only if the recommendation named by COLLECTION is a context-based recommendation. Columns are assettype and assetid.

VALUE (optional)

Input parameter. Default rating for assets that do not have one. If you do not declare a value, unrated assets are assigned a default rating of 50 on a scale of 1-100.

MAXCOUNT (optional)

Input parameter. Maximum number of assets to return. Use this value to constrain the list of recommended assets.

LISTVARNAME (required)

Input and output parameter. As input, name you want to assign to the list of assets returned on output. Its columns are: assettype and assetid.

FILTER (optional)

Input parameter. A Boolean value: `true` specifies that no assets in the input list can be returned as output; `false` (default) allows input assets to be returned as output.

Description

This tag invokes the [COMMERCECONTEXT.GETSEGMENTS](#) and [COMMERCECONTEXT.GETPROMOTIONS](#) tags. For information about creating recommendations, including context-based recommendations, see the *CSEE Developer's Guide*.

COMMERCECONTEXT.GETSEGMENTS

Retrieves the list of segments to which the current visitor belongs.

Syntax

```
<COMMERCECONTEXT.GETSEGMENTS  
  LISTVARNAME="segment list"/>
```

Parameters

LISTVARNAME (required)

Input and output parameter. As input, name of the list to create on output of segments to which the visitor belongs. It has one column named `assetid`.

Description

This tag invokes [COMMERCECONTEXT.CALCULATESEGMENTS](#) to generate the list. This tag is performed whenever you use any of these tags:

```
COMMERCECONTEXT.GETRECOMMENDATION  
COMMERCECONTEXT.GETRECOMMENDATIONS  
COMMERCECONTEXT.CALCULATEPROMOTIONS  
COMMERCECONTEXT.GETPROMOTION  
COMMERCECONTEXT.GETPROMOTIONS
```

You use this tag during the debugging phase of development.

COMMERCECONTEXT.GETSINGLERECOMMENDATION

Returns one recommended asset based on the recommendation criteria passed to the tag.

Syntax

```
<COMMERCECONTEXT.GETSINGLERECOMMENDATION  
  COLLECTION="recommendationname"  
  [LIST="input list"]  
  [VALUE="rating"]  
  VARNAME="assetid"  
  TYPEVARNAME="assettype"  
  [FILTER="true|false"]/>
```

Parameters

COLLECTION (required)

Input parameter. Name of the recommendation. The sort and selection criteria defined in the recommendation are used to determine which asset to return. If there are any promotions in place that override this recommendation, it substitutes the name of the promotion, instead.

LIST (optional)

Input parameter. Name of the list of assets you want to be used as input for the calculation. This argument is only applicable if the recommendation named by COLLECTION is an context-based recommendation. Columns are `assettype` and `assetid`.

VALUE (optional)

Input parameter. Default rating for assets that do not have one. If you do not declare a value, unrated assets are assigned a default rating of 50 on a scale of 1-100.

VARNAME (required)

Input and output parameter. Name you want to assign to the variable that holds the ID of the asset that is returned on output.

TYPEVARNAME (required)

Input and output parameter. Name you want to assign to the variable that holds the type of asset that is returned on output.

FILTER (optional)

Input parameter. A boolean value: `true` specifies that no assets in the input list can be returned as output; `false` (default) allows input assets to be returned as output.

Description

This tag can be used to feature one product or to put one product on sale. This tag automatically invokes the [COMMERCECONTEXT.GETSEGMENTS](#) and [COMMERCECONTEXT.GETPROMOTIONS](#) tags. For information about creating recommendations, including context-based recommendations, see the *CSEE Developer's Guide*.

COMMERCECONTEXT.SETPROMOTIONCUTOFF

Sets a value to compare against to determine promotion eligibility.

Syntax

```
<COMMERCECONTEXT.SETPROMOTIONCUTOFF
  VALUE="cutoff"/>
```

Parameters

VALUE (required)

Input parameter. This number is compared against each promotion's rating to determine if the visitor is eligible for the promotion.

Description

This tag updates the promotion rating threshold for promotion eligibility. Prior to using the tag, the default rating is 50.

COMMERCECONTEXT.SETSEGMENTCUTOFF

Sets a value to compare against to determine segment membership.

Syntax

```
<COMMERCECONTEXT.SETSEGMENTCUTOFF  
  VALUE="cutoff"/>
```

Parameters

VALUE (required)

Input parameter. This number is compared against each segment's rating to determine whether the visitor is a member of the segment.

Description

This tag updates the segment rating threshold for segment membership. Prior to using the tag, the default rating is 50.

Visitor Data Manager Tags

The visitor data manager is the data access method that CS-Engage uses when processing rules. The visitor data manager enables you to gather, store, and retrieve visitor information from both the local, Content Server database as well as various external databases — Transact, for example.

When you create visitor attributes, history attributes, and history types for defining segments, the visitor data manager stores them in the local CS-Engage database. If you are using Transact, you can select Transact visitor attributes when defining segments as well. Then, when a method that determines which segments a visitor qualifies for is invoked, the visitor data manager checks the appropriate database for the needed visitor data. It provides a single access method to all of your visitor data. For information about creating visitor attributes, history attributes, and history types, see the *CSEE Developer's Guide*.

There are other visitor data manager object methods available with CS-Direct Advantage. These object methods let you link visitor sessions marked with different identifiers by assigning the same alias to those sessions.

To maintain visitor session information after visitors end their sessions, you use persistent cookies. These cookies serve two purposes: to store the data and to mark the session with an identifier.

A visitor's Transact ID is also an identifier that can mark a visitor session. If a visitor performs an action on the site that requires him to log into Transact, his Transact ID is passed to his hard drive and identifies him as a valid user.

Use the [VDM.RECORDHISTORY](#) and [VDM.SETSCALAR](#) tags to write visitor data to the CS-Engage database. Use the [VDM.RECORDHISTORY](#) tag to write a time-stamped record of the data defined in a history type to the database. This data could include information from several history attributes, depending on the history type. Use the [VDM.SETSCALAR](#) tag to write data for individual visitor attributes to the database.

Use the GET tags retrieve visitor data from either the local or external databases. (Note that whenever one of the COMMERCECONTEXT methods invokes a SEGMENT tag, they retrieve this data automatically — you do not have to use the VDM.GET tags in those cases.)

Note

Because the visitor data manager does not track revisions you cannot create segments based on changes in visitor data. For example, two sessions ago a visitor changed his job description from “technical” to “financial.” When that change was written to the database, the value “financial” replaced the value “technical” and there is no record of his previous job description. Now, when segments based on job description are calculated, he qualifies only for the ones that include “financial” as an option.

VDM.GETHISTORYCOUNT

Retrieves the number of history records recorded for this visitor that match the specified criteria.

Syntax

```
<VDM.GETHISTORYCOUNT  
    ATTRIBUTE="attribute"  
    VARNAME="varname"  
    [STARTDATE="date1"]  
    [ENDDATE="date2"]  
    [LIST="constraints"]/>
```

Parameters

ATTRIBUTE (required)

Input parameter. Name of a history type or name of an external history vector (that is, a vector of history data stored in the Transact database).

VARNAME (required)

Input and output parameter. As input, name of a variable to create on output to hold the value of the count.

STARTDATE (optional)

Input parameter. Start date, in milliseconds, from epoch.

ENDDATE (optional)

Input parameter. End date, in milliseconds, from epoch.

LIST (optional)

Input parameter. Name of a list of values to use to constrain the count. The list has two columns:

- **field** – Name of an individual history attribute.
- **value** – Corresponding value for the history attribute.

Use this parameter if you want to count only the instances of this record that match specific criteria. There might be more than one row for the same field.

Description

You can use this tag to count the number of times the visitor has viewed a specific advertisement and then display something on the current page based on that number.

VDM.GETHISTORYEARLIEST

Retrieves the timestamp of the first time the specified history type was recorded for this visitor.

Syntax

```
<VDM.GETHISTORYEARLIEST
  ATTRIBUTE="attribute"
  VARNAME="varname"
  [STARTDATE="date1"]
  [ENDDATE="date2"]
  [LIST="constraints"] />
```

Parameters

ATTRIBUTE (required)

Input parameter. Name of a history type or name of an external history record (that is, a record of history data stored in the Transact database).

VARNAME (required)

Input and output parameter. As input, name of a variable to create on output to hold the timestamp.

STARTDATE (optional)

Input parameter. Start date, in milliseconds, from epoch.

ENDDATE (optional)

Input parameter. End date, in milliseconds, from epoch.

LIST (optional)

Input parameter. Name of a list of values to use to constrain the date. The list has two columns:

- **field** – Name of an individual history attribute.
- **value** – Corresponding value for the history attribute.

Use this argument if you want to consider only specific instances of this record. There might be more than one row for the same field.

VDM.GETHISTORYLATEST

Retrieves the timestamp of the last time (that is, the most recent time) the specified history type was recorded for this visitor.

Syntax

```
<VDM.GETHISTORYLATEST
  ATTRIBUTE="attribute"
  VARNAME="varname"
  [STARTDATE="date1"]
  [ENDDATE="date2"]
  [LIST="constraints"] />
```

Parameters

ATTRIBUTE (required)

Input parameter. Name of a history type or name of an external history record (that is, a record of history data stored in the Transact database).

VARNAME (required)

Input and output parameter. As input, name of a variable to create on output to hold the timestamp.

STARTDATE (optional)

Input parameter. Start date, in milliseconds, from epoch.

ENDDATE (optional)

Input parameter. End date, in milliseconds, from epoch.

LIST (optional)

Input parameter. Name of a list of values to use to constrain the date. The list has two columns:

- field – Name of an individual history attribute.
- value – Corresponding value for the history attribute.

Use this argument if you want to consider only specific instances of this record. There might be more than one row for the same field.

VDM.GETHISTORYSUM

Sums the entries in a specific field for the specified history type.

Syntax

```
<VDM.GETHISTORYSUM
  ATTRIBUTE="attribute"
  FIELD="fieldname"
  VARNAME="varname"
  [STARTDATE="date1"]
  [ENDDATE="date2"]
  [LIST="constraints"]/>
```

Parameters

ATTRIBUTE (required)

Input parameter. Name of a history type or name of an external history vector (that is, a vector of history data stored in the Transact database).

FIELD (required)

Input and output parameter. Name of the history attribute in the history type, or the vector attribute in the external history vector to add.

VARNAME (required)

Input and output parameter. As input, name of a variable to create on output to hold the sum.

STARTDATE (optional)

Input parameter. Start date, in milliseconds from epoch.

ENDDATE (optional)

Input parameter. End date, in milliseconds from epoch.

LIST (optional)

Input parameter. Name of a list of values to use to constrain the sum. The list has two columns:

- `field` – Name of an individual history attribute.
- `value` – Corresponding value for the history attribute.

Use this argument if you want to sum only the instances of this record that match specific criteria. There might be more than one row for the same field.

Description

You can use this tag to add the amount of money a visitor spent during a period of time.

VDM.GETSCALAR

Retrieves a specific single (scalar) visitor attribute for the current visitor.

Syntax

```
<VDM.GETSCALAR
  ATTRIBUTE="attribute"
  VARNAME="varname"/>
```

Parameters**ATTRIBUTE (required)**

Input parameter. Name of a visitor attribute or the name of an external scalar visitor data type; that is, a scalar value for visitor data stored in the Transact database.

VARNAME (required)

Input and output parameter. As input, name of a variable to create on output for this attribute with the value for this visitor attribute.

Description

You can use this tag to display a personal message to the visitor based on this information.

VDM.LOADSCALAROBJECT

Retrieves an object stored as a single (scalar) visitor attribute.

Syntax

```
<VDM.LOADSCALAROBJECT
  ATTRIBUTE="attribute"
  VARNAME="varname"/>
```

Parameters**ATTRIBUTE (required)**

Input parameter. Name of a visitor attribute.

VARNAME (required)

Input and output parameter. As input, name of a variable to create on output to hold the object retrieved from the attribute.

See Also

[VDM.SAVESCALAROBJECT](#)

VDM.RECORDHISTORY

Saves a time-stamped copy of the specified history type with its current values for the visitor to the database.

Syntax

```
<VDM.RECORDHISTORY  
    ATTRIBUTE="attribute"  
    LIST="valuelist"/>
```

Parameters

ATTRIBUTE (required)

Input parameter. Name of a history type.

LIST (required)

Input parameter. Name of the list of values to record in the database. The list has two columns:

- field – Name of an individual history attribute.
- value – Corresponding value for the history attribute.

There is only one row for the same field.

Description

Use this tag to save the history data you have gathered about this visitor so that the segments can consider it.

VDM.SAVESCALAROBJECT

Saves an object to the Content Server database as a visitor attribute.

Syntax

```
<VDM.SAVESCALAROBJECT  
    ATTRIBUTE="attribute"  
    OBJECT="objectname"/>
```

Parameters

ATTRIBUTE (required)

Input parameter. Name of the visitor attribute to save the object as.

OBJECT (required)

Input parameter. Name of the object to save as the visitor attribute. This object must implement either the standard Java interface, `Serializable`, or the Content Server implementation of the standard Java interface, `IStorableObject`.

Description

Use this tag to store a shopping cart across sessions.

Example

For example, using the CS-Engage visitor attribute form, you create a visitor attribute of type binary and name it `storedcart`.

You code your application pages to create a cart to be stored when the session ends:

```
<CART.CREATE NAME="localcart"/>
```

Then you use this tag to store that cart object as the visitor attribute named `storedcart`:

```
<VDM.SAVESCALAROBJECT ATTRIBUTE="storedcart" OBJECT="localcart"/>
```

The next time the visitor arrives at the site, you can use

```
<VDM.LOADSCALAROBJECT ATTRIBUTE="storedcart"/>
```

to retrieve the stored cart.

VDM.SETSCALAR

Writes the data for a visitor attribute to the CS-Engage database.

Syntax

```
<VDM.SETSCALAR
    ATTRIBUTE="attribute"
    VALUE="value"/>
```

Parameters

ATTRIBUTE (required)

Input parameter. Name of the visitor attribute.

VALUE (required)

Input and output parameter. As input, value to record for this visitor on output.

Description

Use this tag to save the visitor data you have gathered about the visitor so that the segments can consider it.

Example

```
<!-- Write the registration information to the database.-->
<VDM.SETSCALAR ATTRIBUTE="name" VALUE="Variables.name"/>
<VDM.SETSCALAR ATTRIBUTE="age" VALUE="Variables.age"/>
<VDM.SETSCALAR ATTRIBUTE="jobdesc" VALUE="Variables.jobdesc"/>
```


Chapter 5

Analysis Connector XML Tags

This chapter describes the Analysis Connector XML tags and elements to manage tables and to store and retrieve event data. It also describes which ACLs you need to perform these tag functions.

The chapter contains the following sections:

- [Analysis Connector Security](#)
- [Administrative tags](#)
- [Data Retrieval Tags](#)
- [Data Storage Tags](#)

Analysis Connector Security

Analysis Connector tags are used to create, delete, read, and flush tables. Only logged-in users that have the VisitorAdmin or SiteGod ACLs in their user profiles can use these tags. All Analysis Connector tables created with the [CDM.CREATETABLES](#) tag have the SiteGod and VisitorAdmin ACLs, which are stored in the `systeminfo` table. Users who log into the DatabaseLoader must have these ACLs in their user profiles.

To perform read and write operations on Analysis Connector tables, your user profile requires one of the following ACLs:

- VisitorAdmin
- SiteGod

Administrative tags

The following section describes the administrative tags, beginning with [CDM.CREATETABLES](#).

CDM.CREATETABLES

Creates tables in the Content Server database.

Syntax

```
<CDM.CREATETABLES
  TABLENAMES="tablelist"/>
```

Parameters

TABLENAMES (required)

Input parameter. List of tables to create. Use spaces or tabs to separate names.

Description

Use this tag to create the default tables and any new tables you have defined in the `commercedata.ini` file in the Content Server database. The tables you wish to create must be listed in the `cd.tablenames` property of the `commercedata.ini` file. You can create multiple tables in a single call. You must have the SiteGod or TableEditor ACL in your user profile to use this tag.

The tables created are assigned the following ACLs:

- VisitorAdmin
- SiteGod

Example

```
<!-- Create the five default tables predefined in commercedata.ini
-->
```

```
<CDM.CREATETABLES TABLENAMES="orders order_item cart_event shopper
shopper_map"/>
```

CDM.DELETETABLES

Deletes tables from the Content Server database.

Syntax

```
<CDM.DELETETABLES
  TABLENAMES="tablelist"/>
```

Description

Use this tag to delete tables from the Content Server database when you are uninstalling a product or prior to performing an upgrade. You can delete multiple tables in a single call.

The tables you wish to delete must be listed in the `cd.tablenames` property of the `commercedata.ini` file, or the table deletion fails. You must have the SiteGod ACL or the TableEditor ACL in your user profile before you can use this tag.

Parameters

TABLENAMES (required)

Input parameter. List of tables to delete. Use spaces or tabs to separate names.

CDM.FLUSHTABLEROWS

Purges all rows in the specified tables that were created on or before a specified date.

Syntax

```
<CDM.FLUSHTABLEROWS  
  TABLENAMES="tablelist"  
  BEFOREDATE="date" />
```

Parameters

TABLENAMES (required)

Input parameter. List of tables from which to purge rows. Use spaces or tabs to separate names.

BEFOREDATE (required)

Input parameter. A date expressed in long format; that is, in milliseconds from the epoch. All rows in the specified tables entered on and before this date are purged. The tables on which you wish to use the CDM.FLUSHTABLEROWS tag must be listed in the `cd.tablenames` property of the `commercedata.ini` file.

Description

This tag is used in the element `OpenMarket/CommerceData/FlushTableRows`. A page request to `OpenMarket/CommerceData/FlushTableRows&thru=yyyy-mm-ddhh:mm:ss`, results in a call to [CDM.CONVERTDATETOLONG](#) to convert the entered date to long format; that is, to milliseconds from the epoch. The converted date is then used as input to the CDM.FLUSHTABLEROWS tag. You must have the SiteGod ACL or the TableEditor ACL in your user profile to use this tag.

Data Retrieval Tags

Data retrieval tags are used to retrieve data stored in Analysis Connector tables. The following section describes each data retrieval tag, beginning with [CDM.GETCOMMERCEDATA](#).

CDM.GETCOMMERCEDATA

Retrieves rows of data from the specified table.

Syntax

```
<CDM.GETCOMMERCEDATA
  TABLENAME="tablename"
  BEFOREDATE="date"
  AFTERDATE="date"
  CDMLIST="listvar" />
```

Parameters

TABLENAME (required)

Input parameter. Name of the table from which to retrieve data.

BEFOREDATE (required)

Input parameter. A date expressed in long format; that is, in milliseconds from the epoch. Retrieves rows created on or before this (end) date.

AFTERDATE (required)

Input parameter. A date expressed in long format, that is, in milliseconds from the epoch. Retrieves rows created since this (start) date.

CDMLIST (required)

Input and output parameter. As input, a variable list name in which to store rows of data from the specified table on output. The list has column names that match the columns of the table being called.

Description

This tag is used in the element `OpenMarket/CommerceData/retrieve/GetDataFromTable`. A page request to `OpenMarket/CommerceData/GetCommerceData&tablename=&from=yyyy-mm-ddhh:mm:ss&thru=yyyy-mm-ddhh:mm:ss`, results in a call to [CDM.CONVERTDATETOLONG](#) to convert the entered dates to long format; that is, to milliseconds from the epoch. The converted dates are then used as input to the `CDM.GETCOMMERCEDATA` tag, the `from` date representing the `AFTERDATE` value, and the `thru` date, the `BEFOREDATE` value. These dates are compared against the `last_update` (creation date) column in the default tables and in user-defined tables containing this column.

CDM.GETTABLEDEF

Retrieves the definition of a table.

Syntax

```
<CDM.GETTABLEDEF  
    TABLENAME="tablename"  
    CDMLIST="listvar" />
```

Parameters

TABLENAME (required)

Input parameter. Name of the table for which to retrieve the table definition.

CDMLIST (required)

Input and output parameter. As input, a variable list name in which to store the table definition. The list has the following columns:

- COLNAME
- COLTYPE
- COLSIZE

Description

This tag retrieves the column names and data types from the `commercedata.ini` file to predefine the format of page output when calling [CDM.GETCOMMERCEADATA](#), so that the rows of data can be distributed across the proper columns. This tag is used in the element `OpenMarket/CommerceData/retrieve/GetDataFromTable`.

Data Storage Tags

Data storage tags are used to instrument pages viewed and executed by web site visitors. The following section describes the data storage tags, beginning with [CDM.WRITECOMMERCE DATA](#).

CDM.WRITECOMMERCE DATA

Logs a row of data for a given Analysis Connector table.

Syntax

```
<CDM.WRITECOMMERCE DATA
    TABLENAME="tablename"
    $COLUMNNAME="data value"/>
```

Parameters

TABLENAME (required)

Input parameter. Name of the table to which to write a row of data.

\$COLUMNNAME (required)

Input parameter. \$COLUMNNAME is the name of the column; there are as many of these as columns in the table. There is a data value for each column in the specified table.

Description

Embed this tag in HTML pages to capture events triggered by visitor actions on the page. Such events might be the visit itself, making a selection, clicking a button, and so on. These events are mapped to column names and corresponding data types. Every column must be accounted for. The underlying Java substitutes a 0 (zero) for missing data for any NULL column, but stops processing if there is missing data for a NOT NULL column. The web designers are responsible for specifying the appropriate data types as to input and column name.

This tag checks rendermode and logs data to disk if it is set to null or live.

Example

The element `OpenMarket/CommerceData/WriteExampleData` demonstrates how this tag can be used with each of the default tables.

CDM.FITTOINTERVAL

Slots a raw number into a calculated interval.

Syntax

```
<CDM.FITTOINTERVAL  
    RAWNUMBER="number"  
    TYPE="some type"  
    INTERVAL="intervalvar"/>
```

Description

This tag logs a raw number, such as age or income, as an interval (range). The raw number is converted into a range by the tag, which then stores the calculated value in a string.

Parameters

RAWNUMBER (required)

Input parameter. A string numeric value to fit into an interval.

TYPE (required)

Input parameter. Type of interval to which to fit the raw number. There are two default types defined in the `commercedata.ini` file: `age` and `income`. There can also be user-defined types. Each type has two variants: `typelowest` and `typeinterval`.

INTERVAL (required)

Input and output parameter. As input, name of variable to hold a calculated interval on output.

Example

Given default values of `agelowest=5` and `ageinterval=10`, the following tag returns the value "5-14" in the variable `range`.

```
<CDM.FITTOINTERVAL RAWNUMBER="13" TYPE="age" INTERVAL="range"/>
```

CDM.CONVERTDATETOLONG

Converts a date-time format to a long format.

Syntax

```
<CDM.CONVERTDATETOLONG  
    INPUTDATE="datetime"  
    OUTPUTDATE="longvar"/>
```

Parameters

INPUTDATE (required)

Input parameter. A date of the format `yyyy-mm-ddhh:mm:ss`. Note that there is no space between `dd` and `hh`.

OUTPUTDATE (required)

Input and output parameter. As input, name of a variable to hold the converted long format date on output.

Description

This tag handles HTTP requests that include date-time formats. It converts a date of the form `yyyy-mm-ddhh:mm:ss` to a long format; that is, the number of milliseconds from the epoch. The long format is then used as input to the [CDM.FLUSHTABLEROWS](#) and [CDM.GETCOMMERCE DATA](#) tags.

Chapter 6

CS-Satellite XML Tags

As a page developer, you can control the following:

- Which blobs and pagelets are cached on CS-Satellite.
- How long each blob and pagelet is cached on CS-Satellite.

You control the Satellite servlet through XML tags running on Content Server. That is, you do not place XML tags into CS-Satellite itself; rather, you control the Satellite servlet indirectly through Content Server.

Before reading this chapter, you must be familiar with the Content Server XML concepts described in the *CSEE Developer's Guide*. This chapter contains the following sections:

- [Alphabetical List of CS-Satellite XML Tags](#)

Alphabetical List of CS-Satellite XML Tags

The following section describes each CS-Satellite XML tag, beginning with [satellite.blob](#).

satellite.blob

Generates a URL for a blob to be served from either Content Server using BlobServer or Satellite Server.

Syntax

```
<satellite.blob [service="service_name"]
    [name1="value1"]
    ...
    [nameN="valueN"]
    blobtable="content_catalog_name"
    blobkey="primary_key_name"
    blobwhere="primary_key_value"
    blobcol="column_name"
    blobheader="MIME_type"
    [csblobid="session variable value"]
    [blobheadername1="content type"]
    [blobheadervalue1="image/gif"]
    [blobheadernameN="name"]
    [blobheadervalueN="value"]
    [cachecontrol="expiration_date_and_time"]
    [urlbase="urlbase info"]
    [outstring="outsting info"]
    [classname="classname info"]
    [overridetype="override type"]
    [blobnocache="true|false"]/>
```

Parameters

service (optional)

Specify the start of the HTML tag to hold this blob. For example, if the blob is an image, set the value of *service* to `img src`. If this parameter is not specified, the default is `img src`. Ensure that the service is compatible with the value of *blobheader*.

name (optional)

Specify zero or more valid name/value modifiers to the HTML tag specified by *service_name*. For example, if you set *service* to `img src`, then you can specify modifiers that could go inside an HTML `<img>` tag. For example:

```
width="200"
height="300"
align="left"
```

blobtable (required)

Name of the content or object table containing the binary data.

blobkey (required)

Name of the column used as the table's primary key.

blobwhere (required)

Value of the primary key for the row containing the binary data.

blobcol (required)

Name of the column containing the binary data.

blobheader (required)

MIME type for returned data. It takes the form *description/extension*. For example, the MIME type for a GIF image is *image/gif*.

csblobid (optional)

The value of this parameter must correspond to a session variable of the same name when BlobServer security is on. When a request is made to the BlobServer, the `img src` service parameter sends a new request to the application server after the browser reads the containing page. BlobServer then verifies if the `csblobid` parameter was set. When BlobServer security is set to true and the `csblobid` value matches the session variable value, the blob is served. If the values do not match, a security violation is reported and no blob is served.

blobheadername (optional)

This parameter set specific http header variables to suit your needs, The `blobheadername` parameter is used in conjunction with the `blobheadervalue` parameter to produce name/value pairs. This parameter represents the variable name.

bloheadervalue (optional)

This parameter is used in conjunction with `blobheadername` to specify and represent http header variable values.

cachecontrol (optional)

If you omit this tag, CS-Satellite uses the value of the `expiration` property to determine how long to cache this blob. You can override the default expiration value by specifying one of the following:

- `never` – This blob should never expire for time reasons. Such objects are not guaranteed to stay in the cache forever. For example, if the cache is full, CS-Satellite still removes blobs on a LRU (Least Recently Used) basis.
- `immediate` – Expire this blob immediately; in other words, to not cache this page.
- `expiration_date_and_time` – Specifies the precise date(s) and time(s) this blob should expire.
- `session:expiration_date_and_time` – Specifies the precise date(s) and time(s) this blob should expire. Placing the word `session` in front of `expiration_date_and_time` tells CS-Satellite to serve this blob only to the person who originally requested it.

urlbase (optional)

Specifies the entire URL up to the query string excluding the question mark (?). This parameter can be used to hard code another server or servlet. The default value is `cigpath+blobserver`.

outstring (optional)

If this parameter is specified, the URL is set to a variable. If not, an HTML tag is emitted that corresponds to the value of the service parameter. If no service parameter is specified, `img src` is used.

classname (optional)

This parameter is used by the export mechanism to allow the class implementing the `iReference` interface to be overridden. The default value is `COM.FutureTense.Export.DiskBlobRef`.

ovveridetype (optional)

This parameter is used by the export mechanism to call the blob to be hard coded as dynamic or static after the code is exported. The default is found in context, see the export code for more information.

blobnocache (optional)

Whether or not to disable blob caching. Setting this parameter to `true` prevents the blob from being cached.

Description

This tag gets a blob. If the specified blob is already cached on this CS-Satellite, `satellite.blob` loads it into the page. If the specified blob is not already cached on this CS-Satellite, `satellite.blob` extracts the blob from a table, caches it, and loads it into the page. Keep in mind that this tag works regardless of the cookie setting on the browser.

This tag overrides the default caching expiration time defined by the CS-Satellite `expiration` property.

The `satellite.blob` tag is the preferred method to render images, downloadable documents, and other large objects that are managed by Content Server.

Note

The `satellite.blob` tag functions correctly when cookies are disabled.

Many Web sites contain images, such as banners or corporate logos, that never expire or seldom expire. Such images might appear on every page of content and be served millions of times per day. If a blob never expires (or seldom expires), your first impulse might be to use the `satellite.blob` tag and set the `cachecontrol` attribute to `never`. This is an acceptable method of serving blobs, however, it does not provide the best performance. Whenever the cached blob is requested, a lookup is performed in the cache table. Each individual lookup does not require many resources, but CS-Satellite may have to perform that same lookup millions of times per day.

An alternative way to serve never-expiring images is to do the following:

- Copy the never-expiring images to all your CS-Satellite hosts. Place them under the doc root for the Web server.
- Access them through `` HTML tags rather than through `satellite.blob` Satellite XML tags.

For example, consider a never-expiring corporate logo file named `CorporateLogo.gif`. To use the alternate scheme, you first copy the file to the Web server's doc root on all your CS-Satellite hosts. Then, instead of serving this logo through a `satellite.blob` tag, your element could simply issue a tag like the following:

```

```

Note

Be careful when using this alternative mechanism for serving never-expiring images. For example, CS-Satellite cannot warn you that one of the CS-Satellite hosts does not contain the same image file as the other hosts.

You can use the `blobheadername` and `blobheadervalue` parameters to specify http headers in the BlobServer response. For example, you can specify the content-type as follows:

```
blobheadername1="Content-Type" blobheadervalue1="image/gif"
```

If you want to use http headers to prevent browser caching of BlobServer requests for all browsers, use both HTTP 1.0 and HTTP 1.1 cache control directives. In the following example, the first header (`Cache-Control:no-cache`) is the HTTP 1.1 directive and the second header (`Pragma:no-cache`) is the HTTP 1.0 directive.

```
blobheadername1="Cache-Control" blobheadervalue1="no-cache"
blobheadername2="Pragma" blobheadervalue2="no-cache"
```

To determine how long to cache a page or pagelet, these criteria are evaluated in this order:

1. A value passed in using the `cachecontrol` parameter.
2. If there is no value in `cachecontrol`, the value in the `satellite.page.cachecontrol.default` property, found in the `futuretense.ini` property file, is used.
3. If there is no value in `satellite.page.cachecontrol.default`, the value of the `expiration` property, located in the `satellite.ini` property file, is used.

FatWire recommends that you set values for the `satellite.page.cachecontrol.default` properties that make sense for the majority of your pages and pagelets. If you want to override the default expiration time, use the `cachecontrol` parameter. Use the `expiration` property to determine the timeout for pages that are not generated from this tag; for example, pages that are generated directly from a URL.

Example

The following tag gets a blob from the `PortalImage` table that expires from the cache daily at 5:00 in the afternoon. The blob is a GIF image. In the composed page, this GIF image consumes a space 125x75 pixels with a 3-pixel border.

```
<satellite.blob
  service="img src"
  width="125"
  height="75"
  border="3"
  blobtable="PortalImage"
  blobkey="id"
  blobwhere="imagelist.id"
  blobcol="urlpicture"
  blobheader="image/gif"
  cachecontrol="17:0:0 */*/*/"/>
```

satellite.cookie

Sets a cookie on the user's browser.

Syntax

```
<satellite.cookie
  name="cookie_name"
  value="cookie_value"
  timeout="timeout"
  secure="true|false"
  url="URL"
  [domain="domain"]/>
```

Parameters

name (required)

Name of cookie to set.

value (required)

Value of the cookie being set.

timeout (required)

Required. Timeout value of cookie, in seconds. To set a session-long cookie, set this value to -1.

secure (required)

Set to `true` to indicate a secure cookie; set to `false` to indicate an non-secure cookie.

url (required)

A valid URL. This value restricts the sending of the cookie from the client to this URL only. The path must include the servlet that set the cookie, for example, `/catalog`, which makes the cookie visible to all directories on the server under `/catalog`.

domain (optional)

Optional. Specify a valid domain name. This value restricts the sending of the cookie from the client to addresses in this domain only

Description

The `satellite.cookie` tag sets a cookie on the user's browser.

HTML requires that serverside cookies must be set in the head section of a Web page. However, you don't have to put the `satellite.cookie` tag inside the head section of a Content Server element; you can place it anywhere within the element, because CS-Satellite causes the cookie to be set in a second request to the page. The `satellite.cookie` tag invokes the `CookieServer` to set the cookie.

Example

This example sets the cookie `mycookie` on the client with the value `frank`. The cookie expires 600 seconds from the time it is created.

```
<satellite.cookie
  NAME="mycookie"
  VALUE="frank"
  TIMEOUT="600"
```

```
SECURE="false"
URL="ContentServer/mysite/frankspage" />
```

satellite.flush

Flushes individual items from the cache.

Syntax

```
<satellite.flush
  SERVERN="serverurl"
  USERNAMEN="username"
  PASSWORDN="password"
```

Parameters

SERVERN (required)

URL of the server whose cache you want to flush. *N* is a number that must be the same as the number used for username and password.

USERNAMEN (required)

User name to log in to the server whose cache you want to flush. *N* is a number that must be the same as the number used for username and password.

PASSWORDN (required)

Password to log in to the server whose cache you want to flush. *N* is a number that must be the same as the number used for username and password.

Description

The `satellite.flush` tag flushes all the contents of one or more CS-Satellite caches. To flush individual components from the cache, use the CacheManger API. The CacheManager API provides a flush mechanism to Content Server and CS-Satellite.

Example

The following example flushes a page and two pagelets from the caches of the `shakespeare` and `jonson` servers:

```
<satellite.flush
  SERVER1="http://shakespeare:80"
  USERNAME1="ftuser"
  PASSWORD1="ftuser"
  SERVER2="http://jonson:80"
  USERNAME2="foo"
  PASSWORD2="bar" />
```

satellite.link

Generates links for CS-Satellite.

Syntax

```
<satellite.link
    [name1="value 1"]
    [nameN="value N"]
    [outstring="outstring info"]
    [classname="classname info"]
    [overridetype="override info"]
    [urlbase="url base info"]
    [pagename="page name"]/>
```

Parameters

name1 (optional)

Name/value pair that modifies the query string.

outstring (optional)

If this parameter is specified, no tag is returned and the primary attribute value is placed in the specified output variable. The default variable name is `referURL`.

classname (optional)

This parameter is used by the export mechanism to allow the class implementing the `IRReference` interface to be overridden. The default value is `COM.FutureTense.Export.DiskPageRef`.

overridetype (optional)

This parameter is used by the export mechanism to all the links to be hard-coded as dynamic or static after the code is exported. The default is found in context; see the export code for more information.

urlbase (optional)

Specifies the entire URL up to the query string excluding the question mark (?). This parameter can be used to hard code another server or servlet. The default value is `ft.cgipath/ContentServer`.

pagename (optional)

Name of the page the link is for.

Description

This tag generates a link for CS-Satellite. The link that is created by `satellite.link` is URL-encoded. When the export code is used, this tag allows you to specify the pages to be exported. By default, all the pages referenced are exported unless the `overridetype` parameter is used.

You can use this tag to link pages not rendered by Content Server by specifying the `urlbase` parameter and leaving `pagename` blank.

Example

```
<satellite.linkpagename="OpenMarket/Samples/NewPortal/XML/main"/>
```

```
<a REPLACEALL="Variables.referURL" HREF="Variables.referURL">
```



```

<satellite.blob service="img src"
  blobtable="Variables.tablename"
  blobkey="id"
  blobwhere="imagelist.id"
  blobcol="urlthumbnail"
  blobheader="image/gif"
  cachecontrol="CS.Property.image.time"
  ALT="imagelist.alttext"
  BORDER="0"
  WIDTH="Variables.panewidth"/>
<br/>
</a>

```

satellite.page

Gets a pagelet and specifies its cache expiration.

Syntax

```

<satellite.page pagename="pagelet_name"
  [cachecontrol="expiration_date_and_time"]
  [name1="value1"]
  ...
  [nameN="valueN"]/>

```

Parameters

pagename (required)

Specify the name of a pagelet.

cachecontrol (optional)

If you omit this parameter, CS-Satellite uses the value of the `expiration` property to determine how long to cache this pagelet. You can override the default expiration value by specifying one of the following:

- **never** – This pagelet never expires for time reasons. Such objects are not guaranteed to stay in the cache forever. For example, if the cache is full, CS-Satellite still removes pagelets on a LRU (Least Recently Used) basis.
- **immediate** – Expire this pagelet immediately; in other words, do not cache this page.
- **expiration_date_and_time** – Specifies the precise date(s) and time(s) this pagelet expires.
- **session:expiration_date_and_time** – Specifies the precise date(s) and time(s) this pagelet expires. Placing the word `session` in front of the expiration date and time tells CS-Satellite to serve this pagelet only to the current session. Typically, you specify `session:expiration_date_and_time` when the pagelet contains personalized data.

name (optional)

Specify zero or more name/value pairs to be passed to the page, just as you pass arguments to the `CONTENTSERVER` tag. The arguments must be specified as page criteria in the `resargs` columns of the `sitelog` so that the system can differentiate cached pages properly. Any attribute passed in must be specified as

PageCriteria in the sitecatalog or the page does not get cached properly.
 Pagename and cachecontrol do not need to be specified as page criteria.

errno

The possible values of errno include:

Value	Description
-30	Page not found.
-106	Bad parameters.
-406	Bad XML parameter.

Description

The Satellite equivalent of the CONTENTSERVER tag or EVALSERVER tag. If the specified pagelet is already cached on this CS-Satellite, `satellite.page` loads it. If the specified pagelet is not already cached on this CS-Satellite, `satellite.page` requests the page from Content Server, then caches it and loads it. This tag overrides the default caching expiration time defined by the CS-Satellite expiration property. Scoping for the `satellite.page` tag is set to STACKED. Cookies must be enabled for the `satellite.page` tag to function properly.

To determine how long to cache a page or pagelet, CS-Satellite evaluates these criteria in this order:

1. A value passed in using the `cachecontrol` parameter.
2. If there is no value in `cachecontrol`, the value in the `satellite.page.cachecontrol.default` property, found in the `futuretense.ini` property file, is used.
3. If there is no value in `satellite.page.cachecontrol.default`, the value of the expiration property, located in the `satellite.ini` property file, is used.

FatWire recommends that you set values for the `satellite.page.cachecontrol.default` properties that make sense for the majority of your pages and pagelets. If you want to override the default expiration time, use the `cachecontrol` parameter. Use the expiration property to determine the timeout for pages that are not generated from this tag; for example, pages that are generated directly from a URL.

Example

The following tag identifies a pagelet named `Experiment/CarOffer` that expires yearly at 2:30 p. m. on July 31.

```
<satellite.page
  pagename="Experiment/CarOffer"
  cachecontrol="14:30:0 */31/7"/>
```

The following example shows XML code for a page containing two pagelets. One pagelet (`LeftPagelet`) expires on the hour, or at 15, 30, or 45 minutes past the hour, and the other pagelet (`RightPagelet`) expires on the hour, or at 10, 20, 30, 40, or 50 minutes past the hour.

```

<satellite.tag type="open"/>
  <table border="0">
    <tr>
      <td width="400" height="600">
        <satellite.page
          pagename="Experiment/LeftPagelet"
          cachecontrol="*:0,15,30,45:0 */*/*"/>
      </td>
      <td width="300" height="600">
        <satellite.page
          pagename="Experiment/RightPagelet"
          cachecontrol="*:0,10,20,30,40,50:0 */*/*"/>
      </td>
    </tr>
  </table>
<satellite.tag type="close"/>

```

The following examples show different time formats for expiration:

- To set a pagelet to expire every day at midnight:
cachecontrol="00:00:00 */*/*"
- To set a pagelet to expire every minute, every day:
cachecontrol="*:*:00 */*/*"
- To set a pagelet to expire every hour:
cachecontrol="*:00:00 */*/*"

satellite.tag

Marks a pagelet for use by the Satellite servlet.

Syntax

```
<satellite.tag type="open|close"/>
```

Parameters

type (required)

Specify type=open to mark the start of a pagelet that CS-Satellite needs to parse; specify type=close to mark the end of this pagelet.

Description

Tells the Satellite servlet that this pagelet contains a `satellite.page`, `satellite.link`, `satellite.cookie`, or `satellite.blob` tag. If this pagelet contains `satellite.page` or `satellite.blob` directives or any form or other form, then you must provide the `satellite.tag` pair. If you omit the pair, CS-Satellite streams the page without passing it.

If this pagelet does *not* contain `satellite.page` or `satellite.blob` directives, then you should omit the `satellite.tag` pair. The exception to this is if you are using `render:getpageurl`. If you mistakenly provide the `satellite.tag` pair, CS-Satellite still serves the page but performance may suffer.

If you do need the `satellite.tag` pair, FatWire strongly recommends setting the “open” tag as the first XML tag in your element and the “close” tag as the last XML tag in your element, in between the `ftcs` tags.

Example

The following code demonstrates that `<satellite.page>` and `<satellite.blob>` tags must be inside a `<satellite.tag>` pair:

```
<satellite.tag type="open"/>

  <satellite.page ...
  ...
  <satellite.page ...
  ...
  <satellite.blob ...
  ...

<satellite.tag type="close"/>
```

Chapter 7

CS-Bridge XML Tags

Implemented as Content Server tags, the extended tag set for CS-Bridge XML enables you to perform parsing and transformation operations on different types of input XML.

Note

The tags for CS-Bridge XML must be entered in uppercase letters so that the proper debugging and error checking utilities are employed. Lowercase tags, which are not checked, may result in coding errors.

Alphabetical List of CS-Bridge XML Tags

The CS-Bridge XML tags are backed by logic that calls the XML parser and XSLT processing engine from the Content Server environment.

Note

Although the functionality of the XML tags closely parallels that of the corresponding JSP tags, the JSP tags accept XML DOM as a valid input format for XSLT transformations, while the equivalent XML tags do not.

The following section describes each CS-Bridge XML tag, beginning with [XMLUTILS.PARSE](#).

XMLUTILS.PARSE

Reads data from an input XML file according to specified parameters and returns an XML DOM object. The input file is validated against referenced DTDs, if any.

Syntax

```
<XMLUTILS.PARSE
  INPUTVALTYPE="input value type"
  INPUTVAL="file location"
  COLNAME="url"
  COLVAL="file.xml"
  OUTPUTVAL="variable name"/>
```

Parameters

INPUTVALTYPE (required)

Description of the XML data to be passed in the INPUTVAL parameter and parsed. Data can be referenced as either a URI, Content Server database table, or XML string. Depending on the type of data you plan to pass, specify one of the following valid types:

- uri
Indicates that the input XML is represented by a URI.
- catalog
Indicates that the input XML is located in a Content Server database table.
- data
Indicates that the input XML is actual XML content in string format.

INPUTVAL (required)

Input data, which must correspond to the specified INPUTVALTYPE. Input data can take any of the following forms:

- URI string that conforms to URI syntax and represents the input XML file.
- Name of the Content Server database table that contains the input data.
- XML file contents.

OUTPUTVAL (required)

Name of the variable that contains the resulting DOM object.

COLNAME (required for XML in a database table)

Name of the column in the database table where the XML file resides. This column must be defined as the csindirect (URL) data type.

COLVAL (required for XML in a database table)

Value of the column COLNAME for the row in which the XML file resides.

Description

The XMLUTILS.PARSE tag includes parameters that specify how input data is read. For each reference to XMLUTILS.PARSE, you must initialize some required parameters. Depending on the type of data you are passing in the INPUTVALTYPE parameter (URI, catalog, or XML data), you may need to set some or all of the supplied parameters. If your XML resides in a Content Server database table, you must specify all parameters.

The output of the XMLUTILS.PARSE tag is a variable.

errno

The possible values of errno include:

Value	Description
0	Success. The XML input was successfully parsed.
-100	General exception failure.

Examples

The following examples demonstrate syntax and parameters when the input file is referenced as a URI, referenced as a Content Server database table, or included as an XML string.

Input XML file represented by a URI:

```
<XMLUTILS.PARSE INPUTVALTYPE="uri" INPUTVAL= "file:///location/
filename.xml" OUTPUTVAL="parsedOutputVar"/>
```

This example accepts the input XML file in the form of a URI and returns a DOM object, which is a DOM representation of the XML referenced by the URI.

Input XML file resides in a Content Server database table:

```
<XMLUTILS.PARSE INPUTVALTYPE="catalog" INPUTVAL= "InboundCatalog"
COLNAME="url" COLVAL="filexyz.xml" OUTPUTVAL="parsedOutputVar"/>
```

This example reads the input XML file from a Content Server database table and returns a DOM object, which is a structured representation of the XML.

Input XML is a string that contains the contents of the XML File:

```
<XMLUTILS.PARSE INPUTVALTYPE="data" INPUTVAL= "<?xml version . . .
> . . . " OUTPUTVAL="parsedOutputVar"/>
```

Reads the contents of the input XML file in the form of a string and returns an object, which is the DOM representation of the XML supplied in the string.

XMLUTILS.TRANSFORM

Reads XML data from an input file according to specified parameters, transforms it using an XSLT style sheet, and returns the transformed XML.

Syntax

```
<XMLUTILS.TRANSFORM
  INPUTVALTYPE="input_value_type"
  INPUTVAL="xml_content"
  STYLESHEET="stylesheet_location"
  OUTPUTVALTYPE="output_value_type"
  OUTPUTVAL="variable_name"/>
```

Parameters

INPUTVALTYPE (required)

Description of the XML data to be passed to the INPUTVAL parameter and transformed. Data can be referenced as an XML string only. Specify the following valid type:

- stream

Indicates that the input data is actual XML content in string format.

INPUTVAL (required)

Specifies input data, which must correspond to the specified INPUTVALTYPE. Input data, which takes the following form:

- XML string

STYLESHEET (required for explicit style sheets)

Use this parameter to specify a style sheet that is external to the input XML. As a value, enter a URI that represents the style sheet. A URI is a string that conforms to URI syntax.

Note that this parameter does not apply to implicit style sheets, which are included in the input XML. When the STYLESHEET parameter is omitted, the style sheet is assumed to be implicit by default.

OUTPUTVALTYPE (required)

Specification for the XML format that the output (transformed data) takes. Data can be returned as either an XML DOM or XML string. Depending on the XML format you want as output from your style sheet, specify either of the following valid types:

- dom

Indicates that the transformed XML content is an XML DOM object.

- stream

Indicates that the output data is a string containing the transformed XML content.

OUTPUTVAL (required)

Name of the variable that contains the resulting transformed output.

Description

The XMLUTILS.TRANSFORM tag reads XML data according to specified type of XML (string), transforms it using an XSLT style sheet, and returns the transformed XML. During transformation, the input XML is also checked and validated against referenced DTDs, if any. The style sheet specified for the transformation can be implicit (included in

the XML file) or explicit (included as a separate file). The parameters you specify determine how the input data is read and how the output is returned. If the style sheet is included in a separate file, you must set the `STYLESHEET` parameter. If the style sheet is included in the XML file, the tag reads the style sheet from the input XML.

The output of the `XMLUTILS.TRANSFORM` tag is a variable.

errno

The possible values of `errno` include:

Value	Description
0	Success. The XML input was successfully transformed.
-100	General exception failure.

Examples

The following examples demonstrate syntax and parameters when the input file is an XML string.

Input XML file is a string, an explicit style sheet is applied, and the output is specified as an XML string:

```
<XMLUTILS.TRANSFORM INPUTVALTYPE="stream" INPUTVAL="<?xml version
. . . > . . . " STYLESHEET="file:///location/filename.xml"
OUTPUTVALTYPE = "stream" OUTPUTVAL="transformedOutputVar"/>
```

Accepts the input XML file in the form of a string. Note that the transformation is done using a separate style sheet, specified with the `STYLESHEET` parameter. The output, specified by the `stream` value, is a string.

XMLUTILS.TRANSFORMANDPRINT

Reads XML data from an input file according to specified parameters, transforms it using an XSLT style sheet, and prints the transformed XML.

Syntax

```
<XMLUTILS.TRANSFORMANDPRINT
  INPUTVALTYPE="input_value_type"
  INPUTVAL="xml_content"
  STYLESHEET="stylesheet_location" />
```

Parameters

INPUTVALTYPE (required)

Description of the XML data to be passed to the `INPUTVAL` parameter and transformed. Data can be referenced as an XML string only. Specify the following valid type:

- `stream`

Indicates that the input data is actual XML content in string format.

INPUTVAL (required)

Specifies input data, which must correspond to the specified **INPUTVALTYPE**. Input data, which takes the following form:

- XML string

STYLESHEET (required for explicit style sheets)

Use this parameter to specify a style sheet that is external to the input XML. As a value, enter a URI that represents the style sheet. A URI is a string that conforms to URI syntax.

Note that this parameter does not apply to implicit style sheets, which are included in the input XML. When the **STYLESHEET** parameter is omitted, the style sheet is assumed to be included in the input XML by default.

Description

The **XMLUTILS.TRANSFORMANDPRINT** tag reads XML data according to the specified type of XML (string), transforms it using an XSLT style sheet, and prints the output.

During transformation, the input XML is also checked and validated against referenced DTDs, if any. The style sheet specified for the transformation can be implicit (included in the XML file) or explicit (included as a separate file). The parameters you specify determine how the input data is read. If the style sheet is included in a separate file, you must set the **STYLESHEET** parameter. If the style sheet is included in the XML file, the tag reads the style sheet from the input XML.

The output of the **XMLUTILS.TRANSFORMANDPRINT** tag can be directly viewed within the calling page.

errno

The possible values of **errno** include:

Value	Description
0	Success. The XML input was successfully transformed and returned for viewing in the calling page.
-100	General exception failure.

Example

Input XML file is a string, and an explicit style sheet is applied:

```
<XMLUTILS.TRANSFORMANDPRINT INPUTVALTYPE="stream" INPUTVAL="<?xml
version . . . > . . . " STYLESHEET="file:///location/
filename.xsl"/>
```

This example accepts the input XML file in the form of a string. Note that the transformation is done using a separate style sheet, specified with the **STYLESHEET** parameter. The output, specified by the **stream** value, is a string. If the **STYLESHEET** parameter is omitted then the style sheet is assumed to be implicit and is read from the XML file by default. The output is returned so that it can be viewed in the calling XML.

Chapter 8

Content Server SOAP Web Services

Content Server provides enhancements that permit the construction and streaming of SOAP service responses using the normal `pathname=evaluation` path. The following set of XML tags enable you to construct soap messages using Content Server XML elements.

This chapter contains the following sections:

- [SOAP tags](#)
- [Web Services tags](#)

SOAP tags

The following set of XML tags construct SOAP messages using Content Server XML elements.

SOAP.ARGUMENT

Defines arguments for the SOAP . HEADER tag, which defines a simple SOAP header.

Syntax

```
<soap.header>  
    [<soap.argument name="argName" value="argValue"/>]  
</soap.header>
```

Parameter

ARGUMENT (optional)

Arguments passed to SOAP as the page is called.

Description

The SOAP . HEADER tag defines a simple SOAP header, which should contain at least one name/value pair or the tags should be omitted. The SOAP header and contained arguments are the first set of non-logic tags to be contained in the SOAP message. An error occurs if the SOAP header is in the wrong order.

Example

The following example defines a header for a SOAP message.

```
<soap.header>  
    <soap.argument name="DUMMY" value="VALUE"/>  
</soap.header>
```

See Also

[SOAP.BODY](#)

[SOAP.FAULT](#)

[SOAP.HEADER](#)

[SOAP.MESSAGE](#)

SOAP.BODY

Indicates the main part of a SOAP message body.

Syntax

```
<soap.body
    [tagname="local_name"] >
</soap.body>
```

Parameters

tagname (optional)

First child element after the body has this local name. The defaults is CS.

Description

The soap.body tag indicates the main part of a soap message body where user can stream data. The body tag must follow the soap header tag or an error results. The body can contain any legal XML both in the form of literal data contained in the template and/or emitted by logic coded into the template.

Example

This example calls the element SoapTestOut.

```
<SOAP.BODY TAGNAME="SOAPTESTOUT">
    <dummy xsi:type="soapresponse:CS">
        <LOOP LIST="RS" COUNT="4">
            <headlines xsi:type="soapresponse:headlineStruct">
                <id xsi:type="xsd:string">
                    <csvar NAME="RS.id"/>
                </id>
                <headline xsi:type="xsd:string">
                    <csvar NAME="RS.headline"/>
                </headline>
            </headlines>
        </LOOP>
    </dummy>
</soap.body>
```

See Also

[SOAP.HEADER](#)

[SOAP.FAULT](#)

[SOAP.MESSAGE](#)

SOAP.FAULT

Creates a status or error return message.

Syntax

```
<soap:fault
  code="error_code"
  string="error_as_string"
  [actor="cause_of_error"]
  [detail="error_details"] />
</soap:fault>
```

Parameters

CODE (required)

Code identifying the error. Default codes include:

- Version Mismatch
- MustUnderstand
- Client
- Server

STRING (required)

Error printed out as a string.

ACTOR (optional)

Cause of the error.

DETAIL (optional)

Detailed error information.

Description

SOAP messages can contain, at most, one status or error return message. This is called a SOAP fault element. Content Server does not check that more than one fault element is used. A fault element can be used for both error and status return.

The following table lists the default codes (example table taken from http://www.w3schools.com/soap/soap_errors.asp):

Error	Description
Version Mismatch	Invalid namespace for the SOAP Envelope element.
MustUnderstand	A child element of the Header element, with the mustUnderstand attribute set to "1", was not understood.
Client	The message was incorrectly formed or contained incorrect information.
Server	There was a problem with the server so the message could not proceed.

Example

This example code checks for an error, then displays the soap fault as the error message.

```
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <soap.fault
      code="Server"
      string="Nothing to do!"
      actor="database"
      detail="Dummy"/>
  </THEN>
</IF>
```

See Also

[SOAP.BODY](#)

[SOAP.HEADER](#)

[SOAP.MESSAGE](#)

SOAP.HEADER

Defines a simple SOAP header.

Syntax

```
<soap.header>
  [<soap.argument name="argName" value="argValue"/>]
</soap.header>
```

Parameter

ARGUMENT (optional)

Arguments passed to SOAP as the page is called.

Description

The `soap.header` tag defines a simple SOAP header, which should contain at least one name/value pair, otherwise the tags should be omitted. The SOAP header and contained arguments are the first set of non-logic tags to be contained in the SOAP message. An error occurs if the SOAP header is in the wrong order.

Example

The following example defines a header for a SOAP message.

```
<soap.header>
  <soap.argument name="DUMMY" value="VALUE"/>
</soap.header>
```

See Also

[SOAP.ARGUMENT](#)

[SOAP.BODY](#)

[SOAP.FAULT](#)

[SOAP.MESSAGE](#)

SOAP.MESSAGE

Processes the contents of an element.

Syntax

```
<soap.message  
    [ns="namespace_prefix"]  
    [uri="universal_resource_indicator"] >  
</soap.message>
```

Parameters

NS (optional)

Namespace prefix.

URI (optional)

Universal resource indicator.

Description

The `soap.message` tag processes the contents of an element, which must exist in ElementCatalog. This tag is used to define and surround the SOAP message and control the streaming of content contained within. Do not stream data, comments, or whitespace before or after this tag.

Example

```
<soap.message ns="mynamespace">  
    <soap.body tagname="HelloWorldOut">  
        <echoStringOut xsi:type="xsd:string">  
            <csvar NAME="Variables.echoString"/>  
        </echoStringOut>  
    </soap.body>  
</soap.message>
```

See Also

[SOAP.BODY](#)

[SOAP.FAULT](#)

[SOAP.HEADER](#)

Web Services tags

The following set of XML tags construct SOAP messages using 'generic' Content Server XML elements.

WEBSERVICES.INVOKE

Processes the content of an element.

Syntax

```
<webservices.invoke
  wsdl="http://soapinterop.java.sun.com/round2/base?WSDL"
  target="http://soapinterop.org/"
  service="Round2Base"
  port="RIBaseIFPort"
  operation="echoString"
  object="echostring">
  <webservices.parameter
    type="string"
    value="hello world" />
</webservices.invoke>
```

Parameters

WSDL (optional)

Web Services Description Language (WSDL) defining the webservice

TARGET (optional)

Target name space, as defined by the WSDL

SERVICE (optional)

A service, as defined by the WSDL, groups together a set of related ports. Together with port parameter, service specifies SOAP address

PORT (optional)

A port, as defined by the WSDL, is the combination of a binding and a network address. Together with service parameter, port specifies SOAP address.

OPERATION (optional)

Function being invoked, which is supported by the web service.

OBJECT (optional)

Content Server object name. The returned object from the web service is stored here.

Description

Target, service, port, operation are all specified in the WSDL. See the W3 web site (<http://www.w3.org/TR/wsdl>) for more information.

Example

```
<webservices.invoke
  wsdl="http://soapinterop.java.sun.com/round2/base?WSDL"
  target="http://soapinterop.org/" service="Round2Base"
  port="RIBaseIFPort" operation="echoString"
```

```
        object="echostring"
        <webservices.parameter
            type="string" value="hello world"/>
    </webervices.invoke>
```

See Also

[WEBSERVICES.PARAMETER](#)

WEBSERVICES.PARAMETER

Passes parameters to the WEBSERVICES.INVOKE tag. Must be used together with the WEBSERVICES.INVOKE tag.

Syntax

```
<webervices.invoke ... />
    <webervices.parameter
        type="string"
        value="hello world"/>
</webervices.invoke>
```

Parameters

TYPE (optional)
Name of parameter.

VALUE (optional)
Value of parameter.

Description

The `webervices.invoke` tag, along with the `webervices:parameter` tag, processes the contents of an element, which must exist in the `ElementCatalog`. These tags in conjunction are used to define and surround the SOAP message. They control the streaming of content contained within. Do not stream data, comments, or whitespace before or after these tags.

Example

```
<webervices.invoke
    wsdl="http://soapinterop.java.sun.com/round2/base?WSDL"
    target="http://soapinterop.org/" service="Round2Base"
    port="RIBaseIFPort" operation="echoString"
    object="echostring" >
    <webervices.parameter
        type="string" value="hello world"/>
</webervices.invoke>
```

See Also

[WEBSERVICES.INVOKE](#)

Section 2

JSP Tags

The following chapters in this section describe the JSP tags used in Content Server, CS-Satellite and Content Server content applications:

- [Chapter 9, “Content Server JSP Tags”](#)
- [Chapter 10, “CS-Direct JSP Tags”](#)
- [Chapter 11, “CS-Direct Advantage JSP Tags”](#)
- [Chapter 12, “CS-Engage JSP Tags”](#)
- [Chapter 13, “CS-Satellite JSP Tags”](#)
- [Chapter 14, “Content Server SOAP Web Services”](#)

Chapter 9

Content Server JSP Tags

This chapter describes Content Server JSP tags. It contains the following sections:

- [JSP Tags by Function](#)
- [Alphabetical List of JSP Tags](#)

JSP Tags by Function

The following section categorizes each JSP tag.

ics Tags

- [ics:argument](#)
- [ics:callelement](#)
- [ics:callsql](#)
- [ics:catalogbatch](#)
- [ics:catalogdef](#)
- [ics:catalogmanager](#)
- The output is as follows:
- [ics:copylist](#)
- [ics:dateargument](#)
- [ics:debug](#)
- [ics:disablecache](#)
- [ics:else](#)
- [ics:encode](#)
- [ics:fieldargument](#)
- [ics:fileargument](#)
- [ics:flushcatalog](#)
- [ics:flushstream](#)
- [ics:geterno](#)
- [ics:getproperty](#)
- [ics:getssvar](#)
- [ics:getvar](#)
- [ics:if](#)
- [ics:indexadd](#)
- [ics:indexcreate](#)
- [ics:indexdestroy](#)
- [ics:indexexists](#)
- [ics:indexremove](#)
- [ics:indexreplace](#)
- [ics:insertpage](#)
- [ics:listget](#)
- [ics:listloop](#)
- [ics:literal](#)

- [ics:logmsg](#)
- [ics:mirror](#)
- [ics:mirrorlist](#)
- [ics:mirrortreenode](#)
- [ics:popvars](#)
- [ics:pushvars](#)
- [ics:removessvar](#)
- [ics:removevar](#)
- [ics:resolvevariables](#)
- [ics:search](#)
- [ics:searchdatetnative](#)
- [ics:selectto](#)
- [ics:sendmail](#)
- [ics:setssvar](#)
- [ics:setvar](#)
- [ics:sql](#)
- [ics:sqlexp](#)
- [ics:textargument](#)
- [ics:then](#)
- [ics:queryevents](#)
- [ics:readevent](#)
- [ics:treemanager](#)
- [portal:logdep](#)
- [user:logout](#)

object Tags

- [object:argument](#)
- [object:checkin](#)
- [object:checkout](#)
- [object:children](#)
- [object:create](#)
- [object:deinstall](#)
- [object:delete](#)
- [object:gather](#)
- [object:get](#)
- [object:install](#)
- [object:isempty](#)

- [object:list](#)
- [object:load](#)
- [object:rollback](#)
- [object:root](#)
- [object:save](#)
- [object:scatter](#)
- [object:set](#)
- [object:setobject](#)
- [object:setparent](#)
- [object:stream](#)
- [object:track](#)
- [object:undocheckout](#)
- [object:untrack](#)

user Tags

- [user:add](#) (Deprecated)
- [user:delete](#) (Deprecated)
- [user:login](#)
- [user:logout](#)
- [user:modify](#) (Deprecated)
- [user:su](#)

time Tags

- [portal:logdep](#)
- [time:set](#)

ftcs Tags

- [ftcs1_0:flush](#)
- [ftcs1_0:ftcs](#)

acl Tags

- [acl:argument](#)
- [acl:create](#)
- [acl:delete](#)
- [acl:gather](#)
- [acl:get](#)
- [acl:list](#)

- [acl:load](#)
- [acl:save](#)
- [acl:scatter](#)
- [acl:set](#)

Directory Service Tags

- [dir:addattrs](#)
- [dir:addgroupmember](#)
- [dir:children](#)
- [dir:create](#)
- [dir:delete](#)
- [dir:getattrs](#)
- [dir:groupmembers](#)
- [dir:groupmemberships](#)
- [dir:listusers](#)
- [dir:removeattrs](#)
- [dir:removegroupmember](#)
- [dir:replaceattrs](#)
- [dir:search](#)

Alphabetical List of JSP Tags

The following section describes each Content Server JSP tag, beginning with [acl:argument](#).

acl:argument

Argument for other ACL tags.

Syntax

```
<acl:argument  
    name="name"  
    value="value" />
```

Parameters

name (required)
Name of the tag the argument is for.

value (required)
Value of the argument.

Description

The `acl:argument` tag declares a name/value pair that is available to the parent tag. The parent tag must be present in the ACL tag library.

ernno

None.

Example

```
<acl:create name="acl1">  
    <acl:argument name="aclname" value="myacl"/>  
    <acl:argument name="cataccess" value="1"/>  
    <acl:argument name="description" value="test"/>  
</acl:create>
```

See Also

[acl:create](#)

acl:create

Creates an ACL.

Syntax

```
<acl:create name="ACL name" >
    <acl:argument name="argument name" value="argument value" >
</acl:create>
```

Parameters

name (required)
Name of the ACL.

Description

Creates an ACL for a specified table. Use `acl:argument` to specify names, descriptions and privileges.

Values for `cataccess` (the numeric value that determines the level of permissions) are as follows:

- read = 1
- write = 2
- create = 4
- destroy = 8
- retrieve = 10
- rtaudit = 20
- rtadmin = 40

The values for `cataccess` work similarly to UNIX permissions. For example, if you want a user to have read and write permission, assign 3 to `cataccess`.

errno

The possible values of `errno` include:

Value	Description
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10010	Invalid attribute value.

Example

The following example creates an object in the object pool called `acl1`.

```
<acl:create name="acl1">
    <acl:argument name="aclname" value="myacl"/>
```

```

    <acl:argument name="cataccess" value="1"/>
    <acl:argument name="description" value="test"/>
</acl:create>

```

See Also

[acl:argument](#)

[object:delete](#)

[acl:gather](#)

[portal:logdep](#)

[object:stream](#)

[object:setobject](#)

[object:setparent](#)

[acl:scatter](#)

[acl:set](#)

acl:delete

Deletes an ACL.

Syntax

```

<acl:delete
    name="Some ACL Name"/>

```

Parameters

name (required)

Name of the ACL to be deleted.

Description

The `acl:delete` tag deletes an ACL from the ACL pool. To delete an ACL, it must first be loaded using `acl:load`.

errno

The possible values of `errno` include:

Value	Description
-105	Database error.
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

Example

The following example deletes the ACL record that corresponds to the name `acl1`.

```
<acl:delete name="acl1"/>
```

See Also

[acl:create](#)

[acl:gather](#)

[portal:logdep](#)

[object:stream](#)

[object:setobject](#)

[object:setparent](#)

[acl:scatter](#)

[acl:set](#)

acl:gather

Gathers fields into an ACL.

Syntax

```
<acl:gather
    name="ACL name"
    prefix="some prefix"
    fieldlist="field names"/>
```

Parameters

name (required)

Name of the ACL.

prefix (required)

Specifies the name of a string to be used as the prefix for variable names, which may exist in the environment. A colon (:) is used as a prefix separator, followed by a field name when the `acl:gather` tag is looking for variables in the environment.

fieldlist (required)

Specifies a comma-separated list of field names that should be gathered. Any field in the list that does not exist in the environment is cleared. If this parameter is not specified, then the `acl:gather` tag looks at all fields in the environment, if they are not found the existing value is maintained.

Description

This tag retrieves name/value pairs from the environment, and sets them into a loaded ACL. The name/value pairs are grouped together using a prefix and a prefix separator.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

Example

The following example gathers the `acl` name and description fields from the environment into `acl1` using the prefix `sample`:

```
<acl:gather name="ACL1"
            prefix="sample"
            fieldlist="aclname,description"/>
```

See Also

[acl:create](#)

[portal:logdep](#)

[object:stream](#)

[object:setobject](#)

[object:setparent](#)

[acl:scatter](#)

[acl:set](#)

acl:get

Copies a field from an ACL.

Syntax

```
<acl:get name="object name"
         field="field"
         [output="output"]/>
```

Parameters

name (required)

Name of the ACL.

field (required)

Name of the field to retrieve.

output (optional)

Specifies the name of the output variable, which should be created in the environment. If this parameter is not specified, the variable is given the same name as the field.

Description

This tag copies a field value from a loaded object into the environment.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

Example

The following example copies the name column into a variable called `name`:

```
<acl:get name="acl1"
        field="name"
        output="name"/>
```

See Also

[acl:create](#)

[acl:gather](#)

[object:stream](#)

[object:setobject](#)

[object:setparent](#)

[acl:scatter](#)

[acl:set](#)

acl:list

Retrieves a list of ACLs.

Syntax

```
<acl:list
    list="name of list"
    [order="field names"] />
```

Parameters

list (required)

Specifies the name of the list in the list pool. This is the same list pool used by other Content Server tags that generate lists. If a list exists with the same name, it is replaced by the new list.

order (optional)

Specifies a comma separated list of names that should be used to order the list of ACLs.

Description

This tag builds a list of objects that meet the specified criteria. The criteria are a list of name/value pairs, which must be present in the object. The implementing class must implement a list method. The list method should have one of the following prototypes:

- `public static IList List (ICS ics, FTValList aArgument, StringsOrder)`
- `public static IList List (ICS ics, FTValList aArgument, StringsOrder, String sArg1)`

errno

The possible values of errno include:

Value	Description
-101	No rows returned.
-105	Database error.
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing.
-10010	Invalid attribute value.

Example

The following example puts ACLs into list1:

```
<acl:list list="list1"/>
```

See Also

[acl:create](#)

[acl:gather](#)

[object:stream](#)

[object:setobject](#)

[object:setparent](#)

[acl:scatter](#)

[acl:set](#)

acl:load

Loads an ACL.

Syntax

```
<acl:load
    name="ACLname"
    acl="ACLname"/>
```

Parameters

name (required)
Name of the ACL in the object pool.

acl (required)
Name of the ACL in the database.

Description

This tag creates a new instance of the implementing class, loads a specific ACL into the ACL instance, and saves the ACL instance in the object pool using the specified name.

errno

The possible values of `errno` include:

Value	Description
-101	No rows returned.
-105	Database error.
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing.
-10006	The object ID is not valid.
-10007	The version of the object is not valid.
-10008	The node ID of the object is not valid.
-10010	Invalid attribute value.

Example

The following example loads ACL1 from the sample table.

```
<acl:load name="acl1"
    acl="SiteGod"/>
```

See Also

[acl:create](#)

[acl:gather](#)

[object:setobject](#)

[object:setparent](#)

[acl:scatter](#)

[acl:set](#)

acl:save

Saves an ACL.

Syntax

```
<acl:save
  name="ACL name"
  [flush="true|false"]/>
```

Parameters

name (required)

Name of the ACL to be saved.

flush (optional)

Specifies that upon success, the ACL should be flushed from the object pool. A `true` value indicates the ACL should be flushed. The default value is `false`.

Description

Saves a loaded ACL into storage. The ACL must have been previously created or loaded into the object pool. Saving an ACL either creates a new object in storage or updates an existing ACL in storage.

errno

The possible values of `errno` include:

Value	Description
-105	Database error.
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

Example

The following example saves `acl1` and removes it from memory.

```
<acl:save name="acl1"
  flush="true"/>
```

See Also

[acl:create](#)
[acl:delete](#)
[acl:gather](#)
[acl:get](#)
[acl:list](#)
[acl:load](#)
[acl:scatter](#)
[acl:set](#)

acl:scatter

Scatters a field from an ACL.

Syntax

```
<acl:scatter
    name="ACL name"
    [prefix="prefix"] />
```

Parameters

name (required)

Name of the ACL.

prefix (optional)

Specifies the the name of a string for the variable names that is created in the environment. The prefixes are separated from the field names with a colon (:) when creating variables in the environment.

Description

This command allows fields of a previously loaded ACL to be copied to the environment as variables with a specified prefix. This tag is the preferred alternative to using multiple `acl:get` tags.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

Example

The following example creates a variable in the environment with the prefix, sample.

```
<acl:scatter name="acl1"
            prefix="sample"/>
```

See Also

[acl:create](#)

[acl:delete](#)

[acl:gather](#)

[acl:get](#)

[acl:list](#)

[acl:load](#)

[acl:set](#)

acl:set

Sets a field in an ACL.

Syntax

```
<acl:set
    name="ACL name"
    field="field"
    value="value"/>
```

Parameters

name (required)

Name of ACL.

field (required)

Field to be set.

value (required)

Value to be set in the field.

Description

This tag sets the value of a field in a previously loaded ACL.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

Example

The following example sets the value to editor in the ac lname field of ac l1.

```
<acl:set name="acl1"
        field="aclname"
        value="editor"/>
```

See Also

[acl:gather](#)

[acl:get](#)

[acl:list](#)

[acl:load](#)

[acl:scatter](#)

dir:addattrs

Adds attributes to existing entries.

Syntax

```
<dir:addattrs name="entry">
  <dir:argument name="attribute name"
  <dir:argument value="attribute value"/>
</dir:addattrs>
```

Parameters

name (required)

Unique identifier for the name of the entry to which attributes are being added. The value is relative to the baseDN property.

dir:argument name (required)

Name of the attribute to be added to this entry.

dir:argument value (required)

Specifies the use of the CS.Property.dir.typeUser or CS.Property.dir.typeGroup properties. The property specified identifies the type of the object specified by the name parameter. This value is ignored if the arguments imply the distinguished name.

errno

The possible values of errno include:

Value	Description
-15004	Implementation doesn't support this operation
-15005	Unexpected naming exceptions from JNDI provider.
-15006	A required property missing, or class name specified for factory classes is in error.

Value	Description
-15007	Unexpected schema violation errors from JNDI provider.
-15008	General error in the syntax of an IName.
-15009	Unexpected invalid attribute exceptions from JNDI provider
-15010	Unexpected invalid attribute id exceptions from JNDI provider.
-15011	Unexpected invalid attribute value exceptions from JNDI provider.
-15012	User does not have permission to perform the operation.
-15014	Unexpected JNDI read error.
-15015	Unexpected JNDI write error.
-15016	JNDI attribute already in use.
-15023	Unexpected JNDI connection errors.
-15024	Required attribute missing.
-15025	Unknown error.
-15032	Add attribute SQL failed.
-15036	Missing or invalid child argument tags for tag.

Example

```
<dir:addattrs name="Variables.modname" />
  <dir:argument name="Variables.phoneFieldName"
    value="Variables.phone" />
</dir:addattrs>
```

Release Introduced

This tag was introduced in CSEE release 4.0.

dir:addgroupmember

Adds a member to a group

Syntax

```
<dir:addgroupmember
  name="group"
  member="member entry"/>
```

Parameters

name (required)

Unique identifier for the name of the entry to which members are being added. The value is relative to the baseDN property.

member (required)

The name of the entry to add as a member (usually, but not required to be, a user). The value is relative to the baseDN property, but is changed into the fully qualified distinguished name prior to adding as a value to the CS.Property.dir.uniquemember attribute.

errno

The possible values of errno include:

Value	Description
-15004	Implementation doesn't support this operation
-15005	Unexpected naming exceptions from JNDI provider.
-15006	A required property missing, or class name specified for factory classes is in error.
-15007	Unexpected schema violation errors from JNDI provider.
-15008	General error in the syntax of an IName.
-15009	Unexpected invalid attribute exceptions from JNDI provider
-15010	Unexpected invalid attribute id exceptions from JNDI provider.
-15011	Unexpected invalid attribute value exceptions from JNDI provider.
-15012	User does not have permission to perform the operation.
-15013	Node cannot be deleted because it has children.
-15014	Unexpected JNDI read error.
-15015	Unexpected JNDI write error.
-15016	JNDI attribute already in use.
15017	User is already a member.
-15020	User is not valid.
-15023	Unexpected JNDI connection errors.
-15024	Required attribute missing.
-15025	Unknown error.
-15026	Missing required create attribute.
-15027	User create SQL failure.
-15028	User delete SQL failure.
-15029	Cannot delete user attributes.
-15030	SQL error.
-15031	Query failed.

Value	Description
-15032	Add attribute SQL failed.
-15035	Database error occurred while attempting to assign the user to a group.
-15036	Missing or invalid child argument tags for tag.

Description

This tag adds a member to a group, but does not add a child to a group. The API implementation, not the client code, is responsible for checking that the member is a valid, existing distinguished name. If the specified entry is already a member, nothing is done.

In most SPIs, this entails adding the specified user's distinguished name to the set of values of the specified group's `uniquemember` attribute.

If the `CS.Property.dir.referentialIntegrity` property is turned on, the `CS.Property.dir.member` attribute (backlink) of the user is also be updated, if implemented by the SPI (e.g., the value in the property file is non-empty).

Example

```
<dir:addGroupmember name='<%=ics.GetVar("browser")%>'
                    member='<%=ics.GetVar("nametoadd")%>' />

<ics:if condition='<%=ics.GetErrno()==0%>'>
  <ics:then>
    <p><b> Successful adding group=<ics:getvar name='browser' />
      to user=<ics:getvar name='nametoadd' /> </b></p>
  </ics:then>
  <ics:else>
    <p><b>Could not add acl=<ics:getvar name="browser" /> to user,
      <ics:getvar name="nametoadd" />, error=<ics:geterrno /></p></b>
  </ics:else>
</ics:if>
```

Release Introduced

This tag was introduced in CSEE release 4.0.

dir:children

Retrieves the immediate child entries of the specified parent.

Syntax

```
<dir:children
    name="entry"
    list="children"/>
```

Parameters

name (required)

Unique identifier for the name of the entry to list. The value is relative to the `baseDN` property.

list (required)

The list in which to return the distinguished names of all children of the specified entry. The names returned are relative to the `baseDN` property.

errno

The possible values of `errno` include:

Value	Description
-15004	Implementation doesn't support this operation
-15005	Unexpected naming exceptions from JNDI provider.
-15006	A required property missing, or class name specified for factory classes is in error.
-15007	Unexpected schema violation errors from JNDI provider.
-15008	General error in the syntax of an <code>IName</code> .
-15009	Unexpected invalid attribute exceptions from JNDI provider
-15010	Unexpected invalid attribute id exceptions from JNDI provider.
-15011	Unexpected invalid attribute value exceptions from JNDI provider.
-15012	User does not have permission to perform the operation.
-15014	Unexpected JNDI read error.
-15015	Unexpected JNDI write error.
-15016	JNDI attribute already in use.
-15023	Unexpected JNDI connection errors.
-15024	Required attribute missing.
-15025	Unknown error.
-15036	Missing or invalid child argument tags for tag.

Example

```
<dir:children name="Variables.engineering"
list="engineeringDepartments"/>
```

Release Introduced

This tag was introduced in CSEE release 4.0.

dir:create

Creates new objects.

Syntax

```
<dir:create
    [name="ENTRY_name"]>
    [default="default property"]
    <[dir:argument name="ARG_name]" [value="ARG_value"]/>
</dir:create>
```

Parameters

name (optional)

Unique identifier for the name of the entry to create. The value is relative to the basedN property.

default (optional)

Set of attribute name/value pairs to be set on the entry. The preferred method is to specify the defaultPeopleAttrs.ini property, see the example for details.

dir:argument name (optional)

Name of the attribute to create for this entry.

dir:argument value (required)

Value of the named attribute.

Description

This tag creates a new object. The name of the object is implied from its attributes, specified as arguments.

errno

The possible values of errno include:

Value	Description
-15002	Name being created already exists.
-15004	Implementation doesn't support this operation
-15005	Unexpected naming exceptions from JNDI provider.
-15006	A required property missing, or class name specified for factory classes is in error.
-15007	Unexpected schema violation errors from JNDI provider.

Value	Description
-15008	General error in the syntax of an IName.
-15009	Unexpected invalid attribute exceptions from JNDI provider
-15010	Unexpected invalid attribute id exceptions from JNDI provider.
-15011	Unexpected invalid attribute value exceptions from JNDI provider.
-15012	User does not have permission to perform the operation.
-15014	Unexpected JNDI read error.
-15015	Unexpected JNDI write error.
-15016	JNDI attribute already in use.
-15023	Unexpected JNDI connection errors.
-15024	Required attribute missing.
-15025	Unknown error.
-15026	Missing required create attribute.
-15027	User create SQL failure.
-15036	Missing or invalid child argument tags for tag.

Example

This example contains the following custom .ini properties not found on your system:

- th.AttributeFirstName
- th.AttributeSN
- th.AttributeManager
- th.AttributeTitle
- th.AttributePhone

```
<dir:create name='<%= ics.GetVar("name") %>'
           defaults='<%=ics.GetProperty("defaultPeopleAttrs",
           "dir.ini", true)%>'>
<dir:argument name='<%=ics.GetProperty("username",
           "dir.ini", true)%>'
           value='<%=ics.GetVar("User1ID")%>' />
<dir:argument name='<%=ics.GetProperty("password",
           "dir.ini", true)%>'
           value='<%=ics.GetVar("User1PW")%>' />
<dir:argument name='<%=ics.GetProperty("th.AttributeFirstName",
           "dir.ini", true)%>'
           value='<%=ics.GetVar("User1FirstName")%>' />
<dir:argument name='<%=ics.GetProperty("cn",
           "dir.ini", true)%>'
           value='<%=ics.GetVar("User1FirstName")+"
```

```

        "+ics.GetVar("User1LastName")%>' />
<dir:argument name='<%=ics.GetProperty("th.AttributeSN",
"dir.ini",
        true)%>'
        value='<%=ics.GetVar("User1LastName")%>' />
<dir:argument name='<%=ics.GetProperty("th.AttributeManager",
"dir.ini", true)%>'
        value='<%=ics.GetVar("Manager1")%>' />
<dir:argument name='<%=ics.GetProperty("th.AttributeTitle",
"dir.ini",
        true)%>'?value='<%=ics.GetVar("Title1")%>' />
<dir:argument name='<%=ics.GetProperty("th.AttributePhone",
"dir.ini",
        true)%>'
        value='<%=ics.GetVar("User1phone")%>' />
</dir:create>
<ics:if condition='<%=ics.GetErrno()==0%>'>
<ics:then>
<p><b> Successful Creating User=<ics:getvar
                        name='name' /> </b></p>
</ics:then>
<ics:else>
<p><b>Could not create user, <ics:getvar
                        name='name' />, error=i<ics:geterrno/></p></b>
</ics:else>
</ics:if>

```

Release Introduced

This tag was introduced in CSEE release 4.0.

dir:delete

Deletes a named object from the namespace.

Syntax

```

<dir:delete
    name="entry name"/>

```

Parameters

name (required)

Unique identifier for the name of the entry to delete. The value is relative to the basedN property.

errno

The possible values of errno include:

Value	Description
-15003	Name being deleted does not exist.

Value	Description
-15004	Implementation doesn't support this operation
-15005	Unexpected naming exceptions from JNDI provider.
-15006	A required property missing, or class name specified for factory classes is in error.
-15007	Unexpected schema violation errors from JNDI provider.
-15008	General error in the syntax of an IName.
-15009	Unexpected invalid attribute exceptions from JNDI provider
-15010	Unexpected invalid attribute id exceptions from JNDI provider.
-15011	Unexpected invalid attribute value exceptions from JNDI provider.
-15012	User does not have permission to perform the operation.
-15013	Node cannot be deleted because it has children.
-15014	Unexpected JNDI read error.
-15015	Unexpected JNDI write error.
-15016	JNDI attribute already in use.
15017	User is already a member.
-15018	User is not a member.
-15020	User is not valid.
-15023	Unexpected JNDI connection errors.
-15024	Required attribute missing.
-15025	Unknown error.
-15026	Missing required create attribute.
-15028	User delete SQL failure.
-15029	Cannot delete user attributes.
-15030	SQL error.
-15031	Query failed.
-15034	Database error occurred while attempting to delete an attribute.
-15036	Missing or invalid child argument tags for tag.

Example

```
<dir:delete name='<%=ics.GetVar("name")%>' />
<ics:if condition='<%=ics.GetErrno()==0%>'>
  <ics:then>
```

```

        <p><b> Successful Deleting User=<ics:getvar=name='name' />
        </b></p>
      </ics:then>
      <ics:else>
        <p><b>
          Failed to delete user,
          <ics:getvar name='name' />, error='<ics:geterrno/></p></
b>
      </ics:else>
    </ics:if>

```

Release Introduced

This tag was introduced in CSEE release 4.0.

dir:getattrs

Retrieves the attributes for a specified entry.

Syntax

```

<dir:getattrs
  list="attribute list"
  name="entry"
  [attrs="attribute IDs"]/>

```

Parameters

list (required)

Two-column list in which the attribute values are returned. Columns are name and value. Multi-valued attributes have as many rows in the list as values. The names returned are relative to the basedN property.

name (required)

Unique identifier for the name of the entry from which attributes are being retrieved. The value is relative to the basedN property.

attrs (optional)

A StringList of the attributes to retrieve. The default is to retrieve all attributes. If the attribute is specified, but the StringList is empty, no attributes are retrieved.

errno

The possible values of errno include:

errno	Description
-15004	Implementation doesn't support this operation
-15005	Unexpected naming exceptions from JNDI provider.
-15006	A required property missing, or class name specified for factory classes is in error.
-15007	Unexpected schema violation errors from JNDI provider.

errno	Description
-15008	General error in the syntax of an IName.
-15009	Unexpected invalid attribute exceptions from JNDI provider
-15010	Unexpected invalid attribute id exceptions from JNDI provider.
-15011	Unexpected invalid attribute value exceptions from JNDI provider.
-15012	User does not have permission to perform the operation.
-15014	Unexpected JNDI read error.
-15015	Unexpected JNDI write error.
-15016	JNDI attribute already in use.
-15023	Unexpected JNDI connection errors.
-15024	Required attribute missing.
-15025	Unknown error.
-15036	Missing or invalid child argument tags for tag.

Example

```
<dir:getattrs list="userattributes"
              name='<%=ics.GetVar("nametoget")%>' />
<p><b>FETCHING ATTRIBUTES FOR USER=<ics:getvar
name='nametoget' /></b></p>
  <table border=1 cellpadding=2 cellspacing=0>
    <tr bgcolor=blue><td><font face=tahoma color=white
size=+1><b>ATTRIBUTE NAME</font></td>
    <td><font face=tahoma color=white size=+1><b>ATTRIBUTE
VALUE</font></td></tr>
<ics:listloop listname="userattributes">
  <tr><td><font face=tahoma size=-1><ics:listget
listname="userattributes" fieldname="NAME"/></
font></td>
    <td><font face=tahoma size=-1><ics:listget
listname="userattributes" fieldname="VALUE"/></
font></td></tr>
</ics:listloop>
</table>
```

Release Introduced

This tag was introduced in CSEE release 4.0.

dir:groupmembers

Lists all members of the specified group.

Syntax

```
<dir:groupmembers
    name="group"
    list="members"/>
```

Parameters

name (required)

Unique identifier for the name of the entry for which members are being listed. The value is relative to the baseDN property

list (required)

A list of distinguished names of all the entries belonging to this group. The list has a single column, name. The names returned are relative to the baseDN property.

errno

The possible values of errno include:

Value	Description
-15004	Implementation doesn't support this operation
-15005	Unexpected naming exceptions from JNDI provider.
-15006	A required property missing, or class name specified for factory classes is in error.
-15007	Unexpected schema violation errors from JNDI provider.
-15008	General error in the syntax of an IName.
-15009	Unexpected invalid attribute exceptions from JNDI provider
-15010	Unexpected invalid attribute id exceptions from JNDI provider.
-15011	Unexpected invalid attribute value exceptions from JNDI provider.
-15012	User does not have permission to perform the operation.
-15014	Unexpected JNDI read error.
-15015	Unexpected JNDI write error.
-15016	JNDI attribute already in use.
15017	User is already a member.
-15018	User is not a member.
-15020	User is not valid.
-15023	Unexpected JNDI connection errors.

Value	Description
-15024	Required attribute missing.
-15025	Unknown error.
-15030	SQL error.
-15031	Query failed.
-15036	Missing or invalid child argument tags for tag.

Example

```
<dir:groupmembers name="Variables.groupName" list="members"/>
```

Release Introduced

This tag was introduced in CSEE release 4.0.

dir:groupmemberships

Lists all groups to which a specified entry belongs.

Syntax

```
<dir:groupmemberships
  name="ENTRY"
  list="GROUPS"/>
```

Parameters

name (required)

Unique identifier for the name of the entry for which to find all memberships. The value is relative to the `baseDN` property.

list (required)

A list of distinguished names of all the groups to which this entry has membership. The list has a single column, name. The names returned are relative to the `baseDN` property.

errno

The possible values of `errno` include:

Value	Description
-15004	Implementation doesn't support this operation
-15005	Unexpected naming exceptions from JNDI provider.
-15006	A required property missing, or class name specified for factory classes is in error.
-15007	Unexpected schema violation errors from JNDI provider.

Value	Description
-15008	General error in the syntax of an IName.
-15009	Unexpected invalid attribute exceptions from JNDI provider
-15010	Unexpected invalid attribute id exceptions from JNDI provider.
-15011	Unexpected invalid attribute value exceptions from JNDI provider.
-15012	User does not have permission to perform the operation.
-15014	Unexpected JNDI read error.
-15015	Unexpected JNDI write error.
-15016	JNDI attribute already in use.
15017	User is already a member.
-15018	User is not a member.
-15020	User is not valid.
-15023	Unexpected JNDI connection errors.
-15024	Required attribute missing.
-15025	Unknown error.
-15030	SQL error.
-15031	Query failed.
-15035	Database error occurred while attempting to assign the user to a group.
-15036	Missing or invalid child argument tags for tag.

Example

```
<dir:groupmemberships name="Variables.name" list="groups"/>
```

Release Introduced

This tag was introduced in CSEE release 4.0.

dir:listusers

Returns a list of all ancestors to the property `CS.Property.dir.peopleParent`.

Syntax

```
<dir:listusers
    list="user list"
    [commonancestor="users parent"]>
```

Parameters

list (required)

List in which to return the distinguished names of all descendants found. The names returned are relative to the `baseDN` property.

commonancestor (optional)

Specifies the root of the Content Server user tree. If this parameter is not specified the default is the value specified in the `CS.Property.dir.people.Parent` property.

errno

The possible values of `errno` include:

Value	Description
-15004	Implementation doesn't support this operation
-15005	Unexpected naming exceptions from JNDI provider.
-15006	A required property missing, or class name specified for factory classes is in error.
-15007	Unexpected schema violation errors from JNDI provider.
-15008	General error in the syntax of an <code>IName</code> .
-15009	Unexpected invalid attribute exceptions from JNDI provider
-15010	Unexpected invalid attribute id exceptions from JNDI provider.
-15011	Unexpected invalid attribute value exceptions from JNDI provider.
-15012	User does not have permission to perform the operation.
-15014	Unexpected JNDI read error.
-15015	Unexpected JNDI write error.
-15016	JNDI attribute already in use.
15017	User is already a member.
-15018	User is not a member.
-15020	User is not valid.
-15023	Unexpected JNDI connection errors.

Value	Description
-15024	Required attribute missing.
-15025	Unknown error.
-15030	SQL error.
-15031	Query failed.
-15034	Database error occurred while attempting to delete an attribute.
-15036	Missing or invalid child argument tags for tag.

Description

The parameter `commonancestor` may be specified to override the `CS.Property.dir.peopleParent` value. The same results may be achieved using the `dir.children` tag. The `dir.listusers` tag is provided as a convenience for user operations.

Example

```
<dir:listusers list="alluserlist"/>
  <ics:if condition='<%=ics.GetErrno()==0%>'>
    <ics:then>
      <p><b><ics:listget listname="alluserlist"
        fieldname="#numRows"/> USER IN DN=
        <%=ics.GetProperty("peopleparent",
          "dir.ini", true) %></b></p>
      <table border=1 cellpadding=2 cellspacing=0>
        <tr bgcolor=blue><td><font face=tahoma color=white
          size=+1><b>USER DN's</font></td></tr>
      <ics:listloop listname="alluserlist">
        <tr><td><font face=tahoma size=-1><ics:listget
          listname="alluserlist" fieldname="NAME"/></font></td></tr>
      </ics:listloop>
    </table>
    </ics:then>
    <ics:else>
      <p><b>No Users Found! Error=<ics:geterrno/></p>
    </ics:else>
  </ics:if>
```

Release Introduced

This tag was introduced in CSEE release 4.0.

dir:removeattrs

Deletes the specified attribute from the specified entry.

Syntax

```
<dir:removeattrs name="entry">
  <dir:argument name="attribute name" value="attribute value"/>
</dir:removeattrs>
```

Parameters

name (required)

Unique identifier for the name of the entry from which attributes are being removed.
The value is relative to the `baseDN` property.

dir:argument name (required)

Name of the attribute to be removed from this entry.

dir:argument value (required)

Value of the named attribute. If this argument is not specified, all values are removed.

errno

The possible values of `errno` include:

Value	Description
-15003	Name being deleted does not exist.
-15004	Implementation doesn't support this operation
-15005	Unexpected naming exceptions from JNDI provider.
-15006	A required property missing, or class name specified for factory classes is in error.
-15007	Unexpected schema violation errors from JNDI provider.
-15008	General error in the syntax of an <code>IName</code> .
-15009	Unexpected invalid attribute exceptions from JNDI provider
-15010	Unexpected invalid attribute id exceptions from JNDI provider.
-15011	Unexpected invalid attribute value exceptions from JNDI provider.
-15012	User does not have permission to perform the operation.
-15013	Node cannot be deleted because it has children.
-15014	Unexpected JNDI read error.
-15015	Unexpected JNDI write error.
-15016	JNDI attribute already in use.
-15018	User is not a member.

Value	Description
-15020	User is not valid.
-15023	Unexpected JNDI connection errors.
-15024	Required attribute missing.
-15025	Unknown error.
-15028	User delete SQL failure.
-15029	Cannot delete user attributes.
-15030	SQL error.
-15031	Query failed.
-15034	Database error occurred while attempting to delete an attribute.
-15036	Missing or invalid child argument tags for tag.

Description

This tag removes the specified attributes from the selected entries. If you are using this tag with the `CS.Property.dir.referentialIntegrity` property turned on, and if the requested operation requires a change in the distinguished name (as in, modifying the first `commonname` attribute value), then the operation changes the following attributes:

- `CS.Property.dir.member` attribute on all users belonging to any group being updated
- `CS.Property.dir.uniquemember` on all groups containing a user being updated
- `CS.Property.dir.owner` attribute containing the name of a user being updated
- `CS.Property.dir.seeAlso` attribute containing the name of anything being updated

An update is not be attempted on any attribute for which the above property value is an empty string, regardless of the value of the `referentialIntegrity` property. Do not enable referential integrity if it is configurable in the SPI's directory server itself.

Note

Removing a `commonname` attribute may have the side effect of changing the distinguished name.

Example

```
<dir:removeattrs name="Variables.modname">
  <dir:argument name="Variables.phoneFieldName"/>
</dir:removeattrs>
```

Release Introduced

This tag was introduced in CSEE release 4.0.

dir:removegroupmember

Removes a member from a group.

Syntax

```
<dir:removegroupmember
    name="group"
    member="member entry"/>
```

Parameters

name (required)

Unique identifier for the name of the entry from which members are being removed. The value is relative to the `baseDN` property.

member (required)

Name of the entry to remove as a member (usually, but not required to be, a user). The value is relative to the `baseDN` property, but is changed into the fully qualified distinguished name prior to searching for the matching `CS.Property.dir.uniquemember` attribute value.

errno

The possible values of `errno` include:

Value	Description
-15003	Name being deleted does not exist.
-15004	Implementation doesn't support this operation
-15005	Unexpected naming exceptions from JNDI provider.
-15006	A required property missing, or class name specified for factory classes is in error.
-15007	Unexpected schema violation errors from JNDI provider.
-15008	General error in the syntax of an <code>IName</code> .
-15009	Unexpected invalid attribute exceptions from JNDI provider
-15010	Unexpected invalid attribute id exceptions from JNDI provider.
-15011	Unexpected invalid attribute value exceptions from JNDI provider.
-15012	User does not have permission to perform the operation.
-15013	Node cannot be deleted because it has children.
-15014	Unexpected JNDI read error.
-15015	Unexpected JNDI write error.
-15016	JNDI attribute already in use.
-15018	User is not a member.

Value	Description
-15020	User is not valid.
-15023	Unexpected JNDI connection errors.
-15024	Required attribute missing.
-15025	Unknown error.
-15026	Missing required create attribute.
-15028	User delete SQL failure.
-15029	Cannot delete user attributes.
-15030	SQL error.
-15031	Query failed.
-15034	Database error occurred while attempting to delete an attribute.
-15036	Missing or invalid child argument tags for tag.

Description

This tag removes a member from a group, but does not remove a child from a group. In most SPIs, this entails removing the specified user's distinguished name from the set of values of the specified group's `uniquemember` attribute.

If `CS.Property.dir.referentialIntegrity` is turned on, the `CS.Property.dir.member` attribute (backlink) of the user is also updated, if implemented by the SPI (e.g. the value in the property file is non-empty).

Example

```
<dir:removegroupmember name="Variables.groupName"
                        member="Variables.userName"/>
```

Release Introduced

This tag was introduced in CSEE release 4.0.

dir:replaceattrs

Replaces the existing attribute values with the specified values.

Syntax

```
<dir:replaceattrs name="entry">
  <dir:argument name="attribute name" value="attribute value"/>
</dir:replaceattrs>
```

Parameters

name (required)

Unique identifier for the name of the entry to which attributes are being replaced. The value is relative to the baseDN property.

dir:argument name (required)

Name of the attribute to be replaced for this entry.

dir:argument value (required)

Value of the named attribute.

errno

The possible values of errno include:

Value	Description
-15002	Name being created already exists.
-15004	Implementation doesn't support this operation
-15005	Unexpected naming exceptions from JNDI provider.
-15006	A required property missing, or class name specified for factory classes is in error.
-15007	Unexpected schema violation errors from JNDI provider.
-15008	General error in the syntax of an IName.
-15009	Unexpected invalid attribute exceptions from JNDI provider
-15010	Unexpected invalid attribute id exceptions from JNDI provider.
-15011	Unexpected invalid attribute value exceptions from JNDI provider.
-15012	User does not have permission to perform the operation.
-15014	Unexpected JNDI read error.
-15015	Unexpected JNDI write error.
-15016	JNDI attribute already in use.
15017	User is already a member.
-15018	User is not a member.

Value	Description
-15020	User is not valid.
-15023	Unexpected JNDI connection errors.
-15024	Required attribute missing.
-15025	Unknown error.
-15026	Missing required create attribute.
-15027	User create SQL failure.
-15029	Cannot delete user attributes.
-15030	SQL error.
-15031	Query failed.
-15032	Add attribute SQL failed.
-15033	Database error handling new attribute.
-15034	Database error occurred while attempting to delete an attribute.
-15036	Missing or invalid child argument tags for tag.

Description

This tag replaces existing attribute values with specified values. If the `CS.Property.dir.referentialIntegrity` property is turned on, and if the requested operation requires a change in the distinguished name (as in, modifying the first `commonname` attribute value), then the operation changes to the following attributes:

- `CS.Property.dir.member` attribute on all users belonging to any group being updated
- `CS.Property.dir.uniquemember` on all groups containing a user being updated
- `CS.Property.dir.owner` attribute containing the name of a user being updated
- `CS.Property.dir.seeAlso` attribute containing the name of anything being updated

An update is not attempted on any attribute for which the preceding property value is an empty string, regardless of the value of the `referentialIntegrity` property. Do not enable referential integrity if it is configurable in the SPI's directory server itself.

Example

```
<dir:replaceattrs name="Variables.modname"/>
  <dir:argument name="Variables.phFieldName"
value="Variables.newph"/>
</dir:replaceattrs>
```

Release Introduced

This tag was introduced in CSEE release 4.0.

dir:search

Searches for entries matching the specified attributes.

Syntax

```
<dir:search
    list="matches"
    context="common ancestor"
    [operator=(and|or)]>
  <dir:argument name="attribute name" value="attribute value"/>
</dir:search>
```

Parameters

list (required)

List in which the distinguished names of the matching entries are returned. The list contains a single column called NAME.

context (required)

Overrides the value of the parent in the property file
CS.Property.dir.peopleParent for type=user. Specify this argument for searching under groups, or under a branch of the user tree.

operator (optional)

Determines if the matching entries must contain all and or one or more of the specified attribute/value pairs. The default is and.

dir:argument name (required)

The name of the attribute to be matched.

dir:argument value (required)

Value of the attribute to be matched.

errno

The possible values of errno include:

Value	Description
-15004	Implementation doesn't support this operation
-15005	Unexpected naming exceptions from JNDI provider.
-15006	A required property missing, or class name specified for factory classes is in error.
-15007	Unexpected schema violation errors from JNDI provider.
-15008	General error in the syntax of an IName.
-15009	Unexpected invalid attribute exceptions from JNDI provider
-15010	Unexpected invalid attribute id exceptions from JNDI provider.
-15011	Unexpected invalid attribute value exceptions from JNDI provider.
-15012	User does not have permission to perform the operation.

Value	Description
-15014	Unexpected JNDI read error.
-15015	Unexpected JNDI write error.
-15016	JNDI attribute already in use.
-15023	Unexpected JNDI connection errors.
-15024	Required attribute missing.
-15025	Unknown error.
-15030	SQL error.
-15031	Query failed.
-15036	Missing or invalid child argument tags for tag.

Example

```

<dir:search list='searchlist' operator="OR"
            context='<%=ics.GetProperty("peopleparent", "dir.ini",
            true) %>'>
<dir:argument name='<%=ics.GetProperty("th.AttributeManager",
            "dir.ini", true)%>'
            value='<%=ics.GetVar("SearchValue")%>' />
<dir:argument name='<%=ics.GetProperty("th.AttributeManager",
            "dir.ini", true)%>'
            value='<%=ics.GetVar("SearchValue2")%>' />
</dir:search>
    <ics:if condition='<%=ics.GetErrno()==0%>'>
    <ics:then>
    <p><b>SEARCH REQUEST FOUND <ics:listget
        listname="searchlist" fieldname="#numRows"/> MATCHES FOR
<ics:getvar name="SearchValue"/> and <ics:getvar
name="SearchValue2"/></b></p>
    <table border=1 cellpadding=2 cellspacing=0>
    <tr bgcolor=blue><td><font face=tahoma color=white
        size=+1><b>USER DN's</font></td></tr>
<ics:listloop listname="searchlist">
    <tr><td><font face=tahoma size=-1><ics:listget
        listname="searchlist" fieldname="NAME"/></font></
td></tr>
</ics:listloop>
</table>
</ics:then>
<ics:else>
<p><b>SEARCH FOR <ics:getvar name="SearchValue"/>
Failed!</b> No Matches Found!
Error=<ics:geterrno/></p>
</ics:else>
</ics:if>

```

Release Introduced

This tag was introduced in CSEE release 4.0.

ftcs1_0:flush

Flushes the body content of the `ftcs1_0:ftcs` tag to the servlet output stream.

Syntax

```
<ftcs1_0:flush/>
```

Parameters

None.

Description

This tag flushes the body content of the `ftcs1_0:ftcs` tag to the servlet output stream.

errno

None.

See Also

[ftcs1_0:ftcs](#)

ftcs1_0:ftcs

Root tag for Content Server : jsp elements.

Syntax

```
<ftcs1_0:ftcs
    [path="path"]
    [ini="ini file"]
    [name="name"]/>
```

Parameters

`path` (optional)

Path to the `futuretense.ini` property file. Required when using JSP tags in a JSP file not being served as a Content Server element.

`ini` (optional)

Name of the `.ini` file to be used.

`name` (optional)

Name of the element.

errno

None.

See Also

[ics:indexadd](#)

ics:argument

Declares a value that is available to the parent tag.

Syntax

```
<ics:argument
  name="argument name"
  value="argument value"/>
```

Parameters

name (required)

Name of the argument.

value (required)

Value for this argument.

Description

The `ics:argument` tag declares a value that is available to the parent tag. It is used in the context of another tag, and causes a runtime error if it is not contained within a proper parent tag. Depending on the parent tag used, the following are variants of the `ics:argument` tag:

- `ics:dateargument`
- `ics:textargument`
- `ics:fieldargument`

errno

The value of `errno` is unchanged.

Example

The following example uses four `ics:argument` tags to specify name/value pairs to an `ics:catalogmanager` command:

```
<ics:catalogmanager>
  <ics:argument name="ftcmd" value="addrow"/>
  <ics:argument name="tablename" value="HorrorBooks"/>
  <ics:argument name="title" value="The Shining"/>
  <ics:argument name="author" value="Stephen King"/>
</ics:catalogmanager>
```

See Also

[ics:dateargument](#)

[ics:fieldargument](#)

[object:track](#)

[object:untrack](#)

ics:callelement

Processes the content of an element.

Syntax

```
<ics:callelement element="element name">
    <ics:argument name="argument name" value="arg value"/>
</ics:callelement>
```

Parameters

element (required)
Name of the element to call.

Description

The `ics:callelement` tag processes the content of an element. The element must exist in the `ElementCatalog`.

Variables that exist prior to `ics:callelement` are available to tags within the called element. Variables set within the called element are available in the parent element after the `ics:callelement` tag. Variables can also be set using the `ics:argument` tag within the `ics:callelement` tag.

Variables for the `ics:callelement` tag are subject to an order of precedence. The order of precedence is relative to the location of the variable. The following list describes the order of precedence:

1. Variables set in arguments
2. Variables set in scripts
3. **resdetails1** and **resdetails2** of the `ElementCatalog`

In the following element, the `arg=3` variable is set in the **resdetails1** column in the `ElementCatalog`. The `arg` value with the highest order of precedence is `value1`.

```
<ics:setvar name="arg" value="value2"/>
...
<ics:callelement element="element name">
    <ics:argument name="arg" value="value1"/>
</ics:callelement>
```

In the following element, the `arg=value3` is set in the **resdetails1** column in the `ElementCatalog`. The `arg` value with the highest order of precedence is `value2`.

```
<ics:setvar name="arg" value="value2"/>
...
<ics:callelement element="element name">
</ics:callelement>
```

In the following element, the `arg=value3` is set in the **resdetails1** column of the `ElementCatalog`. The `arg` value with the highest order of precedence is `value3`; this is occurs when a variable is not defined in scope before the `ics:callelement` tag.

```
<ics:callelement name="element name">
```

```
</ics:callelement>
```

errno

The possible values of errno include:

Value	Description
-10	No element exists.
-12	Cannot evaluate element.

Example

The following example calls the element SimpleTest with the arguments Foo and Foo2.

```
<ics:clearerrno/>
<ics:callelement element="SimpleTest">
  <ics:argument name="Foo" value="Bar"/>
  <ics:argument name="Foo2" value="Bar2"/>
</ics:callelement>
<ics:if condition='<%=ics.GetErrno()!=0%>'>
  <ics:then>
    Problem running element SimpleTest
  </ics:then>
</ics:if>
```

See Also

[ics:argument](#)

ics:callsql

Retrieves and executes a SQL query stored in the SystemSQL table.

Syntax

```
<ics:callsql
  qryname="query name"
  listname="list"
  [limit="maximum number of results"]/>
```

Parameters

qryname (required)

Name of query to retrieve and execute.

listname (required)

Name of list to contain resultset.

limit (optional)

Maximum number of rows in the resultset. Note that there is no limit if the parameters are missing or set to a negative number.

Description

The `ics:callsql` tag retrieves and executes an SQL query stored in the `SystemSQL` table. The resultset of the query is cached against the table name specified for the query in the `SystemSQL` table. The table name must exist in the database prior to calling `ics:callsql`.

Note

The `deftable` column in the `SystemSQL` table can specify a comma-separated list of table names.

errno

The possible values of `errno` include:

Value	Description
-15	No file found.
-101	No rows found.
-102	Empty column.
-103	No such table.
-104	No table definition.

Example

```
<ics:clearerrno/>
<ics:callsql qryname="gettables" listname="newsrs"/>
<ics:if condition="<%=ics.GetErrno() == 0%>">
  <ics:then>
    <ics:listloop listname="newsrs" >
      Tablename: <ics:listget listname="newsrs" fieldname="tblname"><br>
      Directory: <ics:listget listname="newsrs" fieldname="defdir"/><br>
    </ics:listloop>
  </ics:then>
  <ics:else>
    report error
  </ics:else>
</ics:if>
```

See Also

[ics:sql](#)

[ics:selectto](#)

ics:catalogbatch

Allows [ics:catalogmanager](#) calls to be used in groups.

Syntax

```
<ics:catalogbatch/>
```

Description

This tag combines commands so that they act as a group. All [catalogmanager](#) tags within [catalogbatch](#) are committed to the database only if all the commands complete successfully. Only the [addrow](#) and [addrows](#) [catalogmanager](#) commands are supported within the [ics:catalogbatch](#) tag.

errno

The value of `errno` is unchanged.

Example

In the following example the data is added to the `StoreHours` table and the `MyFood` table only if there are no errors inserting into either table. Any error causes the transaction to rollback.

```
<ics:catalogbatch>
<ics:catalogmanager>
  <ics:argument name="ftcmd" value="addrows"/>
  <ics:argument name="tablename" value='MyFood' />
  <ics:argument name="id0" value="0"/>
  <ics:argument name="fruit0" value="kiwi"/>
  <ics:argument name="vegetable0" value="asparagus"/>
  <ics:argument name="bread0" value="rye"/>
  <ics:argument name="meat0" value="chicken breast"/>
  <ics:argument name="id1" value="1"/>
  <ics:argument name="fruit1" value="banana"/>
  <ics:argument name="vegetable1" value="peas"/>
  <ics:argument name="bread1" value="sourdough"/>
  <ics:argument name="meat1" value="filet mignon"/>
</ics:catalogmanager>

<ics:catalogmanager>
  <ics:argument name="ftcmd" value="addrow"/>
  <ics:argument name="tablename" value="StoreHours"/>
  <ics:argument name="id" value="332"/>
  <ics:argument name="description" value="Open 24 hours"/>
</ics:catalogmanager>
</ics:catalogbatch>
```

ics:catalogdef

Queries a table for its column definition and stores the results in a list.

Syntax

```
<ics:catalogdef
    listname="some list name"
    table="some table name"/>
```

Parameters

listname (required)

List to contain column information for the table.

table (required)

Name of the table.

Description

The `ics:catalogdef` tag queries a table for its column definition and stores the results in a list.

The list contains a row for each column in the table. The **KEY** and **FOREIGN** values are identical for each row in the resultset. The column names are as follows:

Column Name	Meaning
KEY	Name of table's primary key (unknown if table is foreign).
FOREIGN	If table is foreign, value is <code>true</code> . Otherwise, it is <code>false</code> .
COL	Name of the column in the table.
COLTYPE	The values for COLTYPE can be: • <code>text</code> • <code>integer</code> • <code>double binary</code> • <code>date/time, upload</code> • <code>unknown</code>
COLSIZE	Size of the column based on database information.

errno

The possible values of `errno` include:

Value	Description
-101	No rows found.
-102	Empty column.
-103	No such table.

Value	Description
-104	No table definition.

Example

The following example displays the table definition of a table called movies:

```
<ics:clearerrno/>
<ics:catalogdef list="table" from="movies"/>
Key: <ics:listget listname="table" fieldname="KEY"/> <br>
Foreign: <ics:listget listname="table" fieldname="FOREIGN"/> <br>
<ics:listloop listname="table" >
  Colname: <ics:listget listname="table" fieldname="COL"/>
    <br>
  Coltype: <ics:listget listname="table" fieldname="COLTYPE"/>
    <br>
  Colsize: <ics:listget listname="table" fieldname="COLSIZE"/>
    <br>
</ics:listloop>
```

See Also

[ics:listget](#)

[ics:sql](#)

ics:catalogmanager

Creates, modifies, or deletes tables and rows.

Syntax

```
<ics:catalogmanager/>
```

Description

The `<ics:catalogmanager/>` tag invokes the CatalogManager module. That module creates, modifies, or deletes tables and rows. The `ics:catalogmanager` tag is an alternative to SQL-based tags, such as `<ics:callsql>`.

The following is a list of the commands that you use with `ics:catalogmanager`.

- [addrow](#)
- [addrows](#)
- [addrowtext](#)
- [addUser \(Deprecated\)](#)
- [commit](#)
- [createtable](#)
- [deletelog](#)
- [deleterevision](#)

- [deleterow](#)
- [deleterows](#)
- [delUser \(Deprecated\)](#)
- [deletetable](#)
- [editrow](#)
- [editrows](#)
- [exportform](#)
- [exportlog](#)
- [flushcatalog](#)
- [flushpage](#)
- [history](#)
- [lock](#)
- [login](#)
- [logout](#)
- [mirrorabort](#)
- [mirrorrows](#)
- [release](#)
- [replacerow](#)
- [replacerows](#)
- [rollback](#)
- [release](#)
- [selectrows](#)
- [tracktable](#)
- [unlockrecord](#)
- [untracktable](#)
- [updaterow](#)
- [updaterows](#)
- [updaterow2](#)
- [updaterows2](#)

addrow

Adds a row to a table using `ics:catalogmanager`.

Syntax

```
<ics:catalogmanager>
  <ics:argument name="ftcmd" value="addrow"/>
  <ics:argument name="tablename" value="table"/>
```

```
[<ics:argument name="urlcolumnname_folder" value="folder name"/>]
[<ics:argument name="columnname_file" value="file name"/>]
[<ics:argument name="columnname" value="some column name"/>]
</ics:catalogmanager>
```

Parameters

ftcmd (required)

Value must be set to addrow.

tablename (required)

Name of the table where you plan to add a row.

urlcolumnname_folder (optional)

Subfolder name to store the uploaded file. The file is then stored under the upload folder and the subfolder. The upload folder is specified in the SystemInfo table's `defdir` column. You can have multiple upload columns.

Upload columns are designated by the prefix `url`. The column name in the table must begin with `url` (for example, `urltext`). The argument name must include the table's column name with `_folder` appended to the column name (for example, `urltext_folder`).

columnname_file (required for non-binary files)

Name of the file you want to upload. The column name in the table must begin with `url` (for example, `urltext`). The argument name must have `_file` appended to the table's column name (for example, `urltext_file`).

columnname (optional)

Value of each column in the row to be added. The `columnname` is a table column name.

Description

The `addrow` command adds a row to a table.

Note

When adding a row to a table of type `obj` (an object table), the system creates `Variables.newid`, which contains the system-assigned unique ID for the new record.

errno

The possible values of `errno` include:

Value	Description
-103	No such table.
-104	No table definition.
-105	Database error.

See Also

The following ics:catalogmanager commands:

[addrows](#)

[addrowtext](#)

addrows

Adds multiple rows to a table using ics:catalogmanager.

Syntax

```
<ics:catalogmanager>
  <ics:argument name="ftcmd" value="addrows"/>
  <ics:argument name="tablename" value="table"/>
  [<ics:argument name="column_folderN" value="folder name"/>]
  [<ics:argument name="columnnameN" value="column name"/>]
  [<ics:argument name="columnname_fileN" value="file name"/>]
</ics:catalogmanager>
```

Parameters

ftcmd (Required)

Value must be set to addrows.

tablename (Required)

Table name to add rows.

columnname_folderN (Optional)

Subfolder name to store the uploaded file. The file is then stored under the upload folder and the subfolder. The upload folder is specified in the SystemInfo table's defdir column. You can have multiple upload columns.

Upload columns are designated by the prefix url. The column name in the table must begin with url (for example, urltext). The argument name must include the table's column name with _folder appended to the column name (for example, urltext_folder).

columnnameN (Required for non-binary files)

Name of the file you want to upload. The columnname is a table column name. The column name in the table must begin with url (for example, urltext). The argument name must include the table's column name with _folder appended to the column name (for example, urltext_folder).

columnname_fileN (required for non-binary files)

Name of the file you want to upload. The column name in the table must begin with url (for example, urltext). The argument name must have _file appended to the table's column name (for example, urltext_file).

errno

The possible values of errno include:

Value	Description
-103	No such table.
-104	No table definition.

Value	Description
-105	Database error.

See Also

The following `ics:catalogmanager` commands:

[addrow](#)

[addrowtext](#)

addrowtext

Adds a row to a table using `ics:catalogmanager`.

Syntax

```
<ics:catalogmanager>
  <ics:argument name="ftcmd" value="addrowtext"/>
  <ics:argument name="tablename" value="table"/>
  [urlcolumnname_folder="folder name"/>
  [columnname_file="file name"/>
  [columnname="column name"/>
</ics:catalogmanager>
```

Parameters

ftcmd (required)

Value must be set to `addrowtext`.

tablename (required)

Name of the table to which to add text.

urlcolumnname_folder (optional)

For every file upload column (column name starts with `url`), you can use a parameter named `urlcolumnname_folder` to store the file in a subfolder under the default upload folder for the table.

columnname_file (optional)

Used to specify a name for a file that is created on the server. If not provided, the system generates a file name and gives it the extension `.txt`.

columnname (optional)

Value of each column in the row to be added. The `columnname` is a table column name.

Description

The `addrowtext` command adds a row to a table. If the table contains an upload column, instead of requiring a file upload, text can be uploaded and Content Server automatically creates a file to store the uploaded text.

errno

The possible values of `errno` include:

Value	Description
-103	No such table.
-104	No table definition.
-105	Database error.

See Also

The following `ics:catalogmanager` commands:

[addrow](#)

[addrows](#)

addUser (Deprecated)

As of Content Server version 4.0, replaced by [object:create](#).

Syntax

```
<ics:catalogmanager>
  <ics:argument name="ftcmd" value="addUser" />
  <ics:argument name="username" value="name" />
  <ics:argument name="password" value="password"/>
  <ics:argument name="useracl" value="acl"/>
</ics:catalogmanager>
```

Parameters

`ftcmd` (required)

Value must be set to `addUser`.

`username` (required)

Name of user to add.

`password` (required)

Password for the user. The password is stored in an encrypted format.

`useracl` (required)

Access Control List. Comma-separated list of groups.

errno

The possible values of `errno` include:

Value	Description
-105	Database error.

See Also

The following `ics:catalogmanager` commands:

[delUser \(Deprecated\)](#)

[modifyUser \(Deprecated\)](#)

commit

Commits a record from a tracked table into the RevisionTracking subsystem using `ics:catalogmanager`.

Syntax

```
<ics:catalogmanager>
  <ics:argument name="ftcmd" value="commit" />
  <ics:argument name="tablename" value="table" />
  <ics:argument name="asset" value="primarykey"/>
  [<ics:argument name="checkout" value="true or false"/>]
  [<ics:argument name="annotation" value="comment"/>]
</ics:catalogmanager>
```

Parameters

`ftcmd` (required)

Value must be set to `commit`.

`tablename` (required)

Name of the table.

`asset` (required)

Value of the primary key for the row to commit.

`checkout` (optional)

Indicates lock status of record after commit. Set value to `true` to indicate the record is still locked by the user after the commit.

`annotation` (optional)

Comment describing the new revision.

Description

The `commit` command commits a record from a tracked table into the RevisionTracking subsystem. The current user must have the record locked and have write access to the tracked table to perform this operation.

errno

The possible values of `errno` include:

Value	Description
-103	No such table.
-104	No table definition.
-105	Database error.

See Also

The following `ics:catalogmanager` commands:

[lock](#)
[unlockrecord](#)

createtable

Creates a table using `ics:catalogmanager`.

Syntax

```
<ics:catalogmanager>
  <ics:argument name="ftcmd" value="createtable" />
  <ics:argument name="tablename" value="table" />
  <ics:argument name="systable" value="no/obj"/>
  <ics:argument name="uploadDir" value="directory" />
  <ics:argument name="aclList" value="acl1,acl2..." />
  <ics:argument name="colnameN" value="name" />
  <ics:argument name="colvalueN" value="value"/>
</ics:catalogmanager>
```

Parameters

ftcmd (required)

Value must be set to `createtable`.

tablename (required)

Name of the table to create.

systable (required)

Allows differentiation between ContentCatalog (`systable=no`) and ObjectCatalog (`systable=obj`). The default is ContentCatalog (`systable=no`).

uploadDir (required)

Server-side folder path to contain uploaded files.

aclList (optional)

Comma-separated list of ACLs. When security is enabled, only members of the ACLs can access content in the table.

colvalueN (required)

Database-specific field type for column. The value of `N` is 0 (zero) for the first column, 1 for the second column, and so on.

colnameN (required)

Column field name. Each `colname` parameter corresponds to a column in the table. The value of `N` is 0 (zero) for the first column, 1 for the second column, and so on.

Description

The `createtable` command creates a table. The table's primary key field name is defined in the Content Server properties file (`futuretense.ini`).

Note

The default primary key used by CatalogManager is defined in the property `cc.contentkey`. Unlike a custom property for primary keys, the `cc.contentkey` property does not restrict the name of the tables that you create. The Object table column, version, is reserved for future use.

errno

The possible values of `errno` include:

Value	Description
-22	Table already exists.
-105	Database error.

See Also

The following `ics:catalogmanager` command:

[deletetable](#)

deletelog

Deletes the log file using `ics:catalogmanager (futuretenseip_address.txt)` if per-IP address logging is enabled.

Syntax

```
<ics:catalogmanager>
  <ics:argument name="ftcmd" value="deletelog" />
</ics:catalogmanager>
```

Parameters

`ftcmd` (required)

Value must be set to `deletelog`.

Description

The `deletelog` command deletes the log file (`futuretenseip_address.txt`) if per-IP address logging is enabled. Enable per-IP address logging by setting the `ft.dbl` property, found in the Content Server properties file (`futuretense.ini`), to `yes`.

This command is useful for removing the existing log file so that you can create a new one when debugging. When the application server restarts and Content Server is called the first time, the Content Server servlet attempts to delete all previous log files, regardless of whether per-IP address logging is enabled. This means that the Content Server log file is deleted as well as any other `.TXT` files in the directory where the logs are stored.

errno

The possible values of `errno` include:

Value	Description
-300	File not found.

See Also

The following `ics:catalogmanager` command:

[exportlog](#)

deleterevision

Deletes a revision of a row from a tracked table using `ics:catalogmanager`.

Syntax

```
<ics:catalogmanager>
  <ics:argument name="ftcmd" value="deleterevision" />
  <ics:argument name="tablename" value="table" />
  <ics:argument name="asset" value="primarykey"/>
  [<ics:argument name="version" value="first|last|#|date"/>]
</ics:catalogmanager>
```

Parameters

`ftcmd` (required)

Value must be set to `deleterevision`.

`tablename` (required)

Name of the tracked table.

`asset` (required)

Value of the primary key for the row to obtain history.

`version` (optional)

Possible values are:

- `first` - history of earliest revision of record.
- `last` - history of most recent revision of record.
- `#` - history of a specific revision number.
- `date` - history of the record to whatever revision was current at the specified date. The date must be in SQL format (`yyyy-mm-dd hh:mm:ss`) and is expected to be Greenwich Mean Time.

Description

The `deleterevision` command deletes a revision of a row from a tracked table. The current user must have `rtadmin` access to the tracked table to perform this operation.

errno

The possible values of `errno` include:

Value	Description
-103	No such table.
-104	No table definition.
-105	Database error.

See Also

The following `ics:catalogmanager` command:

[release](#)

deleterow

Deletes a row in a table using `ics:catalogmanager`.

Syntax

```

<ics:catalogmanager>
  <ics:argument name="ftcmd" value="deleterow" />
  <ics:argument name="tablename" value="table" />
  [<ics:argument name="Delete uploaded file(s)" value="yes|no"/>]
  [<ics:argument name="primarykey" value="columnname"/>]
  [<ics:argument name="tablekey" value="column name"/>]
  [<ics:argument name="tablekeyvalue" value="value"/>]
</ics:catalogmanager>

```

Parameters

ftcmd (required)

Value must be set to `deleterow`.

tablename (required)

Name of the table containing the row to delete.

Delete uploaded file(s) (optional)

Set to `yes` if the upload files associated with row should be deleted. Default is `no`.

primarykey (optional)

`primarykey` is the column name of the primary key column. The value is the value of row's primary key.

tablekey (optional)

Use the `tablekey` parameter if you want to delete rows based on a value other than the primary key. The value of `tablekey` is the column name of the column you use to perform the deletion. For example, if you want to delete a table based on the value in the `title` column, the `tablekey` parameter looks as follows:

```
inList.setValString("tablekey", "title");
```

`tablekeyvalue` (optional)

Value in the `tablekey` column of the row you want to delete. For example, if you want to delete a row with a value of `Jaws` in the `title` column, the `tablekeyvalue` parameter looks as follows:

```
inList.setValString("tablekeyvalue", "Jaws");
```

This parameter is required only if the `tablekey` parameter is used.

Description

The `deleterow` command deletes a row in a table. There are two options that you can use to delete a row with this command:

- To delete rows using the primary key, set the following parameters:
 - `ftcmd`
 - `tablename`
 - `primarykey`
 - If the `primarykey` is `id`, you must have a variable called `id set`.
- To delete one or more rows that match some "alternate" column value, set the following parameters:
 - `ftcmd`
 - `tablename`
 - `tablekey`
 - `tablekeyvalue`

Note

This is valid only if the primary key column does not have a variable set.

errno

The possible values of `errno` include:

Value	Description
-105	Database error.

See Also

The following `ics:catalogmanager` commands:

[addrow](#)
[deleterows](#)

deleterows

Deletes multiple rows in a table using `ics:catalogmanager`.

Syntax

```
<ics:catalogmanager>
  <ics:argument name="ftcmd" value="deleterows" />
  <ics:argument name="tablename" value="table" />
  <ics:argument name="primarykeyN" value="keyvalue"/>
  [<ics:argument name="Delete uploaded file(s)"
    value="yes|no"/>]
</ics:catalogmanager>
```

Parameters

`ftcmd` (required)

Values must be set to `deleterows`.

`tablename` (required)

Name of the table containing the rows to delete.

`primarykeyN` (required)

Value of row's primary key. The `primarykey` parameter is the primary key's column name. The value of `N` corresponds to the row number—in the resultset—that you want to delete.

`Delete uploaded file(s)`(optional)

Value should be set to `yes` if upload files associated with row should be deleted. Default is `no`.

Description

The `deleterows` command deletes rows in a table. You can also delete uploaded files.

errno

The possible values of `errno` include:

Value	Description
-105	Database error.

See Also

The following `ics:catalogmanager` commands:

[deleterow](#)

deletetable

Deletes a table using `ics:catalogmanager`.

Syntax

```
<ics:catalogmanager>
  <ics:argument name="ftcmd" value="deletetable" />
  <ics:argument name="tablename" value="table" />
</ics:catalogmanager>
```

Parameters

ftcmd (required)
Specify the literal value `deletetable`.

tablename (required)
Specify the name of the table to delete.

Description

The `deletetable` command deletes a table. This command does not delete the upload storage folder.

errno

The possible values of `errno` include:

Value	Description
-105	Database error.

See Also

[createtable](#)

delUser (Deprecated)

As of Content Server version 4.0, replaced by [object:delete](#).

Syntax

```
<ics:catalogmanager>
  <ics:argument name="ftcmd" value="delUser" />
  <ics:argument name="username" value="name" />
</ics:catalogmanager>
```

Parameters

ftcmd (required)
Value must be set to `delUser`.

username (required)
Name of the user to delete.

errno

The possible values of `errno` include:

Value	Description
-105	Database error.

See Also

The following `ics:catalogmanager` commands:

[addUser \(Deprecated\)](#)

editrow

Modifies the fields of a row in a table using `ics:catalogmanager`.

Syntax

```

<ics:catalogmanager>
  <ics:argument name="ftcmd" value="editrow" />
  <ics:argument name="tablename" value="table" />
  <ics:argument name="primarykey" value="key"/>
  [<ics:argument name="columnname" value="column name"/>]
  [<ics:argument name="urlcolumnname_folder" value="folder name"/>]
  >]
  [<ics:argument name="columnname_file" value="file name"/>]
  [<ics:argument name="tablekey" value="column name"/>]
  [<ics:argument name="tablekeyvalue" value="some value"/>]
</ics:catalogmanager>

```

Parameters

ftcmd (required)

Value must be set to `editrow`.

tablename (required)

Name of the table whose row is edited.

primarykey (required)

Value of a row's primary key. The `primarykey` parameter is the primary key's column name.

columnname (optional)

The information in the field you are editing. The `columnname` parameter is the name of the column to be edited.

urlcolumnname_folder (optional)

Subfolder name to store the uploaded file. The file is then stored under the upload folder and the subfolder. The upload folder is specified in the `SystemInfo` table's `defdir` column. You can have multiple upload columns.

Upload columns are designated by the prefix `url`. The column name in the table must begin with `url` (for example, `urltext`). The argument name must include the table's column name with `_folder` appended to the column name (for example, `urltext_folder`).

`columnname_file` (required for non-binary files)

The name of the file you want to upload. The column name in the table must begin with `url` (for example, `urltext`). The argument name must have `_file` appended to the table's column name (for example, `urltext_file`).

`tablekey` (optional)

Use the `tablekey` parameter if you want to update rows based on a value other than the primary key. The value of `tablekey` is the column name of the column you use to perform the update. For example, if you want to update a table based on the value in the `title` column, the `tablekey` parameter looks as follows:

```
inList.setValString("tablekey", "title");
```

`tablekeyvalue` (optional)

Value in the `tablekey` column of the row you want to update. For example, if you want to update a row with a value of `Jaws` in the `title` column, the `tablekeyvalue` parameter looks as follows:

```
inList.setValString("tablekeyvalue", "Jaws");
```

This parameter is required only if the `tablekey` parameter is used.

Description

The `editrow` command modifies the fields of a row in a table. The fields in the row are modified with the new values. If a URL field is edited, the content of the associated server-side upload file is modified to contain the new value. Thus, a user can pass text to update the back end storage for a given URL column value.

errno

The possible values of `errno` include:

Value	Description
-103	No such table.
-104	No table definition.
-105	Database error.

See Also

The following `ics:catalogmanager` commands:

[editrows](#)

[replacerow](#)

[updaterow](#)

[addrowtext](#)

editrows

Modifies the fields for multiple rows in a table using `ics:catalogmanager`.

Syntax

```
<ics:catalogmanager>
  <ics:argument name="ftcmd" value="editrows" />
  <ics:argument name="tablename" value="table" />
  <ics:argument name="primarykey" value="key"/>
  [<ics:argument name="primarykeyN" value="column name"/>]
  [<ics:argument name="columnnameN" value="column name"/>]
  [<ics:argument name="urlcolumnname_folderN" value="folder
    name"/>]
  [<ics:argument name="columnname_fileN" value="file name"/>]
</ics:catalogmanager>
```

Parameters

ftcmd (required)

Value must be set to `editrows`.

tablename (required)

Name of the table containing rows to be edited.

primarykeynameN (optional)

Name of the column you want to use as the primary key. The value of *N* corresponds to the row number to be edited. For example, if you want to edit the values in row 0 of the table, you use the following code:

```
inList.setValString("id0", "1");
```

columnnameN (optional)

Name of the column that you want to edit. The value of *N* corresponds to the row number—in the resultset—that you want to edit. For example, if you want to edit the value in the `comments` field of row 0 of the table to read “miss it”, you use the following code:

```
inList.setValString("comments0", "miss it");
```

urlcolumnname_folder (optional)

Subfolder name to store the uploaded file. The file is then stored under the upload folder and the subfolder. The upload folder is specified in the `SystemInfo` table’s `defdir` column. You can have multiple upload columns.

Upload columns are designated by the prefix `url`. The column name in the table must begin with `url` (for example, `urltext`). The argument name must include the table’s column name with `_folder` appended to the column name (for example, `urltext_folder`).

columnname_file (required for non-binary files)

Name of the file you want to upload. The column name in the table must begin with `url` (for example, `urltext`). The argument name must have `_file` appended to the table’s column name (for example, `urltext_file`).

Description

The `editrows` command modifies the fields for multiple rows in a table. The fields in the row are modified with the new values. If a `url` field is edited, the content of the associated server-side upload file is modified to contain the new value.

errno

The possible values of `errno` include:

Value	Description
-103	No such table.
-104	No table definition.
-105	Database error.

See Also

The following `ics:catalogmanager` commands:

[editrow](#)

[replacerows](#)

[updaterows](#)

exportform

Displays a default HTML form for adding, deleting, modifying, or selecting rows from a table using `ics:catalogmanager`.

Syntax

```
<ics:catalogmanager>
  <ics:argument name="ftcmd" value="exportform" />
  <ics:argument name="tablename" value="table" />
  <ics:argument name="ftchoice" value="command name"/>
  [<ics:argument name="tablekeyvalue" value="primary key"/>]
</ics:catalogmanager>
```

Parameters

ftcmd (required)

Value must be set to `exportform`.

tablename (required)

Name of the table that the form uses.

ftchoice (required)

Value must be set to one of the following `Catalogmanager` commands: `addrow`, `deleterow`, `updaterow`, `selectrows`, or `replacerow`.

tablekeyvalue (optional)

If `ftchoice` is `deleterow`, `editrow`, or `updaterow`, the value for this input must be the primary key value for the record that you want to delete or update. No `tablekeyvalue` can be supplied when `ftchoice` is set to `addrow`.

errno

The possible values of `errno` include:

Value	Description
-105	Database error.

See Also

The following `ics:catalogmanager` commands:

[addrow](#)

[deleterow](#)

[updaterow](#)

[selectrows](#)

[replacerow](#)

exportlog

Exports a per-IP address log using `ics:catalogmanager`.

Syntax

```
<ics:catalogmanager>
  <ics:argument name="ftcmd" value="exportlog" />
</ics:catalogmanager>
```

Parameter

`ftcmd` (required)

Value must be set to `exportlog`.

Description

The `exportlog` command exports a per-IP address log. You can enable per-IP address logging by setting the `ft.dbl` property, located in the Content Server properties file (`futuretense.ini`), to `yes`.

errno

The possible values of `errno` include:

Value	Description
-300	File not found.

See Also

The following `ics:catalogmanager` commands:

[deletelog](#)

flushcatalog

Flushes the internal memory cache for a table using `ics:catalogmanager`.

Syntax

```
<ics:catalogmanager>
  <ics:argument name="ftcmd" value="flushcatalog" />
  <ics:argument name="tablename" value="table" />
</ics:catalogmanager>
```

Parameters

ftcmd (required)
Value must be set to `flushcatalog`.

tablename (required)
Name of the table to flush.

Description

The `flushcatalog` command flushes the internal memory cache for a table. This command is useful to clear a table's cache after a table update using a non-Content Server mechanism.

`flushcatalog` can also force cache clearing when the SQL used to select against a table has performed a join with other tables; the system may hold the cached information for either table.

errno

The possible values of `errno` include:

Value	Description
-105	Database error.

flushpage

Deletes a page's cache file using `ics:catalogmanager`.

Syntax

```
<ics:catalogmanager>
  <ics:argument name="ftcmd" value="flushpage" />
  <ics:argument name="pagename" value="name" />
</ics:catalogmanager>
```

Parameters

ftcmd (required)
Value must be set to `flushpage`.

pagename (required)
Name of page to be cleared from cache.

Note

The flushpage tag does not return an error in either of the following situations:

- The specified pagename does not exist.
- A page with the supplied criteria does not exist.

history

Gets a revision history for rows in a revision-tracked table using `ics:catalogmanager`.

Syntax

```
<ics:catalogmanager>
  <ics:argument name="ftcmd" value="history" />
  <ics:argument name="tablename" value="table" />
  <ics:argument name="asset" value="primarykey"/>
  [<ics:argument name="version" value="first|last|#|date"/>]
  [<ics:argument name="locker" value="username1"/>]
  [<ics:argument name="creator" value="username2"/>]
  [<ics:argument name="annotation" value="annotation"/>]
</ics:catalogmanager>
```

Parameters

ftcmd (required)

Value must be set to `history`.

tablename (required)

Name of the tracked table.

asset (required)

Value of the primary key for the row to obtain history.

version (optional)

Possible values are:

first - history of earliest revision of record.

last - history of most recent revision of record.

- history of a specific revision number.

date - history of the record to whichever revision was current at the specified date.

The date must be in SQL format (yyyy-mm-dd hh:mm:ss) and is expected to be GMT.

locker (optional)

Filter rows returned based on a user who has rows locked. If the current user does not have `rtaudit` privileges, you must specify a value for `locker`, `creator`, or both. If you specify only `locker`, the value must be the current user's name; otherwise, no data is returned.

creator (optional)

Filter rows returned based on the user who created the row. If the current user does not have `rtaudit` privileges, you must specify a value for `locker`, `creator`, or both. If you specify only `creator`, the value must be the current user's name; otherwise, no data is returned.

annotation (optional)

Filter rows returned based on the annotation of a row.

Description

The history command gets a revision history for rows in a revision-tracked table.

To obtain history for all rows, the current user must have `rtaudit` privileges for the table. If the current user has read privileges on the table, only history for the revisions which the user has created or locked are returned. Specific versions can be specified by using the optional version parameter. Rows may be filtered using the optional parameters `locker` and `creator`. Use the `annotation` parameter to retrieve rows based on their annotation.

errno

The possible values of `errno` include:

Value	Description
-103	No such table.
-104	No table definition.
-105	Database error.

lock

Locks a record for the current user in a tracked table using `ics:catalogmanager`.

Syntax

```
<ics:catalogmanager>
  <ics:argument name="ftcmd" value="lock" />
  <ics:argument name="tablename" value="table" />
  <ics:argument name="asset" value="primarykey"/>
</ics:catalogmanager>
```

Parameters

`ftcmd` (required)

Value must be set to `lock`.

`tablename` (required)

Name of the tracked table.

`asset` (required)

Value of the primary key for the row to lock.

Description

The `lock` command locks a record for the current user in a tracked table. Once a row is locked, the current user can modify fields in the row. The current user must have write access to the table.

errno

The possible values of `errno` include:

Value	Description
-103	No such table.
-104	No table definition.
-105	Database error.

See Also

The following `ics:catalogmanager` command:

[release](#)

login

Authorizes a user within an application using `ics:catalogmanager`.

Syntax

```
<ics:catalogmanager>
  <ics:argument name="ftcmd" value="login" />
  <ics:argument name="username" value="name" />
  <ics:argument name="password" value="password"/>
  [<ics:argument name="system" value="hostname"/>]
</ics:catalogmanager>
```

Parameters

`ftcmd` (required)

Value must be set to `login`.

`username` (required)

Name of existing user.

`password` (required)

Password of user.

errno

The possible values of `errno` include:

Value	Description
-105	Database error.

See Also

The following `ics:catalogmanager` command:

[logout](#)

logout

Logs out a user from within an application using `ics:catalogmanager`.

Syntax

```
<ics:catalogmanager>
  <ics:argument name="ftcmd" value="logout"/>
  [<ics:argument name="killsession" value="true|false"/>]
</ics:catalogmanager>
```

Parameters

ftcmd (required)

Value must be set to `logout`.

killsession (optional)

If the value is set to `true`, it destroys the session maintained on the server. Do this only at the end of processing a given HTTP request since you can not recover or re-create a session until the next HTTP connect issued by the client. This is useful for web users after they finish their session.

errno

The possible values of `errno` include:

Value	Description
-105	Database error.

See Also

The following `ics:catalogmanager` command:

[login](#)

mirrorabort

Aborts a mirror operation in progress using `ics:catalogmanager`.

Syntax

```
<ics:catalogmanager>
  <ics:argument name="ftcmd" value="mirrorabort" />
</ics:catalogmanager>
```

Parameter

ftcmd (required)

Value must be set to `mirrorabort`.

Description

This tag aborts a mirror operation in progress. Note that when you use this command there can be a variety of `errno` values returned.

errno

The possible values of `errno` include:

Value	Description
0	Mirror operation succeeded.
-105	Database error.
-601	Returned from <code>mirrorabort</code> when a mirror operation is not currently in progress.
-602	Mirror operation in progress.
-603	Required variables are missing from the <code>mirrorrows</code> <code>CatalogManager</code> command.
-604	The data (file) associated with a <code>url</code> column could not be found on the source machine.
-605	The schema for a column on the source machine can not be determined.
-606	No rows to mirror from the target system.
-607	Current mirroring operation was aborted via the <code>mirrorabort</code> command.
-608	A connection could not be established to the mirror target server.
-609	The network disconnected communication between the source and target machines.
-610	An attempt was made to mirror from a machine where the database columns are mixed case to a system where the database columns are case-sensitive.
-611	Mirror commit operation failed.
-612	Target server returned an invalid configuration connection message.
-613	Obsolete; is not returned.
-614	Could not create a temporary table on the target server.
-615	Could not delete the temporary table.
-616	Server does not support this mirror operation.
-617	Table is not an Object table.
-618	No tree nodes found matching the <code>nid</code> parameter of <code>FTTreeNode</code> .

See Also

The following `ics:catalogmanager` command:

[mirrorrows](#)

mirrorrows

Copies rows from a table in one database to a table in another database using ics:catalogmanager.

Syntax

```
<ics:catalogmanager>
  <ics:argument name="ftcmd" value="mirrorrows" />
  <ics:argument name="tablekey" value="name" />
  <ics:argument name="tablekeyvalue" value="value"/>
  [<ics:argument name="mirrorport" value="port_number"/>]
  <ics:argument name="mirrortarget" value="target"/>
  [<ics:argument name="mirrorauthusername" value="username"/>]
  [<ics:argument name="mirrorauthpassword" value="password"/>]
  [<ics:argument name="useProxyServer" value="true or false"/>]
  [<ics:argument name="cgiprefix" value="some prefix"/>]
  [<ics:argument name="inimerge" value="ini file"/>]
  [<ics:argument name="secureCommunication" value="HTTPS"/>]
</ics:catalogmanager>
```

Parameters

ftcmd (required)

Value must be set to mirrorrows.

tablename (required)

Name of the source table.

tablekey (required)

Name of the table column to perform a selection on. If the system does not find the key, it uses the primary key of the specified table as the selection column. Contentname is the default value. See the Content Server properties file, futuretense.ini to change this default.

tablekeyvalue (required)

Value of tablekey.

mirrorport (optional)

HTTP target port. Port 80 is used if a value is not specified.

mirrortarget (required)

Network name of the mirror target machine.

mirrorauthusername (optional)

Content Server user name to log in to the target machine. Value is required if security is enabled on the mirror target.

mirrorauthpassword (optional)

Password associated with the value of mirrorauthusername to log into the mirror target machine. Value is required if security is enabled on the mirror target.

useProxyServer (optional)

If the value is true, transmission should be routed through the proxy server. The proxy server and port to use are specified using the cs.mirrorproxyserver and cs.mirrorproxyport properties.

cgiprefix (optional)

Specifies the application server type on the target system. For iPlanet Application Server, set it to:

/NASApp/CS/

For WebLogic and WebSphere, set it to:

/servlet

inimerge (optional)

Name of a property file on the target system that should be loaded before the mirroring operation proceeds. This parameter is used to define primary key values for tables on the target server.

secureCommunication (optional)

If true, uses secure protocol (HTTPS) to transmit the data to the target server.

errno

The possible values of `errno` include:

Value	Description
0	Mirror operation succeeded.
-105	Database error.
-601	Returned from <code>mirrorabort</code> when a mirror operation is not currently in progress.
-602	Mirror operation in progress.
-603	Required variables are missing from the <code>mirrorrows CatalogManager</code> command.
-604	The data (file) associated with a <code>url</code> column could not be found on the source machine.
-605	The schema for a column on the source machine cannot be determined.
-606	No rows to mirror from the target system.
-607	Current mirroring operation was aborted via the <code>mirrorabort</code> command.
-608	A connection could not be established to the mirror target server.
-609	The network disconnected communication between the source and target machines.
-610	An attempt was made to mirror from a machine where the database columns are mixed case to a system where the database columns are case-sensitive.
-611	Mirror commit operation failed.
-612	Target server returned an invalid configuration connection message.
-613	Obsolete; is not returned.

Value	Description
-614	Could not create a temporary table on the target server.
-615	Could not delete the temporary table.
-616	Server does not support this mirror operation.
-617	Table is not an Object table.
-618	No tree nodes found matching the nid parameter of FTTreeNode.

See Also

The following `ics:catalogmanager` command:

[mirrorabort](#)

modifyUser (Deprecated)

As of Content Server version 4.0, replaced by [dir:replaceattrs](#).

Syntax

```
<ics:catalogmanager>
  <ics:argument name="ftcmd" value="modifyUser" />
  <ics:argument name="username" value="name" />
</ics:catalogmanager>
```

Parameters

ftcmd (required)

Value must be set to `modifyUser`.

username (required)

Name of the user to modify.

password (optional)

Password for the user. The password is stored in an encrypted format.

useracl (optional)

Access Control List. Comma-separated list of groups.

errno

The possible values of `errno` include:

Value	Description
-105	Database error.

See Also

The following `ics:catalogmanager` commands:

[addUser \(Deprecated\)](#)

[delUser \(Deprecated\)](#)

release

Releases a lock on a row from a tracked table using `ics:catalogmanager`.

Syntax

```
<ics:catalogmanager>
  <ics:argument name="ftcmd" value="release" />
  <ics:argument name="tablename" value="table" />
  <ics:argument name="asset" value="primarykey"/>
</ics:catalogmanager>
```

Parameters

`ftcmd` (required)

Value must be set to `release`.

`tablename` (required)

Name of the tracked table.

`asset` (required)

Value of the primary key for the row to lock.

Description

The `release` command releases a lock on a row from a tracked table. Only the user who has the row locked against the table can release it. This also reverts the row in the table to its last committed state, overwriting any changes the user made while the row was locked.

errno

The possible values of `errno` include:

Value	Description
-103	No such table.
-104	No table definition.
-105	Database error.

See Also

The following `ics:catalogmanager` command:

[lock](#)

replacerow

Replaces an existing row in a table on a remote server by first deleting it, and then inserting the specified information into the row using `ics:catalogmanager`.

Syntax

```
<ics:catalogmanager>
  <ics:argument name="ftcmd" value="replacerow" />
  <ics:argument name="tablename" value="table" />
  [<ics:argument name="columnname" value="column name"/>]
  [<ics:argument name="urlcolumnname_folder" value="folder name"/>]
  [<ics:argument name="columnname_file" value="file name"/>]
  [<ics:argument name="tablekey" value="column name"/>]
  [<ics:argument name="tablekeyvalue" value="some value"/>]
</ics:catalogmanager>
```

Parameters

ftcmd (required)

Value must be set to `replacerow`.

tablename (required)

Name of the table that contains the row to replace.

primarykey (optional)

Value of row's primary key. The `primarykey` parameter is the primary key's column name.

columnname (optional)

Each column in the row to be replaced. `columnname` is a table column name.

urlcolumnname_folder (optional)

Subfolder name to store the uploaded file. The file is then stored under the upload folder and the subfolder. The upload folder is specified in the `SystemInfo` table's `defdir` column. You can have multiple upload columns.

Upload columns are designated by the prefix `url`. The column name in the table must begin with `url` (for example, `urltext`). The argument name must include the table's column name with `_folder` appended to the column name (for example, `urltext_folder`).

columnname_file (required for non-binary files)

The name of the file you want to upload. The column name in the table must begin with `url` (for example, `urltext`). The argument name must have `_file` appended to the table's column name (for example, `urltext_file`).

tablekey (optional)

Use the `tablekey` parameter if you want to update rows based on a value other than the primary key. The value of `tablekey` is the column name of the column you use to perform the update. For example, if you want to update a table based on the value in the `title` column, the `tablekey` parameter looks as follows:

```
inList.setValString("tablekey", "title");
```

tablekeyvalue (optional)

Value in the `tablekey` column of the row you want to update. For example, if you want to update a row with a value of `Jaws` in the `title` column, the `tablekeyvalue` parameter looks as follows:

```
inList.setValString("tablekeyvalue", "Jaws");
```

This parameter is required only if the `tablekey` parameter is used.

Description

The `replacerow` command replaces an existing row in a table on a remote server by first deleting it, and then inserting the specified information into the row. This command should be used to clear the value of a column in a row.

Note

If the row specified by the `primarykey` value does not exist, then the row is added to the table.

errno

The possible values of `errno` include:

Value	Description
-103	No such table.
-104	No table definition.
-105	Database error.

See Also

The following `ics:catalogmanager` commands:

[editrow](#)

[updaterow](#)

[updaterow2](#)

replacerows

Replaces multiple rows in a table using `ics:catalogmanager`.

Syntax

```
<ics:catalogmanager>
  <ics:argument name="ftcmd" value="replacerows" />
  <ics:argument name="tablename" value="table" />
  [<ics:argument name="primarykeyN" value="some value"/>]
  [<ics:argument name="columnnameN" value="column name"/>]
  [<ics:argument name="urlcolumnname_folderN" value="folder
name"/>]
  [<ics:argument name="columnname_fileN" value="file name"/>]
  [<ics:argument name="tablekeyN" value="column name"/>]
  [<ics:argument name="tablekeyvalueN" value="some value"/>]
  [<ics:argument name="inList.setValString" value="some value"/>]
</ics:catalogmanager>
```

Parameters

ftcmd (required)

Value must be set to `replaceroes`.

tablename (required)

Name of the table that contains the rows to replace.

primarykeyN (optional)

Value of row's primary key. The `primarykey` parameter is the primary key's column name. The value of `N` corresponds to the row number—in the resultset—that you want to replace.

columnnameN (optional)

Each column in the row to be replaced. `columnname` is a table column name. The value of `N` corresponds to the row number—in the resultset—that you want to replace.

urlcolumnname_folderN (optional)

Subfolder name to store the uploaded file. The file is then stored under the upload folder and the subfolder. The upload folder is specified in the `SystemInfo` table's `defdir` column. You can have multiple upload columns.

Upload columns are designated by the prefix `url`. The column name in the table must begin with `url` (for example, `urltext`). The argument name must include the table's column name with `_folder` appended to the column name (for example, `urltext_folder`).

columnname_fileN (required for non-binary files)

Name of the file you want to upload. The column name in the table must begin with `url` (for example, `urltext`). The argument name must have `_file` appended to the table's column name (for example, `urltext_file`).

tablekeyN (optional)

Use the `tablekeyN` parameter if you want to update rows based on a value other than the primary key. The value of `tablekeyN` is the column name of the column you use to perform the update. The value of `N` corresponds to the row number—in the resultset—that you want to update. For example, if you want to update a table based on the value in the `title` column, the `tablekeyN` parameter looks as follows:

```
inList.setValString("tablekey0", "title");
```

tablekeyvalueN (optional)

Value in the `tablekey` column of the row you want to update. The value of `N` corresponds to the row number—in the resultset—that you want to update. For example, if you want to update a row with a value of `Jaws` in the `title` column, the `tablekeyvalueN` parameter looks as follows:

```
inList.setValString("tablekeyvalue0", "Jaws");
```

This parameter is required only if the `tablekey` parameter is used.

Description

The `replaceroes` command replaces multiple rows in a table. If a value is not specified for a column, the column value is cleared.

Note

If the row specified by the primary key value does not exist, then the row is added to the table.

errno

The possible values of `errno` include:

Value	Description
-22	Table already exists.
-103	No such table.
-104	No table definition.
-105	Database error.

See Also

The following `ics:catalogmanager` commands:

[editrows](#)

[updaterows](#)

rollback

Reverts a row in a tracked table to a previous revision using `ics:catalogmanager`.

Syntax

```

<ics:catalogmanager>
  <ics:argument name="ftcmd" value="rollback" />
  <ics:argument name="tablename" value="table" />
  <ics:argument name="asset" value="primarykey"/>
  [<ics:argument name="version" value="first|last|#|date"/>]
</ics:catalogmanager>

```

Parameters

ftcmd (required)

Value must be set to `rollback`.

tablename (required)

Name of the tracked table.

asset (required)

Value of the primary key for the row to roll back.

version (optional)

Possible values are:

- `first` - roll back to earliest revision of record.
- `last` - roll back to most recent revision of record.
- `#` - roll back to a specific revision number.
- `date` - roll back the record to whatever revision was current at the specified date. The date must be in SQL format (`yyyy-mm-dd hh:mm:ss`) and is expected to be GMT.

Description

The rollback command reverts a row in a tracked table to a previous revision. The current user must have the row locked and have `rtaudit` privileges against the table to roll back the row. A row can be rolled back to its earliest revision, its most recent revision, a specific version number or whichever revision was current at a specified date.

errno

The possible values of `errno` include:

Value	Description
-103	No such table.
-104	No table definition.
-105	Database error.

See Also

The following `ics:catalogmanager` commands:

[history](#)

selectrows

Executes a query against a given table and displays records from a table using `ics:catalogmanager`.

Syntax

```
<ics:catalogmanager>
  <ics:argument name="ftcmd" value="selectrow(s)" />
  <ics:argument name="tablename" value="tablename"/>
  [<ics:argument name="selwhere" value="keycolumnname"/>]
  [<ics:argument name="selwhat" value="keycolumnvalue"/>]
  <CSVAR name="Variables.cshttp"/>
</ics:catalogmanager>
```

Parameters

ftcmd (required)

Value must be set to `selectrows`.

tablename (required)

Name of the table to query.

selwhere (optional)

Lists the where clause components. If the system does not find any where clause components, it attempts to build a where clause using variables that correspond to column names. You can have multiple instances of the `selwhere` parameter.

selwhat (optional)

Lists columns to select from the table. The default is `*`, which selects all columns.

Description

The `selectrow(s)` command executes a query against a given table and displays records from a table. The rows displayed match the criteria specified by the value of the parameters. CatalogManager assigns the returned HTML to the `cshtml` variable.

setversions

Sets the number of revisions stored for each row in a tracked table using `ics:catalogmanager`.

Syntax

```
<ics:catalogmanager>
  <ics:argument name="ftcmd" value="setversions" />
  <ics:argument name="tablename" value="table" />
  <ics:argument name="numversions" value="number"/>
</ics:catalogmanager>
```

Parameters

`ftcmd` (required)

Value must be set to `setversions`.

`tablename` (required)

Name of the tracked table.

`numversions` (required)

Number of revisions to be stored for each row in a tracked table.

Description

The `setversions` command sets the number of revisions stored for each row in a tracked table. The current user must have `rtadmin` privileges on the table.

errno

The possible values of `errno` include:

Value	Description
-103	No such table.
-104	No table definition.
-105	Database error.

See Also

The following `ics:catalogmanager` command:

[tracktable](#)

tracktable

Enables revision tracking operations on a table using `ics:catalogmanager`.

Syntax

```
<ics:catalogmanager>
  <ics:argument name="ftcmd" value="tracktable" />
  <ics:argument name="tablename" value="table" />
  [<ics:argument name="numversions" value="some number"/>]
  [<ics:argument name="storage" value="storage location"/>]
</ics:catalogmanager>
```

Parameters

ftcmd (required)

Value must be set to `tracktable`.

tablename (required)

Name of the table to track.

numversions (optional)

Maximum number of versions to store for each row. When the maximum is reached, the oldest version is removed. If the value of `numversions` is not specified or is set to 0, there is no limit to the number of revisions.

storage (optional)

Location to store revisions associated with a row in the table.

Description

The `tracktable` command enables revision tracking operations on a table. The maximum number of revisions per row can be set, as well as the storage location for the revisions. Only users who have `rtadmin` privileges against the table can track the table.

errno

The possible values of `errno` include:

Value	Description
-103	No such table.
-104	No table definition.
-105	Database error.

See Also

The following `ics:catalogmanager` command:

[untracktable](#)

untracktable

Stops tracking a table using `ics:catalogmanager`.

Syntax

```
<ics:catalogmanager>
  <ics:argument name="ftcmd" value="untracktable" />
  <ics:argument name="tablename" value="table" />
</ics:catalogmanager>
```

Parameters

`ftcmd` (required)

Value must be set to `untracktable`.

`tablename` (required)

Name of the tracked table to untrack.

Description

The `untracktable` command stops tracking a table. Revision tracking operations are no longer be available on the table. Only users who have `rtadmin` privileges against the table can untrack the table.

errno

The possible values of `errno` include:

Value	Description
-103	No such table.
-104	No table definition.
-105	Database error.

See Also

The following `ics:catalogmanager` command:

[tracktable](#)

unlockrecord

Unlocks a locked row using `ics:catalogmanager`.

Syntax

```
<ics:catalogmanager>
  <ics:argument name="ftcmd" value="unlockrecord" />
  <ics:argument name="tablename" value="table" />
  <ics:argument name="asset" value="primarykey"/>
</ics:catalogmanager>
```

Parameters

`ftcmd` (required)

Value must be set to `unlockrecord`.

`tablename` (required)

Name of the tracked table.

`asset` (required)

Value of the primary key for the row to unlock.

Description

The `unlockrecord` command unlocks a locked row. Only the user who has the row locked or a user who has `rtadmin` privileges against the table can unlock the row. `unlockrecord` does not restore the row to its most recent revision.

errno

The possible values of `errno` include:

Value	Description
-103	No such table.
-104	No table definition.
-105	Database error.

See Also

The following `ics:catalogmanager` command:

[lock](#)

updaterow

Updates field values for a row in a table using `ics:catalogmanager`.

Syntax

```
<ics:catalogmanager>
  <ics:argument name="ftcmd" value="updaterow" />
  <ics:argument name="tablename" value="table" />
  [<ics:argument name="primarykey" value="primary key value"/>]
  [<ics:argument name="columnname" value="column name"/>]
  [<ics:argument name="urlcolumnname_folder" value="folder name"/>]
  [<ics:argument name="columnname_file" value="file name"/>]
  [<ics:argument name="tablekey" value="column name"/>]
  [<ics:argument name="tablekeyvalue" value="value"/>]
  [<ics:argument name="inList.setValString" value="value"/>]
</ics:catalogmanager>
```

Parameters

ftcmd (required)

Value must be set to `updaterow`.

tablename (required)

Name of the table containing the row to be updated.

primarykey (optional)

Value of row's primary key. The `primarykey` parameter is the primary key's column name.

columnname (optional)

Each column in the row to be modified. `columnname` is a table column name.

urlcolumnname_folder (optional)

Subfolder name to store the uploaded file. The file is then stored under the upload folder and the subfolder. The upload folder is specified in the `SystemInfo` table's `defdir` column. You can have multiple upload columns.

Upload columns are designated by the prefix `url`. The column name in the table must begin with `url` (for example, `urltext`). The argument name must include the table's column name with `_folder` appended to the column name (for example, `urltext_folder`).

columnname_file (required for non-binary files)

Name of the file you want to upload. The column name in the table must begin with `url` (for example, `urltext`). The argument name must have `_file` appended to the table's column name (for example, `urltext_file`).

tablekey (optional)

Use the `tablekey` parameter if you want to update rows based on a value other than the primary key. The value of `tablekey` is the column name of the column you use to perform the update. For example, if you want to update a table based on the value in the `title` column, the `tablekey` parameter looks as follows:

```
inList.setValString("tablekey", "title");
```

`tablekeyvalue` (optional)

Value in the `tablekey` column of the row you want to update. For example, if you want to update a row with a value of `Jaws` in the `title` column, the `tablekeyvalue` parameter looks as follows:

```
inList.setValString("tablekeyvalue", "Jaws");
```

This parameter is required only if the `tablekey` parameter is used.

Description

The `updaterow` command updates field values for a row in a table. For each column and specified value, the existing column value is replaced with the incoming data for that column. Only values specified are changed; you can not clear a value in a table row or column by using `editrows` or `updaterows`. You must use `replacerow` or `updaterow2`.

errno

The possible values of `errno` include:

Value	Description
-103	No such table.
-104	No table definition.
-105	Database error.

See Also

The following `ics:catalogmanager` commands:

[editrow](#)

[replacerow](#)

[updaterow2](#)

updaterow2

Updates or clears values in columns for a row in a table using `ics:catalogmanager`.

Syntax

```
<ics:catalogmanager>
  <ics:argument name="ftcmd" value="updaterow2" />
  <ics:argument name="tablename" value="table" />
  [<ics:argument name="primarykey" value="primary key value"/>]
  [<ics:argument name="columnname" value="columnname"/>]
  [<ics:argument name="urlcolumnname_folder" value="folder
name"/>]
  [<ics:argument name="columnname_file" value="file name"/>]
  [<ics:argument name="tablekey" value="key value"/>]
  [<ics:argument name="tablekeyvalue" value="value"/>]
  [<ics:argument name="inList.setValString" value="value"/>]
```

```
[<ics:argument name="updaterowNullColumns" value="column1,
column2"/>]
</ics:catalogmanager>
```

Parameters

ftcmd (required)

Value must be set to `updaterow2`.

tablename (required)

Name of the table containing the row to be updated.

primarykey (optional)

Value of row's primary key. The `primarykey` parameter name is the primary key's column name.

columnname (optional)

Each column in the row to be modified, `columnname` is a table column name.

urlcolumnname_folder (optional)

Subfolder name to store the uploaded file. The file is then stored under the upload folder and the subfolder. The upload folder is specified in the `SystemInfo` table's `defdir` column. You can have multiple upload columns.

Upload columns are designated by the prefix `url`. The column name in the table must begin with `url` (for example, `urltext`). The argument name must include the table's column name with `_folder` appended to the column name (for example, `urltext_folder`).

columnname_file (required for non-binary files)

Name of the file you want to upload. The column name in the table must begin with `url` (for example, `urltext`). The argument name must have `_file` appended to the table's column name (for example, `urltext_file`).

tablekey (optional)

Use the `tablekey` parameter if you want to update rows based on a value other than the primary key. The value of `tablekey` is the column name of the column you use to perform the update. For example, if you want to update a table based on the value in the `title` column, the `tablekey` parameter looks as follows:

```
inList.setValString("tablekey", "title");
```

tablekeyvalue (optional)

Value in the `tablekey` column of the row you want to update. For example, if you want to update a row with a value of `Jaws` in the `title` column, the `tablekeyvalue` parameter looks as follows:

```
inList.setValString("tablekeyvalue", "Jaws");
```

This parameter is required only if the `tablekey` parameter is used.

updaterowNullColumns (optional)

A comma-separated list of columns that are to be cleared if no data is supplied when the user submits a form.

Note that if this list is supplied, other columns are ignored and is not cleared.

Description

Like `updaterow`, `updaterow2` updates values in columns for a row in a table; however, in contrast to `updaterow`, which does not allow you to clear columns, `updaterow2` allows you to clear columns if there is no value for the specified column (for example, if there is no related field in the form, or if the field on the form was left blank).

errno

The possible values of `errno` include:

Value	Description
-103	No such table.
-104	No table definition.
-105	Database error.

See Also

The following `ics:catalogmanager` commands:

[editrow](#)

[replacerow](#)

updaterows

Modifies field values for multiple rows in a table using `ics:catalogmanager`.

Syntax

```
<ics:catalogmanager>
  <ics:argument name="ftcmd" value="updaterows" />
  <ics:argument name="tablenameN" value="table" />
  [<ics:argument name="primarykeyN" value="primary key value"/>]
  [<ics:argument name="columnnameN" value="columnname"/>]
  [<ics:argument name="urlcolumnname_folderN" value="folder
    name"/>]
  [<ics:argument name="columnname_fileN" value="file name"/>]
  [<ics:argument name="tablekeyN" value="key value"/>]
  [<ics:argument name="inList.setValString" value="value"/>]
  [<ics:argument name="tablekeyvalueN" value="value"/>]
</ics:catalogmanager>
```

Parameters

ftcmd (required)

Value must be set to `updaterows`.

tablename (required)

Name of the table containing rows to be updated.

primarykeyN (optional)

Value of row's primary key. The `primarykey` parameter is the primary key's column name. The value of `N` corresponds to the row number—in the resultset—that you want to update.

columnnameN (optional)

Each column in the row to be updated. `columnname` is a table column name. The value of `N` corresponds to the row number—in the resultset—that you want to update.

urlcolumnname_folderN (optional)

Subfolder name to store the uploaded file. The file is then stored under the upload folder and the subfolder. The upload folder is specified in the SystemInfo table's defdir column. Note that you can have multiple upload columns.

Upload columns are designated by the prefix url. The column name in the table must begin with url (for example, urltext). The argument name must include the table's column name with _folder appended to the column name (for example, urltext_folder).

columnname_fileN (required for non-binary files)

Name of the file you want to upload. The column name in the table must begin with url (for example, urltext). The argument name must have _file appended to the table's column name (for example, urltext_file).

tablekeyN (optional)

Use the tablekeyN parameter if you want to update rows based on a value other than the primary key. The value of tablekeyN is the column name of the column you use to perform the update. The value of N corresponds to the row number—in the resultset—that you want to update. For example, if you want to update a table based on the value in the title column, the tablekeyN parameter looks as follows:

```
inList.setValString("tablekey0", "title");
```

tablekeyvalueN (optional)

Value in the tablekey column of the row you want to update. The value of N corresponds to the row number—in the resultset—that you want to update. For example, if you want to update a row with a value of Jaws in the title column, the tablekeyvalueN parameter looks as follows:

```
inList.setValString("tablekeyvalue0", "Jaws");
```

This parameter is required only if the tablekey parameter is used.

Description

The updatetrows command modifies field values for multiple rows in a table. For fields other than URL fields, the value of the field is modified. For URL fields, updatetrows deletes the file associated with the URL field and modifies the URL field to point to the newly uploaded file.

errno

The possible values of errno include:

Value	Description
-103	No such table.
-104	No table definition.
-105	Database error.

See Also

The following ics:catalogmanager commands:

[addrows](#)

[updatetrows](#)

[updaterows2](#)[editrows](#)[replacerows](#)

updaterows2

Modifies field values or clears columns for multiple rows in a table using ics:catalogmanager.

Syntax

```
<ics:catalogmanager>
  <ics:argument name="ftcmd" value="updaterows2" />
  <ics:argument name="tablename" value="table" />
  <ics:argument name="primarykeyN" value="key"/>
  <ics:argument name="ftcmd" value="updaterows" />
  <ics:argument name="tablenameN" value="table" />
  [<ics:argument name="primarykeyN" value="primary key value"/>]
  [<ics:argument name="columnnameN" value="column name"/>]
  [<ics:argument name="urlcolumnname_folderN" value="folder
  name"/>]
  [<ics:argument name="columnname_fileN" value="file name"/>]
  [<ics:argument name="tablekeyN" value="key value"/>]
  [<ics:argument name="inList.setValString" value="value"/>]
  [<ics:argument name="tablekeyvalueN" value="value"/>]
  [<ics:argument name="updaterowNullColumns" value="column1,
  column2"/>]
</ics:catalogmanager>
```

Parameters

ftcmd (required)

Value must be set to updaterows2.

tablename (required)

Name of the table containing rows to be updated.

primarykeyN (required)

Value of row's primary key. The primarykey parameter is the primary key's column name. The value of N corresponds to the row number—in the resultset—that you want to update.

columnnameN (optional)

Each column in the row to be replaced. columnname is a table column name. The value of N corresponds to the row number—in the resultset—that you want to update.

urlcolumnname_folderN (optional)

Subfolder name to store the uploaded file. The file is then stored under the upload folder and the subfolder. The upload folder is specified in the SystemInfo table's defdir column. You can have multiple upload columns.

Upload columns are designated by the prefix url. The column name in the table must begin with url (for example, urltext). The argument name must include the table's column name with _folder appended to the column name (for example, urltext_folder).

`columnname_fileN` (required for non-binary files)

Name of the file you want to upload. The column name in the table must begin with `url` (for example, `urltext`). The argument name must have `_file` appended to the table's column name (for example, `urltext_file`).

`tablekeyN` (optional)

Use the `tablekeyN` parameter if you want to update rows based on a value other than the primary key. The value of `tablekeyN` is the column name of the column you use to perform the update. The value of `N` corresponds to the row number—in the resultset—that you want to update. For example, if you want to update a table based on the value in the `title` column, the `tablekeyN` parameter looks as follows:

```
inList.setValString("tablekey0", "title");
```

`tablekeyvalueN` (optional)

Value in the `tablekey` column of the row you want to update. The value of `N` corresponds to the row number—in the resultset—that you want to update. For example, if you want to update a row with a value of `Jaws` in the `title` column, the `tablekeyvalueN` parameter looks as follows:

```
inList.setValString("tablekeyvalue0", "Jaws");
```

This parameter is required only if the `tablekey` parameter is used.

`updaterowNullColumns` (optional)

Comma-separated list of columns that are cleared if no data is supplied when the user submits a form. Note that if this list is supplied, other columns are ignored and are not cleared.

Description

Like `updaterows`, the `updaterow2` command modifies field values for multiple rows in a table; however, in contrast to `updaterows`, which does not allow you to clear columns, `updaterows2` allows you to clear columns if there is no value for the specified column (for example, if there is no related field in the form). For `url` fields, `updaterows2` deletes the file associated with the `url` field and modifies the `url` field to point to the newly uploaded file.

errno

The possible values of `errno` include:

Value	Description
-103	No such table.
-104	No table definition.
-105	Database error.

See Also

The following `ics:catalogmanager` commands:

[addrows](#)

[editrows](#)

[replacerows](#)

[updaterows](#)

ics:cdata

Includes a CDATA in the output of a rendered XML element.

Syntax

```
<ics:cdata>
    ...data...
</ics:cdata>
```

Parameters

None.

Description

This tag enables streaming a CDATA wrapper around some data from Content Server. Use this tag in cases where another XML parser reads the page; for example WAP or SOAP.

Example

In the following example, the variable `body` is set to “This is the body of my article.” The following code is used:

```
<ics:cdata>
    <ics:getvar name="body"/>
</ics:cdata>
```

The output is as follows:

```
<![CDATA[
    This is the body of my article.
]]>
```

Release Introduced

This tag was introduced in CSEE release 5.0.

ics:clearerrno

Clears the current error state.

Syntax

```
<ics:clearerrno/>
```

Description

The `ics:clearerrno` tag clears the current error state, setting the value of the error state to 0. Call this tag immediately before invoking a tag that may alter `errno`.

errno

None.

Example

The following example clears the error state before executing the `ics:catalogmanager` command:

```
<ics:clearerrno/>
  <ics:catalogmanager>
    <ics:argument name="ftcmd" value="deletetable" />
    <ics:argument name="tablename" value="movies" />
  </ics:catalogmanager>
```

ics:copylist

Creates a copy of the specified list.

Syntax

```
<ics:copylist
  to="new list"
  from="list"/>
```

Parameters

to (required)

Name of the list created as a result of the copy.

from

Source list.

Description

The `ics:copylist` tag creates a copy of the specified list. The column values for the current row can be retrieved by using the `ics:listget` tag.

You can use the following attributes as the `fieldname` parameter of the `ics:listget` tag to return information about the list:

#curRow	Retrieves the current row number.
#numCols	Retrieves the number of columns in the list.
#numRow	Retrieves the number of rows in the list.

errno

The possible values of `errno` include:

Value	Description
-505	List copy error.

Example

The following example selects all rows from the NewPortalArticle table and stores them in PortalArticles. PortalArticles is copied to Articles and headline from each row in Articles is displayed.

```
<ics:selectto from="NewPortalArticle" what="*"
list="PortalArticles"/>
  <ics:clearerrno/>
  <ics:copylist to = "Articles" from = "PortalArticles" />
    <ics:if condition=i<%=ics.GetErrno()=0%>i>
      <ics:then>
<ics:looplist list="Articles">
<ics:listget listname="Article" fieldname="headline"/>
<br>
</ics:looplist>
      </ics:then>
    </ics:if>
```

See Also

[ics:listget](#)

ics:dateargument

Argument for the ics:indexadd tag.

Syntax

```
<ics:dateargument
  name="some argument name"
  value="date"/>
```

Parameters

name (required)

Field name.

value (required)

Date to be added to the search index.

Description

This tag specifies a date argument to be added to a search index created with the ics:indexadd tag.

errno

None.

Example

The following example adds a record containing a date field called articledate to the search index c:/searchidx.

```
<ics:clearerrno/>
```

```
<ics:indexadd index="c:/searchidx" entryname="A111112"
entrydetail="Ohio towns" searchengine="AV">
<ics:textargument name="headline" value="Places I have visited in
Ohio"/>
<ics:fileargument name="body" value="c:/temp/body.txt"/>
<ics:dateargument name="articledate" value="1999-01-29"/>
</ics:indexadd>
<ics:if condition=<%ics.GetErrno() != 0%>i>
<ics:then>
Report error
</ics:then>
</ics:if>
```

See Also

[ics:indexadd](#)

[ics:fieldargument](#)

[ics:fileargument](#)

ics:debug

Activates debugging for the ics tags.

Syntax

```
<ics:debug
    on="true|false"/>
```

Parameters

on (required)

Set this parameter to `true` to enable debug output. Set it to `false` to disable debug output.

Description

The `ics:debug` tag is used with other tags and provides debugging information for tags executed within a specific thread.

errno

None.

Example

The following example activates debugging:

```
<ics:debug on="true"/>
```

See Also

[object:install](#)

ics:disablecache

Disables caching.

Syntax

```
<ics:disablecache/>
```

Description

Forces the current evaluation to not be cached either by the EvalServer or the ContentServer servlets.

errno

The value of `errno` is unchanged.

Example

The following example disables caching.

```
<ics:disablecache/>
```

ics:else

Conditional else clause.

Syntax

```
<ics:else/>
```

Description

The `ics:else` tag executes a block of code when the parent `ics:if` conditional tag value is false.

errno

None.

Example

```
<ics:if condition='<%=ics.GetErrno()==0%>'>
<ics:then>
Operation succeeded
</ics:then>
<ics:else>
Operation failed<BR>
</ics:else>
</ics:if>
```

See Also

[ics:argument](#)

[ics:if](#)

ics:encode

Builds a URL string from a base and key/value pairs.

Syntax

```
<ics:encode
    base="base"
    [session="true|false"]
    [output="output"]/>
```

Parameters

base (required)

Base of the URL.

session (optional)

Specifies to include the session ID in the URL when set to `true`. Set this parameter to `true` if the session ID is required by the client.

output (optional)

Specifies an optional output variable. If this parameter is not specified the output is streamed to the browser.

Description

This tag builds a URL string from a base and key/value pairs. The keys and values are encoded as required and the session ID can be added optionally.

errno

None.

Example

```
<ics:encode
base='<%ics.GetProperty("ft.cgipath")+ "ContentServer"%>'
    <ics:argument="pagename" value="testpage"/>
    <ics:argument name="k1" value="v1"/>
    <ics:argument name="k2" value="v2"/>
    <ics:argument name="k3" value="v3"/>
    <ics:argument name="k4" value="v4"/>
    <ics:argument name="k5" value="v5"/>
</ics:encode>
```

The browser displays:

/servlet/

ContentServer?pagename=testpage&k1=v1&k2=v2&k3=v3&k4=v4&k5=v5

See Also

[ics:argument](#)

ics:fieldargument

Defines index fields for the Verity search engine.

Syntax

```
<ics:fieldargument
    name="field name"
    value="value" />
```

Parameters

name (required)

Name of the field to be defined.

value (required)

Specifies the type of index field. Supported values are date or text.

Description

The `ics:fieldargument` tag is used by the `ics:indexcreate` tag to specify search index field definitions for Content Server systems using the Verity search engine.

errno

None.

Example

The following example creates a Verity search index with fields called `headline`, `body` and `articledate`:

```
<ics:clearerrno/>
<ics:indexcreate index="c:/verityidx" searchengine="Verity">
    <ics:fieldargument name="headline" value="text"/>
    <ics:fieldargument name="body" value="text"/>
    <ics:fieldargument name="articledate" value="date"/>
</ics:indexcreate>

<ics:if condition=<%ics.GetErrno() != 0%>'>
<ics:then>
    Report error
</ics:then>
</ics:if>
```

See Also

[ics:indexcreate](#)

ics:fileargument

Argument to the ics:indexadd tag.

Syntax

```
<ics:fileargument  
    name="file name"  
    value="value"/>
```

Parameters

name (required)

File name.

value (required)

Full path to the file to be added.

Description

The ics:fileargument tag specifies a file to be added to a search index using the ics:indexadd tag. This tag is an additional argument for the ics:indexadd tag.

errno

None.

Example

The following example adds a record to a search index containing the upload file c:/temp/body.txt:

```
<ics:clearerrno/>  
<ics:indexadd index="c:/searchidx" entryname="A111112"  
entrydetail="Ohio towns" searchengine="AV">  
    <ics:textargument name="headline" value="Places I have visited  
in Ohio"/>  
    <ics:fileargument name="body" value="c:/temp/body.txt"/>  
    <ics:dateargument name="articledate" value="1999-01-29"/>  
</ics:indexadd>  
  
<ics:if condition=<%ics.GetErrno() != 0%>i>  
<ics:then>  
    Report error  
</ics:then>  
</ics:if>
```

See Also

[ics:indexadd](#)

ics:flushcatalog

Flushes the internal memory cache for a table (known previously as a "catalog" in earlier versions of the product).

Syntax

```
<ics:flushcatalog
    catalog="some table"/>
```

Parameters

catalog (required)
Name of the table to be flushed.

Description

The `ics:flushcatalog` command flushes the internal memory cache for a table (known previously as a "catalog" in earlier versions of the product). Use this command if you want to clear a table's cache after a table update using a non-Content Server mechanism.

`ics:flushcatalog` can also force cache clearing when the SQL used to select against a table has performed a join with other catalogs and the system may hold the cached information for either table.

errno

The possible values of `errno` include:

Value	Description
-105	Database error.

Example

The following example flushes the table for the `Article` table

```
<ics:flushcatalog catalog="Article"/>
```

ics:flushstream

Flushes a data stream.

Syntax

```
<ics:flushstream/>
```

Parameters

None.

errno

None.

See Also

[ics:flushcatalog](#)

ics:geterrno

Gets the current error code as an integer.

Syntax

```
<ics:geterrno/>
```

Description

Retrieves the error code of an element as an integer.

errno

None.

Example

The following example issues a call to a tag, then uses `ics:geterrno` to determine if the tag reported an error.

```
<ics:clearerrno/>
<ics:callsql qryname="gettables" list="newrs"/>
<ics:geterrno/>
```

See Also

[ics:debug](#)

ics:getproperty

Gets a property value from the currently loaded Content Server property files.

Syntax

```
<ics:getproperty
  name="property name"
  [file="property file name"]
  [output="variable name"]/>
```

Parameters

name (required)

Name of the property to retrieve.

file (optional)

Semicolon-separated list of property file names.

output (optional)

Specifies an optional output variable. If this parameter is not specified the output is streamed to the browser.

Description

Retrieves a property value from the currently loaded or optionally specified Content Server property file. The property may then be used in conjunction with other Content Server JSP tags.

errno

None.

Example

The following example displays the value of a property from an alternative property file.

```
<ics:getproperty name="myproperty"
                file="myfile.ini"/>
```

ics:getssvar

Returns a Content Sever session variable.

Syntax

```
<ics:getssvar
    name="session variable name"/>
```

Parameters

name (required)
Name of the session variable.

Description

The `ics:getssvar` tag returns the value of a session variable. If the variable does not exist, `ics:getvar` returns an empty string.

errno

None.

Example

The following example retrieves a session variable called `color`:

```
<ics:getssvar name="color"/>
```

See Also

[object:stream](#)

ics:getvar

Returns the value of a Content Server variable.

Syntax

```
<ics:getvar
    name="varname"
    encoding="default"
    [output="variable name"]/>
```

Parameters

name (required)

Name of the variable.

encoding (required)

Encoding used to convert binary values to a string. The value of this parameter must be set to default.

output (optional)

Specifies an optional output variable. If this parameter is not specified, the output is streamed to the browser.

Description

The `ics:getvar` tag returns the value of a variable. If the variable does not exist, `ics:getvar` returns an empty string.

errno

None.

Example

The following line retrieves the value of the Content Server variable `username`.

```
<ics:getvar name="username"
    encoding="default" />
```

See Also

[ics:setvar](#)

ics:if

Processes other tags or displays text based on a logical condition.

Syntax

```
<ics:if condition="logical expression">
  <ics:then>
    tags and/or text
  </ics:then>
  <ics:else>
    tags and/or text
  </ics:else>
</ics:if>
```

Parameters

condition (required)

Takes a true or false logical expression in Java.

Description

The `ics:if` tag processes other tags or displays text based on a logical condition. If the condition is true, the content of the `ics:then` tag is processed. If the condition is false, the content of the `ics:else` tag is processed. The `ics:then` tag is required as part of an `ics:if` tag. The `ics:else` tag is not required.

errno

The value of `errno` is unchanged.

Example

The following example checks the value of `errno`:

```
<ics:if condition="<%=ics.GetErrno()!=0%>">
```

See Also

[ics:else](#)

[ics:then](#)

ics:indexadd

Adds an entry to an existing search index.

Syntax

```
<ics:indexadd
  entryname="entry name"
  [index="index name"]
  [entrydetail="entry details"]
  [wordlist="words"]
  [wordlistdelimiters="delimiters"]
  [ics:filelist="files"]
  [date="date string"]
  [charset="character set"]
  [searchengine="name"]>
  <ics:fileargument name="file argument"
    value="file argument value"/>
  <ics:dateargument name="date argument name"
    value="date argument value"/>
  <ics:textargument name="text argument"
    value="text argument value"/>
</ics:indexadd>
```

Parameters

entryname (required)

Name of the index entry to add. This value is returned in the search results. The entry value must be unique within an index. Choose entry values carefully so that the caller of the search can reference the associated article(s).

index (optional)

Name of the index to which the entry should be added. Default is specified in the Content Server properties file.

entrydetail (optional)

Detail text returned in search results.

wordlist (optional)

Words to add to the index. These are whitespace-separated by default but you can use the `wordlistdelimiters` parameter to specify something else. Use this parameter to add text that is not associated with a field of the index.

wordlistdelimiters (optional)

String containing delimiter characters to use to separate words in the value of the `TEXTics:argument` tag and the `wordlist` parameter. White space is used as delimiter characters if `wordlistdelimiters` is not specified.

ics:filelist (optional)

For AltaVista, specify a comma-separated list of files to add. For Verity, specify a single file (not a list of files). Use this parameter to add text that is not associated with a field of the index.

date (optional)

Date string to add to the index entry. If no date is specified, the `ics:indexadd` time is returned in the search results. Format is in Java SQL.

charset (optional)

Constant value representing the character set the index uses. For the AltaVista search engine this value may be 0, 1, or 2 (ISO_LATIN1, ASCII8, UTF8). If you do not specify `charset`, Content Server uses the values of `av.charset` from the Content Server the properties file.

For the Verity search engine, this value specifies the name of the subdirectory of the common directory where the locale is defined. If you do not specify `charset`, Content Server uses the value of `verity.charset` in the Content Server properties file.

searchengine (optional)

Name of the search engine to use. If `searchengine` is not specified, Content Server uses the value of `cs.searchengine` from the Content Server properties file.

Description

The `ics:indexadd` tag adds an entry to an existing search index.

For both AltaVista and Verity, arguments are optional using `ics:fileargument`, `ics:dateargument` and `ics:textargument` tags. These arguments specify the index field (the name parameter) to which the data (the value parameter) is to be added. When using the `ics:fileargument` value with AltaVista, specify a comma-separated list of files to add to the index with the given name. When using the `ics:argument` value with Verity, specify a single file (not a list of files). Use this parameter to add text that is not associated with a field of the index. The `ics:dateargument` is a date string specified in JAVASQL format. The `ics:textargument` value is a string delimited by white space. The white-space delimiter characters may be overridden by the `wordlistdelimiters` parameter.

The search engine collects arguments with the same name into a concatenated single value for that field.

errno

The possible values of `errno` include:

Value	Description
-300	File not found.
-801	Cannot load search engine.
-804	Add index failed.
-805	No default index specified.
-806	Unknown search engine.
-811	Could not call native method.
-812	Index does not exist.
-815	Duplicate field values. (Verity only)
-816	Could not create temporary populate file. (Verity only)
-817	Could not create temporary data file. (Verity only)

Example

The following example adds an entry to an existing AltaVista search index.

```
<ics:clearerrno/>
<ics:indexadd index="c:/avindex" entryname="A111111" date="1999-
01-29" entrydetail="Example index entry" searchengine="AV">
  <ics:textargument name="headline" value="Clinton to seek third
term"/>
  <ics:fileargument name="body" value="c:/articles/
clinton12423.txt"/>
</ics:indexadd>

<ics:if condition=<%ics.GetErrno() != 0%>'>
<ics:then>
  Report error
</ics:then>
</ics:if>
```

See Also

[ics:indexcreate](#)

[ics:indexremove](#)

[ics:fileargument](#)

[ics:fieldargument](#)

[ics:dateargument](#)

ics:indexcreate

Creates a search index.

Syntax

```
<ics:indexcreate
  index="index name"
  charset="character set"
  [searchengine="name"]/>
```

Parameters

index (required)

Full path name of the index to create.

searchengine (optional)

Name of the search engine to use. If searchengine is not specified, Content Server uses the value of `cs.searchengine` from the Content Server properties file.

Description

The `ics:indexcreate` tag creates a search index. For AltaVista, this creates the directory where the index is stored. For Verity, however, this creates a more complex base structure for the index in the directory specified by the `index` parameter. In particular, it needs to create the structures that hold the field information for the index. Therefore, the

fields that are used must be specified using `ics:fieldargument` tags when the index is created.

errno

The possible values of `errno` include:

Value	Description
-801	Cannot load search engine.
-802	Unsupported search function. Content Server 3.0 and earlier returned this error; however, Content Server 3.1 and higher no longer return this error.
-803	Create index failed.
-805	No default index specified.
-806	Unknown search engine.
-811	Could not call native method.
-815	Duplicate field values. (Verity only)
-819	Unrecognized field type. (Verity only)
-822	Index already exists.(Verity only)

Example

The following example creates an AltaVista search index in the directory `c:/search/avindex`. Note that index fields are not predefined for AltaVista indexes.

```
<ics:clearerrno/>
<ics:indexcreate index="c:/search/avindex" searchengine="AV"/>

<ics:if condition=<%ics.GetErrno() != 0%>'>
<ics:then>
  Report error
</ics:then>
</ics:if>
```

See Also

[ics:indexdestroy](#)

[ics:indexexists](#)

[ics:fieldargument](#)

ics:indexdestroy

Deletes an existing search index.

Syntax

```
<ics:indexdestroy
    index="index name"
    [charset="character set"]
    [searchengine="name"]/>
```

Parameters

index (required)

Full path name of the index to delete. If you specify a nonexistent index name, `ics:indexdestroy` does not return an error.

charset (optional)

Constant value representing the character set the index uses.

For the AltaVista search engine this value may be 0, 1, or 2 (ISO_LATIN1, ASCII8, UTF8). If you do not specify `charset`, Content Server uses the value of `av.charset` in the Content Server properties file.

Because Java Strings are transformed to UTF8, as they pass through the JNI interface to AltaVista, the 0 value only works for 7-bit ASCII. For all other cases, use the value for UTF8.

For the Verity search engine, this value specifies the name of the subdirectory of the common directory where the locale is defined. If you do not specify `charset`, Content Server uses the value of `verity.charset` in the Content Server properties file.

searchengine (optional)

Name of the search engine to use. If `searchengine` is not specified, Content Server uses the value of `cs.searchengine` from the Content Server properties file.

Description

The `ics:indexdestroy` tag deletes an existing search index.

If you call `ics:` after deleting an index, `ics:indexexists` returns -812 (indicating that the index no longer exists).

For AltaVista, after calling `ics:indexdestroy` to delete an existing index, the base directory specified in the `index` parameter usually still exists and contains an empty lock file. In this case, `ics:indexexists` still reports, as expected, that the index does not exist.

For Verity, after calling `ics:indexdestroy` to delete an existing index, the entire directory structure is left.

Note

The `ics:indexdestroy` tag deletes indexes only. Directories that contain the indexes must be deleted manually for both AltaVista and Verity.

errno

The possible values of `errno` include:

Value	Description
-801	Cannot load search engine.
-804	Delete index failed.
-805	No default index specified.
-806	Unknown search engine.

Example

The following example deletes the `Article` index.

```
<ics:clearerrno/>
<ics:indexdestroy index="c:/ArticleIdx" searchengine="AV"/>
<ics:if condition=<%ics.GetErrno() != 0%>'>
<ics:then>
  Report error
</ics:then>
</ics:if>
```

See Also

[ics:indexcreate](#)

ics:indexexists

Use this tag to check for the existence of an index.

Syntax

```
<ics:indexexists
  index="index name"
  [charset="character set"]
  [searchengine="name"/>
```

Parameters

`index` (required)

Full path name of the index to check.

charset (optional)

Constant value representing the character set the index uses.

For the AltaVista search engine this value may be 0, 1, or 2 (ISO_LATIN1, ASCII8, UTF8). If you do not specify charset, Content Server uses the value of `av.charset` in the Content Server properties file.

Because Java Strings are transformed to UTF8, as they pass through the JNI interface to AltaVista, the 0 value only works for 7-bit ASCII. For all other cases, use the value for UTF8.

For the Verity search engine, this value specifies the name of the subdirectory of the common directory where the locale is defined. If you do not specify charset, Content Server uses the value of `verity.charset` in the Content Server properties file.

searchengine (optional)

Name of the search engine to use. If searchengine is not specified, Content Server uses the value of `cs.searchengine` from the Content Server properties file.

errno

The possible values of errno include:

Value	Description
9	Index exists.
-806	Unknown search engine.
-812	Index does not exist.

Example

The following example checks for the existence of `c:\ArticleIdx` and creates it if it doesn't exist.

```
<ics:clearerrno/>
<ics:indexexists index="c:/articleidx" searchengine="AV"/>
<ics:if condition='<%=ics.GetErrno() != 9%>'>
  <ics:then>
    <ics:indexcreate index="c:/articleidx" searchengine="AV"/>
  </ics:then>
</ics:if>
```

See Also

[ics:indexcreate](#)

ics:indexremove

Removes an entry from an existing search index.

Syntax

```
<ics:indexremove
    entryname="entry name"
    [searchengine="name"]
    [index="some index path name"]
```

Parameters

entryname (required)

Name of the index entry to remove.

searchengine (optional)

Name of the search engine to use. If searchengine is not specified, Content Server uses the value of `cs.searchengine` from the Content Server properties file.

index (optional)

Full path name of the index from which the `entryname` should be removed. If not specified, Content Server takes the value of `av.defaultindex` or `verity.defaultindex` in the Content Server properties file.

errno

The possible values of `errno` include:

Value	Description
-801	Cannot load search engine.
-804	Remove index failed.
-805	No default index specified.
-806	Unknown search engine.
-811	Could not call native method.
-812	Index does not exist.

Example

The following example removes `Article083742132` from the index `c:/ArticleIdx`.

```
<ics:clearerrno/>
<ics:indexremove index="c:/ArticleIdx" searchengine="AV"
entry="Article083742132"/>

<ics:if condition='<%ics.GetErrno() != 0%>'>
<ics:then>
    Report error
</ics:then>
</ics:if>
```

See Also

[ics:indexadd](#)

[ics:indexcreate](#)

[ics:indexdestroy](#)

ics:indexreplace

Replaces an existing entry in a search index.

Syntax

```
<ics:indexreplace
  entry="entry name"
  index="index name"
  detail="entry detail"
  [wordlist="words"]
  [wordlistdelimiters="delimiters"]
  [filelist="files"]
  [date="date string"]
  [characterset="character_set"]
  [searchengine="name"]
  [filelist="file list"]
  [date="date"] />
```

Parameters

wordlist (optional)

Words to add to the index. These are whitespace-separated by default but you can use the **wordlistdelimiters** parameter to specify something else. Use this parameter to replace text that is not associated with a field of the index.

wordlistdelimiters (optional)

String containing delimiter characters to use to separate words in the value of the **ics:textargument** tag and the **wordlist** parameter. White space is used as delimiter characters if **wordlistdelimiters** is not specified.

filelist (optional)

For AltaVista, specify a comma-separated list of files to add. For Verity, specify a single file (not a list of files). Use this parameter to replace text that is not associated with a field of the index.

date (optional)

Date string to replace to the index entry. If no date string is specified, the **ics:indexadd** time is returned in the search results. Format is in Java SQL.

characterset (optional)

Constant value representing the character set the index uses. For the AltaVista search engine this value may be 0, 1, or 2 (ISO_LATIN1, ASCII8, UTF8). If you do not specify **characterset**, Content Server uses the value of **av.charset** from the Content Server properties file. Because Java Strings are transformed to UTF8, as they pass through the JNI interface to AltaVista, the 0 value only works for 7-bit ASCII. For all other cases, use the value for UTF8. For the Verity search engine, this value specifies the name of the subdirectory of the common directory where the locale is

defined. If you do not specify `charset`, Content Server uses the value of `verity.charset` in the Content Server properties file.

searchengine (optional)

Name of the search engine to use. If `searchengine` is not specified, Content Server uses the value of `cs.searchengine` in the Content Server properties file.

Description

The `ics:indexreplace` tag replaces an existing entry in a search index.

Arguments are optional using `ics:fileargument`, `ics:dateargument`, and `ics:textargument` tags. These arguments specify the index field (the name parameter) to which the data (the value parameter) is added. The `ics:fileargument` value is a comma separated list of files whose content is added to the index with the given name. The `ics:dateargument` is a date string specified in `JAVASQL` format. The `ics:textargument` value is a string delimited by white space. The white-space delimiter characters may be overridden by the `worddelimiters` tag.

The search engine collects arguments with the same name into a concatenated single value for that field.

errno

The possible values of `errno` include:

Value	Description
-300	File not found.
-801	Cannot load search engine.
-804	Index remove failed.
-805	No default index specified.
-806	Unknown search engine.
-811	Could not call native method.
-812	Index does not exist.
-815	Duplicate field values. (Verity only)
-816	Could not create temporary populated file. (Verity only)
-817	Could not create temporary data file. (Verity only)

See Also

[ics:indexadd](#)

ics:insertpage

Inserts the results of a Content Server page evaluation into the current output stream.

Syntax

```
<ics:insertpage pagename="page name">
  <ics:argument name="arg name" value="value" />
</ics:insertpage/>
```

Parameters

pagename (required)

Name of the page to be inserted into the current output stream.

Description

The `ics:insertpage` tag inserts the results of a Content Server page evaluation into the current output stream or sets the content into a variable. In addition, it supports `ics:argument` constructs for compound URL `ics:insertpage` tags.

errno

The possible values of `errno` include:

Value	Description
-300	File not found.
-303	Path not found.

Example

The following example inserts a page called `MainPage` into the output stream.

```
<ics:insertpage pagename="MainPage"/>
```

See Also

[ics:getvar](#)

ics:listget

Retrieves a list.

Syntax

```
<ics:listget
    listname="list name"
    fieldname="field"
    [output="name"] />
```

Parameters

listname (required)

Name of the list to be retrieved.

fieldname (required)

Any column in the list to retrieve that columns value for the current row. In addition the following keywords may be specified for the field name:

- #curRow - retrieve the current row number
- #numCols - retrieve the number of columns in the list
- #numRows - retrieve the number of rows in the list

output (optional)

If this parameter is specified, the value of the request field in the list is set as a variable using the name specified.

Description

This tag retrieves the value of the specified field in a specified list.

errno

None.

Example

The following example retrieves the Color field from Mylist.

```
<object:list
classname="com.openmarket.framework.objects.AbstractObject"
arg1="obj03" list="list1"/>
<ics:listloop listname="list1">
    <ics:listget listname="list1" fieldname="id" output="id"/>
    <ics:getvar name="id"/><br/>
</ics:listloop>
```

See Also

[ics:copylist](#)

ics:listloop

Iterates a list.

Syntax

```
<ics:listloop  
    listname="some list"  
    [maxrows="number of loops"]  
    [startrow="start row"]  
    [endrow="end row"]/>
```

Parameters

listname (required)

Name of the list to iterate.

maxrows (optional)

Maximum iterations the loop should be executed. If maxrows is omitted, the loop iterates to the end of list.

startrow (optional)

Row number to start the iteration. If omitted, ics:listloop begins iterating from the current row.

endrow (optional)

Row number of the row to end the iteration. If omitted, the iteration continues to the end of list.

errno

The value of errno is unchanged.

Example

```
<ics:listloop listname="MyList" startrow="5" maxrows="30">  
<ics:listget listname="MyList" fieldname="Color"/>
```

See Also

[ics:if](#)

[object:stream](#)

ics:literal

Formats literals for [ics:callsql](#) and other tasks beyond those for which [ics:sqlexp](#) is designed.

Syntax

```
<ics.literal  
    table="table name"  
    column="column name"  
    string="string"  
    output="variable name" />
```

Parameters

table (required)

Name of table.

column (required)

Name of column in the table.

string (required)

String that needs to be formatted into a literal for the specified column.

output (required)

Variable name where the literal is placed.

errno

The possible values of errno include:

Value	Description
-100	General exception error.
-103	Invalid table name.
-106	Invalid column name.
-10004	Missing required parameter.

Example

The following query, named myquery, exists in the SystemSQL table.

```
SELECT * FROM Name WHERE firstname = Variables.firstliteral
```

The query is called in an element as follows:

```
<ics:literal
  table="Name"
  column="firstname"
  string="Bubba"
  output="firstliteral"/>
```

```
<ics:clearerrno/>
```

```
<ics:callsql
  qryname="myquery"
  list="mylist"/>
```

In this example, Bubba is the string to be converted to a literal. The string is formatted according to the properties set for the database.

Release Introduced

This tag was introduced in CSEE release 5.0.

ics:logmsg

Writes strings to a Content Server log file for debugging or system monitoring.

Syntax

```
<ics:logmsg  
    msg="text" />
```

Description

Use this tag if you want to perform debugging or system monitoring. The log strings are written to your Content Server log file.

errno

The value of errno is unchanged.

Example

```
<ics:if condition='<%ics.GetErrno() != 0%>'  
<ics:then>  
    <ics:logmsg str='<%=ics.GetErrno()%>' />  
</ics:then>  
</ics:if>
```

ics:mirror

The ics:mirror tag mirrors one or more mirrorlists.

Syntax

```
<ics:mirror  
    target="target server"  
    username="target username"  
    password="target password"  
    cgiprefix="target CGI prefix"  
    targetinfile="name of target's ini file"  
    port="target's port number"  
    useproxy="true|false"  
    secure="true|false"  
    httpversion="true|false"  
    outputvariable="http variable"/>
```

Parameters

target (required)

Name or IP address of the target server.

username (required)

Username for the Content Server installation on the target server.

cgiprefix (required)

CGI prefix of the target server. If the CGI prefix is not specified, the remote server uses the same CGI prefix as the source machine.

targetinfile (required)

List of property files separated by semicolons to merge on the target server. If this parameter is not specified, the default property file is used.

port (required)

TCP/IP port for communication with the target server. If this parameter is not specified, the system defaults to port 80.

useproxy (required)

Set this parameter to `true` to communicate via a proxy server or `false` otherwise. If not specified, a proxy server is used.

secure (required)

Set this parameter to `true` to encrypt and transmit the mirror communication using the `https` protocol. If this parameter is not specified, the system defaults to unencrypted `http` to communicate with the target server.

httpversion (required)

Specifies the `http` protocol version to be used to communicate with the target server. Content Server supports versions 1.0 or 1.1. If this parameter is not specified the system defaults to `http 1.0`.

outputvariable (required)

Name of the variable to store the `http` response from the target server. This variable may then be used to display informational or error messages.

Description

The `ics:mirror` tag mirrors one or more mirrorlists. Each mirrorlist is declared using the `ics:mirrorlist` tag.

errno

None.

Example

The following example mirrors all rows of the table called `MyTable` to a server called `OtherMachine` using port 7001.

```
<ics:selectto listname="list1" table="MyTable" what="*" />
<ics:mirror target="OtherMachine" port="7001" cgiprefix="/servlet"
            username="ContentServer" password="FutureTense">
  <ics:mirrorlist listname="list1" />
</ics:mirror>
```

ics:mirrorlist

Specifies a list name to mirror.

Syntax

```
<ics:mirrorlist  
    listname="list name"/>
```

Parameters

listname (required)
Name of the list to be mirrored.

Description

The ics:mirrorlist tag is used as a parameter for the ics:mirror tag. This tag specifies a list name for the ics:mirror tag.

errno

None.

Example

The following example mirrors a list called names.

```
<ics:mirrorlist listname="names"/>
```

See Also

[ics:mirror](#)

[ics:mirrortreenode](#)

ics:mirrortreenode

Specifies a subtree to be mirrored.

Syntax

```
<ics:mirrortreenode  
    name="tree"  
    node="node"/>
```

Parameters

name (required)
Name of the tree.
node (required)
Name of the node to be mirrored.

Description

This tag specifies a subtree to be mirrored. The ics:mirrortreenode is a parameter tag for the ics:mirror tag.

errno

None.

Example

The following example mirrors the status node from the inventory tree.

```
<ics:mirrortreenode name="inventory"
                    node="status"/>
```

See Also

[ics:mirror](#)

[ics:mirrorlist](#)

ics:popvars

Saves a copy of all the current variables to a stack. The current variable state is unchanged.

Syntax

```
<ics:popvars/>
```

Parameters

None.

errno

None.

Example

```
<ics:setvar name="ArticleID" value="99202122"/>
<ics:pushvars/>
<ics:setvar name="ArticleID" value="224212"/>
<ics:callelement element="DisplayArticle"/>
<ics:popvars/>
```

See Also

[ics:pushvars](#)

ics:pushvars

Pops the variable stack, returning the variables to their previously stacked state. If the stack is empty, `ics:popvars` does nothing.

Syntax

```
<ics:pushvars/>
```

Parameters

None.

errno

None.

Example

```
<ics:setvar name="ArticleID" value="99202122"/>
<ics:pushvars/>
<ics:setvar name="ArticleID" value="224212"/>
<ics:callelement element="DisplayArticle"/>
<ics:popvars/>
```

See Also

[ics:popvars](#)

ics:removessvar

Deletes a session variable.

Syntax

```
<ics:removessvar
  name="ssvar name"/>
```

Parameters

name (required)

Name of the session variable to remove.

Description

The `ics:removessvar` tag deletes a session variable. After a session variable is removed, it is no longer defined for the session and its value cannot be referenced.

errno

The value of `errno` is unchanged.

Example

The following example removes the session variable `userpref`:


```
<ics:removessvar name="userpref"/>
```

See Also

[object:stream](#)

ics:removevar

Deletes a variable.

Syntax

```
<ics:removevar
    name="variable name"/>
```

Parameters

name (required)
Name of the variable to remove.

Description

The `ics:removevar` tag deletes a variable. After a variable is removed, it is no longer defined for the processing of the page and its value cannot be referenced. For more information about variables, see the *CSEE Developer's Guide*.

errno

The value of `errno` is unchanged.

Example

The following example removes the variable "myvar":

```
<ics:removevar name="myvar"/>
```

See Also

[ics:setvar](#)

ics:resolvevariables

Resolves Content Server multiple variables.

Syntax

```
<ics:resolvevariables
    name="variable name"
    [output="variable name"]
    [delimited="true|false"] />
```

Parameters

name (required)
String with embedded variable references.

output (optional)

Output variable name. If this parameter is not specified, the results are streamed.

delimited (optional)

true indicates new syntax for resolving variables (see the following Description section); false indicates the old syntax.

Description

The new syntax for resolving variables uses a \$(suffix and a) prefix. For example, Variables.pagename becomes \$(Variables.pagename). This allows you to resolve variables that may be a prefix of another variable name. For example, Variables.listname.column becomes \$(\$(Variables.listname).column) to indicate that listname is in the variable.

errno

The possible values of errno are:

Value	Description
-106	Invalid variable references.
-10004	A required parameter is missing.

Example

The following example resolves a variable called Variables.Year and streams the result to the browser.

```
<ics:resolvevariables name="Variables.Year"/>
```

This example resolves the variables Variables.lastname and Variables.firstname then streams the result to the browser.

```
<ics:resolvevariables
name="Variables.lastname,Variables.firstname"/>
```

See Also

[ics:getvar](#)

[ics:if](#)

[ics:getproperty](#)

[ics:listget](#)

ics:search

Searches an index based on a query.

Syntax

```
<ics:search
    [index="index name"]
    searchterm="search string"
    listname="output list"
    queryparser="parser name"
    relevanceterm="relevance term"
    [maxresults="max number of results"]
    [charset="character set"]
    [searchengine="name"]/>
```

Parameters

index (optional)

Name of the index to search. Default is specified in the Content Server properties file.

searchterm (required)

Query to submit. This should be in the format the given search engine understands and may also depend on the queryparser value.

listname (required)

Name of the output list. Columns returned are:

- **entry** - Name of the index entry that matches the search criteria.
- **detail** - Details of the index entry that matches the search criteria.
- **date** - Date the entry was added or the date specified when the index was added. Format is in Java SQL.
- **relevance** - Relevance value associated with the search result. The closer the value is to 1, the more useful and relevant the search result is likely to be.

queryparser (optional for Verity)

The AltaVista search engine supports three search types—Simple, Advanced, and Combined. Advanced is the default, and was previously the only one supported.

The simple search syntax uses optional + or - signs on a word or phrase to indicate that it is required or prohibited in the search results. The Advanced syntax uses Boolean operators (such as and, or, not, near) to constrict complex queries. In both cases, the query is passed in the searchterm parameter. The relevanceterm parameter is ignored.

The combined syntax uses both types of syntax. A Boolean query is specified by the searchterm parameter and a simple query is specified by the relevanceterm parameter to determine the ranking order of the results.

The Verity search engine supports three query parsers—Simple, FreeText, and BoolPlus. The optional queryparser tag tells the Verity search engine the syntax of the searchterm. If this argument is omitted, then the Simple parser is used by default. The parser may be specified by the `verity.parser` property in the `futuretense.ini` file. The choice of parser determines the syntax and form of the query

relevanceterm (optional for AltaVista)

Boolean term to be used for computing relevance ranking.

maxresults (optional)

Maximum number of results to return to the list.

charset (optional)

Constant value representing the character set the index uses.

For the AltaVista search engine this value can be 0, 1, or 2 (ISO_LATIN1, ASCII8, UTF8). If you do not specify `charset`, Content Server uses the value of `av.charset` from the Content Server properties file.

Because Java Strings are transformed to UTF8, as they pass through the JNI interface to AltaVista, the 0 value only works for 7-bit ASCII. For all other cases, use the value for UTF8.

For the Verity search engine, this value specifies the name of the subdirectory of the common directory where the locale is defined. If you do not specify `charset`, Content Server uses the value of `verity.charset` in the Content Server properties file.

searchengine (optional)

Name of the search engine to use. If `searchengine` is not specified, Content Server uses the value of `cs.searchengine` from the Content Server properties file.

Description

The `ics:search` tag searches an index based on a query. The resultset of the query is stored in a list.

errno

The possible values of `errno` include:

Value	Description
-101	No search results.
-800	Bad search type.
-801	Cannot load search engine.
-802	Unsupported search function.
-805	No default index specified.
-806	Unknown search engine.
-809	Search failed.
-810	Bad character set.
-811	Could not call native method.
-812	Index does not exist.

Example

```
<ics:clearerrno/>
<ics:search index="c:/articleidx" list="headlist"
what="headline:Clinton" searchengine="AV"/>
```

```

<ics:if condition='<%ics.GetErrno() == 0%>'>
<ics:then>
  <% IList il = ics.GetList("headlist", false);%>
  <ics:listloop listname="headlist">
    <ics:selectto table="Article" what="*"
where='<%=il.getValue("id")%' list="thearticle"/>
    <ics:if condition='<%ics.GetErrno() == 0%>'>
      <ics:then>
        <ics:listget listname="Article" fieldname="headline"/><br>
        <ics:listget listname="Article" fieldname="byline"/><br>
      </ics:then>
    </ics:if>
  </ics:listloop>
</ics:then>
</ics:if>

```

Note

This example requires that the database be accessed for each article. This can be inefficient, especially in cases where the search results are to be displayed only so that the one element required may be selected for further use. In this case, you should use the `DETAIL` field of the search result list. When entries are added to the index, you should set the `detail` parameter to some useful text string. For this example, it could be set to the article title so that you can then display and use it to select the article of interest.

See Also

[ics:indexcreate](#)

ics:searchdatetonative

Converts the Java SQL-formatted date to the date format native to the search engine.

Syntax

```

<ics:searchdatetonative
  date="Java date"
  outputvariable="variable name"
  [searchengine="name"]/>

```

Parameters

date (required)

Java SQL date to convert.

outputvariable (required)

Name of the variable to store the converted date string.

searchengine (optional)

Name of the search engine to use. If `searchengine` is not specified, Content Server uses the value of `cs.searchengine` from the Content Server properties file.

Description

The `ics:searchdatettonative` tag converts the Java SQL-formatted date to the date format native to the search engine. Use in the `WHAT` parameter of a `ics:search` tag. Dates in queries must be in a format that the search engine can parse.

The input to this tag (the `date` parameter) must be a valid Java SQL date. Content Server does not detect certain errors in the date format.

The Verity search engine handles most date formats, including the Java SQL format. If you are using Verity, you do not need to call `ics:searchdatettonative`.

errno

The possible values of `errno` include:

Value	Description
-801	Cannot load search engine.

Example

The following example converts the date in Java SQL format to the AltaVista date format and sets the results in the variable `AVDate`.

```
<ics:searchdatettonative date="1998-05-22" outputvariable="AVDate"
searchengine="AV"/>
```

```
<ics:if condition=<%ics.GetErrno() != 0%>i>
<ics:then>
  Report error
</ics:then>
<ics:else>
<ics:search index="c:/articleidx" list="articledatelist"
what='articledate:<%=ics.GetVar("AVDate")%>' />
</ics:else>
</ics:if>
```

ics:selectto

Executes a simple query from a table.

Syntax

```
<ics:selectto
    table="table name"
    [what="what criteria"]
    listname="output list"
    [where="where criteria"]
    [orderby="orderby expression"]
    [limit="max number of results"]/>
```

Parameters

table (required)

Name of the Content Server table to query. The table must be registered in the SystemInfo table. This table verifies security and caches result sets when the `cc.cacheResults` property is enabled.

what (optional)

List of columns to return. This parameter accepts a comma-separated list of column names.

listname (required)

Name of the list to contain the resultset.

where (optional)

Comma-separated list of table columns to select against. For each item in the *where* string, the value of the corresponding variable is used to construct a SQL *where* clause. All the elements in the comma-separated list **must** be variables. When you specify a datetime field as the *where criteria*, Content Server uses the `cc.datepicture` property to convert the supplied datetime to the proper format.

orderby (optional)

Orders the rows returned by the query. The format of *orderby* is a comma-separated list of column names. You can also override the default sort order by specifying ascending/descending with a (potentially) database specific command like:

```
orderby="colname1 desc,colname2 asc"
```

limit (optional)

Maximum number of rows to return. Note that if the `limit` parameter is omitted or is less than or equal to 0, then no limit exists and the entire recordset is returned.

Description

The `ics:selectto` tag executes a simple query from a table. The results are stored in a list.

errno

The possible values of `errno` include:

Value	Description
-101	No search results.
-106	One of the elements in the WHERE clause was not a variable. (All elements in the WHERE clause must be variables.)

Example

The following example selects all columns from the `SystemInfo` table and puts the results into a list called `RS` for later use.

```
<ics:selectto table="SystemInfo" what="*" listname="RS" />
```

The next example demonstrates a comma-separated list. It selects records in the `visitor` table where `firstname = Fred` and `lastname = Smith`. It then loops through the list if the list is created and has records.

```
<ics:setvar name="firstname" value="Fred"/>
<ics:setvar name="lastname" value="Smith"/>
<ics:setvar name="tablename" value="visitor"/>
<ics:selectto table="<%=ics.GetVar["tablename"]%>" what="*"
listname="VisitorList" where="firstname, lastname"/>
<ics:if condition='<%=ics.GetErrno() != -101%>'>
<ics:then>
  <ics:listloop listname="VisitorLst">
    FOUND MATCH:<ics:listget listname="VisitorList"
fieldname="firstname"/>
    <ics:listget listname="VisitorList" field="lastname"/>
    <br>
  </ics:listloop>
</ics:then>
<ics:else>
  No records found.
</ics:else>
</ics:if>
```

See Also

[ics:callsql](#)

[ics:sql](#)

ics:sendmail

Sends an SMTP mail message to one or more recipients.

Syntax

```
<ics:sendmail
    to="users"
    [replyto="address"]
    subject="subject"
    [body="message"]
    [contenttype="type"]/>
```

Parameters

to (required)

Comma-separated list of the recipients to receive the message

replyto (optional)

E-mail address that you want replies to go to. If a value is not specified, replies go to the sender's address.

subject (required)

Subject of the message.

body (optional)

Body of the e-mail message.

contenttype (optional)

Type of content you are sending. The default is plain/text.

Description

The SendMail tag sends an SMTP message to one or more recipients. To send mail, the Content Server properties file must contain valid values for the `cs.emailhost` and `cs.emailreturnto` properties. To receive mail, the Content Server properties file must contain valid values for the `cs.emailhost`, `cs.emailaccount`, and `cs.emailpassword` properties.

errno

The possible values of `errno` include:

Value	Description
-202	Cannot send mail.

Example

The following example sends an e-mail message to `joeuser` and `janeuser`. The syntax of all parameter values must comply with SMTP syntax.

```
<ics:sendmail to="joeuser,janeuser"
    subject="Test Message"
    body="Congratulations on becoming a member."/>
```

See Also

[ics:sendmail](#)

ics:setssvar

Sets the value of a session variable.

Syntax

```
<ics:setssvar name="ssvar name"
              value="ssvar value"/>
```

Parameters

name (required)

Name of the session variable to set.

value (required)

Value of the session variable.

Description

The `ics:setssvar` tag sets the value of a session variable. The value of the session variable exists for the duration of the session unless it is explicitly deleted using `ics:removessvar`.

errno

The value of `errno` is unchanged.

Example

```
<ics:setssvar name="color"
              value="red"/>
```

See Also

[ics:setvar](#)

[ics:removessvar](#)

ics:setvar

Sets the value of a variable.

Syntax

```
<ics:setvar
    name="variable name"
    value="variable value"/>
```

Parameters

name (required)

Name of the variable to set.

value (required)

Value to which to set the variable.

Description

The `ics:setvar` tag sets the value of a variable. The value of the variable exists for the duration of the page evaluation unless it is explicitly deleted using `ics:removevar`. For more information about variables, see the *CSEE Developer's Guide*.

errno

The value of `errno` is unchanged.

Example

The following example sets the variable "color" to "red".

```
<ics:setvar name="color" value="red"/>
```

The next example assumes that the variable "color" has been set to "red" and appends "and blue".

```
<ics:setvar name="color" value="blue"/>
```

The browser displays:

red and blue

If a variable was set by loading a file or URL and the content contains `Variables.varname` syntax, then the variable values can be resolved by:

```
<ics:setvar name="myvar" value="myvar"/>
```

Any variables as part of the value of `myvar` are resolved before the value is reset.

See Also

[ics:removevar](#)

[object:stream](#)

ics:sql

Executes an inline SQL statement.

Syntax

```
<ics:sql
    sql="sql commands"
    listname="list name"
    table="name of table"
    [limit="max number of results"]/>
```

Parameters

sql (required)

SQL commands to execute.

listname (required)

Name of list to contain the resultset.

table (required)

Name of the Content Server table to query. The table must be registered in the SystemInfo table. This table verifies security and caches resultsets when the `cc.cacheResults` property is enabled. You may specify a comma-separated list of tables to cache or synchronize against.

limit (optional)

Maximum number of rows to be retrieved. If limit is omitted or if the limit value is set to less than or equal to 0, all records from the SQL command is returned.

Description

The `ics:sql` tag executes an inline SQL statement. It is the equivalent of the `EXECSQL` XML tag. The table specified by the `tablename` must exist in the database prior to calling `ics:sql`. The resultset is cached against the table indicated by the value of the `tablename` variable.

Note

If the limit is less than or equal to 0, then no limit exists and the entire recordset is returned.

errno

The possible values of `errno` include:

Value	Description
-101	No rows found.
-102	Empty column.
-103	No such table.
-104	No table definition.

See Also

[ics:callsql](#)

[ics:selectto](#)

ics:sqlexp

Passes a column name and a nexpression that contains the column name to be used as the left side in the where clause.

Syntax

```
<ics:sqlexp
    column="column name"
    output="variable name"
    string="string"
    table="table name"
    [type="logical operator"]
    verb="comparison operator"
    [expression="expression"]/>
```

Parameters

column (required)

Name of the column in the table.

output (required)

Name of the output variable.

string (required)

Value or values to compare against.

table (required)

Name of the table.

type (optional)

Logical operator to use between. Default is and.

verb (required)

Comparison operator.

expression (optional)

Column expression to be used instead of column name.

errno

The possible values of errno include:

Value	Description
-100	Exception error
-103	Invalid table name
-106	Bad parameter value
-406	Bad parameter value

Value	Description
-10004	Missing required attribute

Release Introduced

This tag was introduced in CSEE release 5.0.

ics:textargument

Argument for the ics:indexadd tag.

Syntax

```
<ics:textargument
  name="field name"
  value="value"/>
```

Parameters

name (required)

Name of the field.

value (required)

Text string to be added to the search index.

Description

The ics:textargument tag specifies a text field to be added to a search index created with the ics:indexadd tag. This tag is an argument for the ics:indexadd tag.

errno

None.

Example

The following example adds a record containing a text field called headline to the search index c:/searchidx.

```
<ics:clearerrno/>
<ics:indexadd index="c:/searchidx" entryname="A111112"
entrydetail="Ohio towns" searchengine="AV">
  <ics:textargument name="headline" value="Places I have visited
in Ohio"/>
  <ics:fileargument name="body" value="c:/temp/body.txt"/>
  <ics:dateargument name="articledate" value="1999-01-29"/>
</ics:indexadd>

<ics:if condition=<%ics.GetErrno() != 0%>'>
<ics:then>
  Report error
</ics:then>
</ics:if>
```

See Also

[ics:indexadd](#)

ics:then

Creates a conditional then clause.

Syntax

```
<ics:then>
...
</ics:then>
```

Description

The `ics:then` tag is used to create then clauses within a string.

errno

None.

Example

```
<ics:if condition='<%=ics.GetErrno() != 0%>'>
<ics:then>
  Report error
</ics:then>
</ics:if>
```

See Also

[ics:else](#)

[ics:if](#)

ics:queryevents

Performs a query for a list of registered events.

Syntax

```
<ics:queryevents
  list="list name"
  [name="event name pattern"]
  [type="event type"]
  [enabled="true|false"]/>
```

Parameters

`list` (required)
Name of the output list.

name (optional)

Name of the event.

type (optional)

Specifies if the event is a servlet event or an email event. Legal values are 1 to specify a servlet event and 2 to specify an email event.

enabled (optional)

Specifies whether the query runs on enabled events or disabled events. Legal values are `true` to query only enabled events or `false` to query only disabled events.

Description

This tag queries Content Server for a list of events that have been registered with using the `ics:emailevent` tag. The `ics:queryevent` tag returns a list with the following columns:

- Name
- Type
- Enabled
- Time
- Target (for application events)
- Recipient (for email events)
- Params (for application events)
- Filename (for email events)

errno

The possible values of `errno` include:

Value	Description
-203	Event error
-10004	A required parameter is missing.
-10010	Invalid parameter value

Example

The following example lists all events:

```
<ics:queryevents list="events" enabled="true"/>
<ics:listloop list="events">
<ics:getvar="events.name"/> <br/>
</ics:listloop>
```

Release Introduced

This tag was introduced in CSEE release 4.0.

See Also

[ics:readevent](#)

ics:readevent

Reads an event from storage.

Syntax

```
<ics:readevent
    list="list name"
    name="event name"/>
```

Parameters

list (required)
Name of the output list.

name (required)
Name of the event.

Description

This tag reads an event from storage. The `ics:readevent` tag returns a list with the following columns:

- Name
- Type
- Enabled
- Time
- Target (for application events)
- Recipient (for email events)
- Params (for application events)
- Filename (for email events)

errno

The possible values of `errno` include:

Value	Description
-203	Event error.
-10004	A required parameter is missing.
-10010	Invalid parameter value.

Example

The following example reads the email event `confirmation_email` from storage and places the results in a list called `events`.

```
<ics:readevent list="events" name="confirmation_email"/>
```

Release Introduced

This tag was introduced in CSEE release 4.0.

See Also

[user:su](#)

ics:treemanager

Calls the TreeManager servlet.

Syntax

```
<ics:treemanager>
  <ics:argument name="arg" value="value"/>
  [<cgiargument name="cgiarg" value="cgivalue"/>]
  [<argument name="scoped" value="global|local|stacked"/>]
</ics:treemanager>
```

Parameters

cgiargument (optional)

CGI arguments passed to the TreeManager.

scoped (optional)

Valid options for this attribute are as follows:

- GLOBAL - The default behavior
- LOCAL - Arguments passed in are restored (or destroyed) on return from the call.
- STACKED - The child only sees the explicit arguments passed in and can only return values in that same namespace.

Description

The `ics:treemanager` tag calls the TreeManager servlet. The GLOBAL value of the SCOPED parameter does not protect input variables. Use `cgiargument` to specify that specific CGI parameters be made available to EvalServer. Use `argument` for specific string values.

The following is a list of the commands that you use with the `ics:treemanager` tag.

addchild	addchildren	copychild	createtree
delchild	delchildren	deletetree	findnode
getchildren	getnode	getparent	listtrees
movechild	nodepath	setobject	validatenode
verifypath			

errno

The value of `errno` is unchanged.

See Also

[object:children](#)

addchild

Adds a node to a tree table using the `ics:treemanager` command.

Syntax

```
<ics:treemanager>
  <ics:argument name="ftcmd" value="addchild"/>
  <ics:argument name="treename" value="name"/>
  [<ics:argument name="parentid" value="node ID"/>]
  [<ics:argument name="otype" value="object table"/>]
  [<ics:argument name="oid" value="object ID"/>]
  [<ics:argument name="overversion" value="object version"/>]
  [<ics:argument name="ncode" value="string value"/>]
  [<ics:argument name="nrank" value="node rank"/>]
</ics:treemanager>
```

Parameters

ftcmd (required)

Value must be set to `addchild`.

treename (required)

Name of the tree.

parentid (optional)

Node ID of the parent node. Omit or set to `NULL` to make this node a top level node. Top level nodes are identified by a 0. Must be an integer.

otype (optional)

Name of an object table.

oid (optional)

Identifies the current object on this node. The `oid` should correspond to a value in the object table's `id` column.

overversion (optional)

Numeric value that specifies the version of the object. This parameter is not supported for this release of Content Server, and is reserved for use in later versions.

ncode (optional)

16-character string value that is available for use by the application.

nrank (optional)

Numeric value of the rank to assign this node.

Description

The `addchild` command adds a node to a tree table.

After executing the `addchild` command, a list named after the tree table or found in a variable called `treename` is created. A unique identifier for the node in the tree, the `nid` value, is automatically generated when you successfully add a node. The value of `nid` can be retrieved in the tree result list.

errno

The possible values of `errno` include:

Value	Description
-3	No access privileges to table.
-103	No table.
-105	Database error.
-106	Missing or invalid parameter.
-110	Invalid tree name.

See Also

The following `ics:treemanager` commands:

[addchildren](#)

[delchild](#)

addchildren

Adds multiple nodes to a tree table using the `ics:treemanager` command.

Syntax

```
<ics:treemanager>
  <ics:argument name="ftcmd" value="addchildren"/>
  <ics:argument name="treename" value="name"/>
  <ics:argument name="nparentidN" value="node ID"/>
  [<ics:argument name="otypeN" value="table name"/>]
  [<ics:argument name="oidN" value="object ID"/>]
  [<ics:argument name="oversionN" value="object version"/>]
  [<ics:argument name="ncodeN" value="string value"/>]
  [<ics:argument name="nrankN" value="node rank"/>]
</ics:treemanager>
```

Parameters

ftcmd (required)

Value must be set to `addchildren`.

treename (required)

Name of the tree.

nparentidN (required)

NID of the parent node. The value *N* corresponds to the row number to be added starting from 0 (zero). Top level nodes are identified by a zero value. Must be an integer. Unlike the `addchild` command, `nparentidN` must be provided and cannot be left NULL.

otypeN (optional)

Name of an object table. The value *N* corresponds to the row number to be added starting from 0 (zero).

oid N (optional)

Identifies the current object on this node. The `oid` should correspond to a value in the object table's `id` column. The value of N corresponds to the row number to be added starting from 0 (zero).

oversion N (optional)

A numeric value that specifies the version of the object. This parameter is not supported for this release of Content Server, and is reserved for use in later versions.

ncode N (optional)

16-character string value that can be used by an application. The value of N corresponds to the row number to be added starting from 0 (zero).

nrank N (optional)

Numeric value of the rank to assign this node. The value of N corresponds to the row number to be added starting from 0 (zero).

errno

The possible values of `errno` include:

Value	Description
-3	No access privileges to table.
-103	No table.
-105	Database error.
-106	Missing or invalid parameter.
-110	Invalid tree name.

See Also

The following `ics:treemanager` commands:

[addchild](#)

[createtree](#)

copychild

Copies one or more nodes from one tree to another tree using the `ics:treemanager` command.

Syntax

```
<ics:treemanager>
  <ics:argument name="ftcmd" value="copychild"/>
  <ics:argument name="treename" value="name"/>
  <ics:argument name="node" value="node ID"/>
  [<ics:argument name="newtreename" value="treename"/>]
  [<ics:argument name="nparentid" value="node ID"/>]
</ics:treemanager>
```

Parameters

`ftcmd` (required)

Value must be set to `copychild`.

`treename` (required)

Name of the source tree from which the nodes are copied.

`node` (required)

Node ID of the node to copy.

`newtreename` (required)

Name of the destination tree where the nodes is copied.

`nparentid` (optional)

Node ID of the new parent of the node in the destination tree (`newtreename`). Omit or set `NULL` to make this node a top-level node.

Description

The `copychild` command copies one or more nodes from one tree to another tree. Multiple nodes are copied if the node is the parent of one child or a number of children.

errno

The possible values of `errno` include:

Value	Description
-3	No access privileges to table.
-103	No table.
-105	Database error.
-106	Missing or invalid parameter.
-110	Invalid tree name.

See Also

The following `ics:treemanager` commands:

[getchildren](#)[movechild](#)

createtree

Creates a new tree table using the `ics:treemanager` command.

Syntax

```
<ics:treemanager>
  <ics:argument name="ftcmd" value="createtree"/>
  <ics:argument name="treename" value="name"/>
  [<ics:argument name="aclList" value="acl list"/>]
</ics:treemanager>
```

Parameters

ftcmd (required)
Value must be set to `createtree`.

treename (required)
Name of the tree.

aclList (optional)
ACLs that can access this tree.

Description

The `createtree` tag creates a new tree table. The system constructs a new table that represents a tree and registers it in the `SystemInfo` table as a tree table.

errno

The possible values of `errno` include:

Value	Description
-3	No access privileges to table.
-22	Table already exists.

See Also

The following `ics:treemanager` command:

[deletetree](#)

delchild

Deletes a node from a tree using the `ics:treemanager` command.

Syntax

```
<ics:treemanager>  
  <ics:argument name="ftcmd" value="delchild"/>  
  <ics:argument name="treename" value="name"/>  
  <ics:argument name="node" value="node ID"/>  
</ics:treemanager>
```

Parameters

ftcmd (required)
Value must be set to `delchild`.

treename (required)
Name of the tree.

node (required)
Node ID of the child to remove.

Description

The `delchild` command deletes a node from a tree. All child nodes are also be removed. The objects pointed to by the nodes are not affected.

errno

The possible values of `errno` include:

Value	Description
-3	No access privileges to table.
-103	No table.
-105	Database error.
-101	No rows returned.
-111	No nodes.

See Also

The following `ics:treemanager` commands:

[delchildren](#)

[addchild](#)

delchildren

Deletes multiple nodes from a tree using the `ics:treemanager` command.

Syntax

```
<ics:treemanager>
  <ics:argument name="ftcmd" value="delchildren"/>
  <ics:argument name="treename" value="name"/>
  <ics:argument name="nodeN" value="node ID"/>
</ics:treemanager>
```

Parameters

`ftcmd` (required)

Value must be set to `delchildren`.

`treename` (required)

Name of the tree.

`nodeN` (required)

Where N is 0 to the number of nodes that you want to delete.

Note that these parameters are the same as for `delchild`. Children are deleted from a single tree and the node parameter must be specified by counting from 0. See `addchildren` for more information.

Description

The `deletetchildren` command deletes multiple nodes from a tree. Child nodes of the nodes deleted are also removed. The objects pointed to by the nodes are not affected.

errno

The possible values of `errno` include:

Value	Description
-3	No access privileges to table.
-103	No table.
-105	Database error.
-110	Invalid tree name.
-111	No nodes.

Example

```
<ics:setvar name="errno" value="0"/>
<ics:treemanager>
  <ics:argument name="ftcmd" value="delchildren" />
  <ics:argument name="treename" value="TreetestTree" />
  <ics:argument name="node0" value="34253224" />
  <ics:argument name="node1" value="94589343" />
  <ics:argument name="node2" value="83473888" />
```

```

</ics:treemanager>
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <!--Handle error-->
  </THEN>
</IF>

```

See Also

The following ics:treemanager commands:

[delchild](#)

[addchildren](#)

deletetree

Deletes a tree table using the ics:treemanager command.

Syntax

```

<ics:treemanager>
  <ics:argument name="ftcmd" value="deletetree"/>
  <ics:argument name="treename" value="name"/>
</ics:treemanager>

```

Parameters

ftcmd (required)
Value must be set to deletetree.

treename (required)
Name of the tree.

Description

The deletetree command deletes a tree table. This does not affect objects referenced by the tree. The tree is deleted and is no longer registered in the SystemInfo table.

errno

The possible values of errno include:

Value	Description
-3	No access privileges to table.
-103	No table.
-110	Invalid tree name.

See Also

The following ics:treemanager commands:

[createtree](#)

findnode

Gets a list of nodes from a tree using the `ics:treemanager` command.

Syntax

```
<ics:treemanager>
  <ics:argument name="ftcmd" value="findnode"/>
  <ics:argument name="treename" value="name"/>
  [<ics:argument name="where" value="column names"/>]
  [<ics:argument name="column name" value="value"/>]
  [<ics:argument name="orderby" value="tree column"/>]
</ics:treemanager>
```

Parameters

ftcmd (required)

Value must be set to `findnode`.

treename (required)

Name of the tree.

where (optional)

Values of the column names containing data to use in constructing the query. Only variables whose column names appear in this list are used in constructing the query, and only columns referring to objects are allowed. If you specify the `where` parameter, you must specify the *column name* parameter.

Legal values are as follows:

- otype
- oid
- overversion

column name (optional)

If you specify the `where` parameter, you must specify this parameter to indicate the value of the column name. For example,

```
<ics:argument name="oid" value="10234562312">
```

orderby (optional)

Specifies the tree column in which to sort results.

Wildcards are **not** allowed in any of the variables being searched for. If a % is found in a variable whose column is specified as part of the search, it is ignored.

The name of the list that is constructed is stored in the value of the variable, `treename` (that is, `Variables.treename`).

Description

The `findnode` command gets a list of nodes from a tree. This is meant to perform a fast lookup of nodes referring to specific objects.

errno

The possible values of `errno` include:

Value	Description
-3	No access privileges to table.
-103	No table.
-110	Invalid tree name.
-111	No nodes.

See Also

The following `ics:treemanager` command:

[getchildren](#)

getchildren

Retrieves a list of child nodes from a tree using the `ics:treemanager` command, specifically when you want to retrieve the children of a particular node (specified by `nparentid`).

Syntax

```

<ics:treemanager>
  <ics:argument name="ftcmd" value="getchildren"/>
  <ics:argument name="treename" value="name"/>
  [<ics:argument name="orderby" value="tree column"/>]
  [<ics:argument name="where" value="column name"/>]
  [<ics:argument name="where2" value="SQL filter"/>]
  [<ics:argument name="depth" value="recursion depth"/>]
  [<ics:argument name="width" value="number of children"/>]
  [<ics:argument name="join" value="true|false"/>]
  [<ics:argument name="limit" value="rows returned"/>]
</ics:treemanager>

```

Parameters

`ftcmd` (required)

Value must be set to `getchildren`.

`treename` (required)

Name of the tree.

`orderby` (optional)

Specifies the tree column in which to sort results. Note that this order is valid only if `depth=1`.

where (optional)

Values of the column names containing data to use in constructing the query. Only variables whose column names appear in this list are used in constructing the query.

Legal values are: `nid`, `nparentid`, `nrank`, `ncode`, `otype`, `oid`, `oversion`.

For example, `where=nparentid`, `otype` uses the values in the variables `nparentid` and `otype` to build the query.

`nparentid` is **always** part of the query, if it is not specified in the where clause; or, if there is no `nparentid` variable set, the value of the tree root (0) is used.

Also, `otype` **must** be specified if the join is `true`.

where2 (optional)

Additional SQL filtering (must be well formed SQL).

depth (optional)

Defaults to 1. Limits the depth of recursion when scanning a tree. Illegal values are converted to 1. Specify -1 to remove depth restrictions on scan. Note below the impact of a deep recursive search. The depth parameter **must** be set to 1 if join is `true`.

width (optional)

Defaults to -1 (that is, no limit). Limits the number of children per parent that are returned in the list. Illegal values are converted to -1. If width is specified, it overrides the value of limit as they become interchangeable in a single depth tree. If depth is greater than 1, width, limit, and depth are all relevant in defining the values returned in the list.

join (optional)

`true/false` to join to an object table. Note that the object table holds the cache information. Defaults to `false`.

limit (optional)

The maximum rows returned in a list.

Description

The `getchildren` command retrieves a list of child nodes from a tree, specifically when you want to retrieve the children of a particular node (specified by `nparentid`). The parent node is not returned as part of the list. The corresponding object data for the nodes may be joined to the result if desired.

Wildcards are **not** allowed in any of the variables being searched for. If a % is found in a variable whose column is specified as part of the search, it is ignored.

Object tables that are joined to are required to be registered as type `obj` in the `SystemInfo` table in order to support the caching of resultsets that are created. You cannot join to an object table if the where clause does not specify a unique classtype; that is, the returned resultset is a combination of a `treenode` and a single object type. Specifying a join forces the depth to 1. You may only retrieve the joined results for the current node.

Scan is depth first. Recursive (`depth > 1`) scan is considerably more expensive than a flat scan. All tree scan results can be cached based on system configuration. Then name of the LIST is *value-of-treename*.

When depth is specified as a legal positive value greater than one, an additional column `depth` is added to the resulting list. The name of the LIST constructed is stored in the value of the `Variables.treename`.

Retrieves a list of child nodes from a tree, specifically when you want to retrieve the children of a particular node (specified by `nparent id`). The parent node is not returned as part of the list. The corresponding object data for the nodes may be joined to the result if desired.

errno

The possible values of `errno` include:

Value	Description
-3	No access privileges to table.
-103	No table.
-106	Missing or invalid parameters.
-110	Invalid tree name.
-111	No nodes.

See Also

The following `ics:treemanager` commands:

[findnode](#)

getnode

Retrieves a node from a table using the `ics:treemanager` command. The object data may be joined to the result.

Syntax

```
<ics:treemanager>
  <ics:argument name="ftcmd" value="getnode"/>
  <ics:argument name="treename" value="name"/>
  <ics:argument name="node" value="nodeid"/>
  [<ics:argument name="join" value="true|false"/>]
  [<ics:argument name="depth" value="recursion depth"/>]
</ics:treemanager>
```

Parameters

`ftcmd` (required)

Value must be set to `getnode`.

`treename` (required)

Name of the tree.

`node` (required)

ID of the node.

`join` (optional)

`true` or `false` to join the object table. Note that the object table holds the cache information.

depth (optional)

Defaults to 1. Limits the depth of recursion when scanning a tree. Illegal values are converted to 1. Specify -1 to remove depth restrictions on scan. Note below the impact of a deep recursive search. The depth parameter **must** be set to 1 if join is true.

Object tables that are joined to are required to be registered as type obj in the SystemInfo table in order to support the caching of resultsets that are created.

The name of the LIST constructed is stored in the value of the variables, `treeName` (that is, `Variables.treeName`).

errno

The possible values of `errno` include:

Value	Description
-3	No access privileges to table.
-103	No table.
-108	Node not input.
-110	Invalid tree name.

See Also

The following `ics:treemanager` command:

[findnode](#)

getparent

Gets the parent record of a given node using the `ics:treemanager` command.

Syntax

```
<ics:treemanager>
  <ics:argument name="ftcmd" value="getparent"/>
  <ics:argument name="treeName" value="name" />
  <ics:argument name="node" value="node ID" />
</ics:treemanager>
```

Parameters

ftcmd (required)

Value must be set to `getparent`.

treeName (required)

Name of the tree.

node (required)

ID of the node.

errno

The possible values of `errno` include:

Value	Description
-3	No access privileges to table.
-101	No rows. (If node is invalid.)
-103	No table.
-105	Database error.
-106	Missing or invalid parameters.
-110	Invalid tree name.
-112	The parent node does not exist or this node is a root node. Note that versions of Content Server earlier than 3.5 reacted to these errors by returning an <code>errno</code> of 0 (with an empty resultset 0 rather than an <code>errno</code> of -112).

listtrees

Retrieves a list of all the trees in the system using the `ics:treemanager` command.

Syntax

```

<ics:treemanager>
  <ics:argument name="ftcmd" value="listtrees"/>
  [<ics:argument name="treename" value="name of tree"/>]
</ics:treemanager>

```

Parameters

`ftcmd` (required)

Value must be set to `listtrees`.

`treename` (optional)

Name of the tree. If specified, this parameter causes tree filtering. For example, `treename=Article%` finds all the trees that begin with the word `Article`.

Description

The `listtrees` command retrieves a list of all the trees in the system. The list created is `SystemTrees` when invoked via the `ics:treemanager` tag. The list field names are from the fields in the `SystemInfo` table.

errno

The possible values of `errno` include:

Value	Description
-3	No access privileges to table.
-101	No rows returned. This may be because no trees match the requested filter.

See Also

The following `ics:treemanager` commands:

[createtree](#)

[deletetree](#)

movechild

Moves a node and its children to a new parent that may be in the same or a different tree table using the `ics:treemanager` command.

Syntax

```
<ics:treemanager>
  <ics:argument name="ftcmd" value="movechild"/>
  <ics:argument name="treename" value="name"/>
  <ics:argument name="node" value="node ID"/>
  <ics:argument name="newtreename" value="treename"/>
  <ics:argument name="nparentid" value="parent node ID"/>
</ics:treemanager>
```

Parameters

`ftcmd` (required)

Value must be set to `movechild`.

`treename` (required)

Name of the source tree.

`node` (required)

ID of the node to move.

`newtreename` (required)

Name of the new (destination) tree.

`nparentid` (optional)

ID of the new parent of the node (omit or NULL to move to root).

errno

The possible values of `errno` include:

Value	Description
-3	No access privileges to table.
-103	No table.
-105	Database error.
-106	Missing or invalid parameters.
-110	Invalid tree name.
-112	The parent node does not exist or this node is a root node. Note that versions of Content Server earlier than 3.5 reacted to these errors by returning an <code>errno</code> of 0 (with an empty resultset 0 rather than an <code>errno</code> of -112).

See Also

The following `ics:treemanager` command:

[copychild](#)

nodepath

Traverses the tree beginning at the parent of the current node, returning a list of all ancestor nodes within this same tree using the `ics:treemanager` command.

Syntax

```

<ics:treemanager>
  <ics:argument name="ftcmd" value="nodepath"/>
  <ics:argument name="treename" value="name"/>
  <ics:argument name="node" value="node ID"/>
</ics:treemanager>

```

Parameters

ftcmd (required)
Value must be set to `nodepath`.

treename (required)
Name of the tree.

node (required)
Node ID of the node.

Description

The `nodepath` command traverses the tree beginning at the parent of the current node, returning a list of all ancestor nodes within this same tree.

The list represents the path of the node backwards to its root node (that is, the first record added is the immediate parent of the node, the next row is that parent's parent, and so forth). It stops at a root node in the tree.

The node is not included in the returned data, only the parent's. An empty table/list can be returned if node is a top-level node; that is, if its parent is 0. The name of the list that is constructed is stored in the value of the `Variables.treeName`.

Note

This command does not span trees. In other words, the path to a node always is within the tree. If a node is pointed to by a node from another tree, it is not in the node path.

errno

The possible values of `errno` include:

Value	Description
-3	No access privileges to table.
-103	No table.
-105	Database error.
-110	Invalid tree name.

setobject

Sets an object on a node using the `ics:treemanager` command.

Syntax

```
<ics:treemanager>
  <ics:argument name="ftcmd" value="setobject"/>
  <ics:argument name="treeName" value="name"/>
  <ics:argument name="node" value="node ID"/>
  [<ics:argument name="otype" value="object table"/>]
  [<ics:argument name="oid" value="object ID"/>]
  [<ics:argument name="ncode" value="string value"/>]
  [<ics:argument name="validate" value="true|false"/>]
  [<ics:argument name="nrank" value="node rank"/>]
</ics:treemanager>
```

Parameters

`ftcmd` (required)

Value must be set to `setobject`.

`treename` (required)

Name of the tree.

`node` (required)

ID of the node.

`otype` (optional)

Name of the object table.

`oid` (optional)

Identifies the object in an object table.

`ncode` (optional)

16-character string available for use by an application.

`validate` (optional)

If `true`, the object is validated before being set (see also `validatenode`).

`nrank` (optional)

Value of rank to assign this node.

Description

The `setobject` command sets an object on a node. The `otype`, `oid`, and `ncode` parameters may be `NULL` if `validate` is `false`. If `validate` is `true`, the existence of the table named with the `otype` parameter, and the existence of a record identified by the `oid` parameter, is verified.

errno

The possible values of `errno` include:

Value	Description
-3	No access privileges to table.
-103	No table.
-105	Database error.
-106	Missing or invalid parameters.
-110	Invalid tree name.
-114	Missing class type (If <code>validate</code> is <code>true</code>).

validatenode

Verifies that the given node is valid using the `ics:treemanager` command.

Syntax

```
<ics:treemanager>
  <ics:argument name="ftcmd" value="validatenode"/>
  <ics:argument name="treename" value="name"/>
  <ics:argument name="node" value="node ID"/>
</ics:treemanager>
```

Parameters

`ftcmd` (required)

Value must be set to `validatenode`.

`treename` (required)

Name of the tree.

`node` (required)

ID of the node. Returns success or failure in the result code.

Description

The `validatenode` command verifies that the given node is valid. There must exist an object table with the `otype` of the node and there must be an object in that table with the `oid` of the node.

errno

The possible values of `errno` include:

Value	Description
-3	No access privileges to table.
-110	Invalid tree name.
-114	Missing class type (if <code>validate</code> is true).
-115	Unknown object class (if <code>validate</code> is true).

verifypath

Verifies that the given path of nodes matches the actual path, using the `ics:treemanager` command.

Syntax

```
<ics:treemanager>  
  <ics:argument name="ftcmd" value="verifypath"/>  
  <ics:argument name="treename" value="name"/>  
  <ics:argument name="npath" value="path"/>  
</ics:treemanager>
```

Parameters

ftcmd (required)
Value must be set to `verifypath`.

treename (required)
Name of the tree.

npath (required)
A string representing the path to the node.

Description

The `verifypath` command verifies that the given path of nodes matches the actual path. `verifypath` returns success or failure in the result code.

errno

The possible values of `errno` include:

Value	Description
-3	No access privileges to table.
-109	Tree path not valid.
-110	Invalid tree name.

localestring:argument

Declares a name/value pair that is available to the parent tag, [localestring:getstring](#).

Syntax

```
<localestring:argument name="name1" value = "value1" />
<localestring:argument name="namen" value = "valuen" />
```

Parameters

name (required)
Name of the argument.

value (required)
Value of the argument.

errno

None.

Release Introduced

This tag was introduced in CSEE release 5.0.

See Also

[localestring:getstring](#)

localestring:getstring

Returns a localized value for a string.

Syntax

```
<localestring:getstring
  name="name of string"
  locale="locale"
  [output="variable name"]
  [resolve="true"]>
<localestring:argument name="name1" value = "value1" />
<localestring:argument name="namen" value = "valuen" />
</localestring:getstring>
```

Parameters

name (required)
Defines string name. FatWire recommends using a naming convention to make further localization easier.

locale (required)
Five-character locale name; for example, EN_US.

output (optional)
If specified, a variable is created by the requested name with the locale-specific value; otherwise, the value is streamed.

`resolve` (optional)

If set to `true`, this parameter specifies that the localized value has variables that should be replaced using the rules for recursive Content Server variable replacement `$(Variables.name1)`.

errno

The possible values of `errno` include:

Value	Description
-103	Missing or invalid SystemLocaleString table.
-10004	Missing required attribute.

Release Introduced

This tag was introduced in CSEE release 5.0.

name:get

Retrieves the distinguished name of the user that is currently logged in.

Syntax

```
<name:get
  output="output variable"/>
```

Parameters

`output` (required)

Variable name to in which to return the distinguished name.

errno

When using the `name:get` tag, `errno` must always be set to zero.

Example

```
<name:get output="curruser"/>
```

See Also

[name:makechild](#)

name:makechild

Builds a distinguished name.

Syntax

```
<name:makechild
    [output="name"]
    context="parent name">
    <name:argument name="naming attribute" value="some value"/>
</name:makechild>
```

Parameters

output (optional)

Variable name in which the distinguished name is returned.

context (required)

Parent name to be used as a prefix for the newly constructed name.

name:argumentname (required)

Name of the naming attribute, such as cn or uid.

name:argumentvalue (required)

Value of the named attribute.

Description

This tag builds a distinguished name from the name of the parent and the least significant component of the child's name. The least significant component is specified using the `dir:argument` parameter. The name and value of the attributes are separated by the `syntax.speratorypeval.ini` property.

Errno

The possible values of `errno` include:

Value	Description
-15024	Required parameter missing

object:argument

Argument for other object tags.

Syntax

```
<object:argument  
    name="some name"  
    value="some value"/>
```

Parameters

name (required)
Name of the tag the argument is for.

value (required)
Value for this argument.

Description

The `object:argument` tag declares a name/value pair that is available to the parent tag. The parent tag must be present in the object tag library

errno

None.

Example

The following example creates an object table with the additional columns name and description.

```
<object:install  
    classname="com.openmarket.framework.objects.AbstractObject"  
    arg1="sample">  
    <object:argument name="name" value="VARCHAR(32) NOT NULL"/>  
    <object:argument name="description" value="VARCHAR(128) NOT  
NULL"/>  
</object:install>
```

See Also

[object:install](#)

[object:isempty](#)

[object:create](#)

object:checkin

Checks in an object.

Syntax

```
<object:checkin
    name="object name"
    [annotation="annotation"]
    [checkout="checkout info"]
    [flush="true|false"]/>
```

Parameters

name (required)

Specifies the name of the object in the object pool. A previous load command is required to bring the object in to the object pool.

annotation (optional)

Specifies an annotation description that should be included with the checkin.

checkout (optional)

Specifies if the object should remain checked out. The default is `false`, and does not leave the object checked out.

flush (optional)

Specifies that upon success, the object should be flushed from the object pool. A `true` value indicates that object should be flushed. The default is value if `false`.

Description

This tag checks an object in to storage. The object must have been checked out and loaded in to the object pool previously for this tag to function properly. Successful execution of this tag changes object's revision tracking state from `checked out` to `checked in`.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-10012	Object is not checked out.
-10014	Object already checked out.
-10016	Unable to check in specified object.

Example

The following example checks in an object called `object1`. The example provides an annotation stating that the object's description has been changed. In addition to values that specify the object is no longer checked out, and to flush the object from the object pool.

```
<object:checkin name="object1"
                annotation="changed description"
                checkout="false"
                flush="true"/>
```

See Also

[object:checkin](#)

[user:su](#)

object:checkout

Checks out an object.

Syntax

```
<object:checkout
    classname="classname"
    objecttype="object type"
    objectid="object ID"
    [objectversion="object version"]/>
```

Parameters

objecttype (required)

Type of object to be checked out.

objectid (required)

Content Server ID of the object to be checked out.

objectversion (optional)

Version of the object to be checked out.

Description

This tag checks out an existing object in the revision tracking system.

errno

Value	Description
-10001	The implementing class is invalid.
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-10006	The object ID is not valid.

Value	Description
-10007	The version of the object is not valid.
-10011	Database table is not revision tracked.
-10013	Object is checked out by another user.
-10014	Object is already checked out
-10015	Unable to check out specified object.
-10019	Database table is not an object table.

Example

The following example checks out an object called sample from the revision tracking system.

```
<object:checkout
  classname="com.openmarket.framework.objects.AbstractObject"
    objecttype="sample"
    objectid='<%=ics.GetVar("id")%>' />
```

See Also

[object:checkin](#)

[user:su](#)

object:children

Retrieves children of a particular node.

Syntax

```
<object:children
  name="name of node"
  list="list"
  [objecttype="object type"]
  [objectid="object ID"]
  [objectversion="object version"]
  [code="selected code"]
  [order="name1, name2, name3"] />
```

Parameters

name (required)

Name of the parent node.

list (required)

Specifies the name of the list in the list pool. This is the same list pool used by other Content Server tags that generate lists. If a list exists with the same name, it is replaced by the new list.

objecttype (optional)

Specifies a requested object type for the child nodes. It can be used in conjunction with the `objectid` and `objectversion` parameters to request a specific object, or it can be used independently to request all objects of a specific class.

objectid (optional)

Specifies a requested object identifier for the child nodes. It should be used in conjunction with the `objecttype` parameter, to specify the class of object.

objectversion (optional)

Specifies a requested object version for the child nodes. This parameter is typically used to uniquely identify an object, when used in conjunction with the `objecttype` and `objectid` parameters.

code (optional)

Allows for selection of child nodes based on the code associated with the node.

order (optional)

Specifies a comma-separated list of field names that should be used to order the list of objects.

Description

This tag builds a list of child node objects that meet specified criteria.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-10006	The object ID is not valid.
-10007	The version of the object is not valid.

Example

The following example retrieves only the child nodes from the `sample` object type from parent `node1`:

```
<object:children name="node1"
                 list="list1"
                 objecttype="sample"/>
```

See Also

[object:root](#)

[object:setobject](#)

[object:setparent](#)

object:create

Creates an object.

Syntax

```
<object:create
  classname="classname of object"
  name="name of object"
  [allocate="true|false"]
  [arg1="argument 1"]
  [arg2="argument 2"]/>
<object:argument name="name" value="value"/>
</object:create>
```

Parameters

classname (required)

Classname of the object to be created.

name (required)

Name of the object to be created.

allocate (optional)

When set to true, the tag allocates the object, and makes the object ID available before the object is saved to the database.

arg1 (optional)

Argument 1.

arg2 (optional)

Argument 2.

Description

This tag creates a new object instance of the implementing class, and saves it in the object pool using the specified name. The implementing class must implement a create method in order for an instance of the implementing class to be constructed. The create method should have one of the following prototypes:

- `public static IAbstractBase Create(ICS ics)`
- `public static IAbstractBase Create(ICS ics, String sArg1)`
- `public static IAbstractBase Create(ICS ics, String sArg1, String sArg2)`

errno

The possible values of errno include:

Value	Description
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.

Value	Description
-10004	A required parameter is missing.
-10010	Invalid attribute value

Example

The following example creates an instance of `object1` in memory to be saved in the sample table.

```
<object:create
  classname="com.openmarket.framework.objects.AbstractObject"
    name="object1"
    arg1="sample"
  <object:argument name="name" value="John Doe"/>
</object:create>
```

See Also

[object:gather](#)

[object:save](#)

[object:set](#)

object:deinstall

Deinstalls an object by deleting its supporting tables.

Syntax

```
<object:deinstall
  classname="some classname"
  [arg1="argument 1"]
  [arg2="argument 2"]/>
```

Parameters

`classname` (required)

Classname of the object to be deinstalled.

`arg1` (optional)

Argument 1.

`arg2` (optional)

Argument 2.

Description

This tag deletes the table(s) supporting the specified class with optional arguments.

errno

The possible values of `errno` include:

Value	Description
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing.
-10010	Invalid attribute value

Example

The following example deletes the object table called `sample`:

```
<object:deinstall
  classname="com.openmarket.framework.objects.AbstractObject"
  arg1="sample"/>
```

See Also

[object:install](#)

object:delete

Deletes an object from storage and the object pool.

Syntax

```
<object:delete
  name="object name"/>
```

Parameters

`name` (required)

Name of the object in the object pool to be deleted.

Description

This tag deletes a loaded object from storage and removes it from the object pool. The object to be deleted must exist in storage and the object pool.

errno

The possible values of `errno` include:

Value	Description
-105	Database error.
-10004	A required parameter is missing.

Value	Description
-10005	The requested object is not in the object pool (is not loaded into memory).

Example

The following example deletes `object1` from the object pool.

```
<object:delete name="object1"
classname="com.openmarket.framework.objects.AbstractObject"
objectid='<%=ics.GetVar("id"%>' arg1="sample"/>
```

See Also

[object:isempty](#)

object:gather

Gathers fields into an object.

Syntax

```
<object:gather
    name="object name"
    prefix="prefix"
    [fieldlist="list of field names"]/>
```

Parameters

name (required)

Name of the object in the object pool.

prefix (required)

Specifies the name of a string to be used as the prefix for variable names, which may exist in the environment. A colon (:) is used as a prefix separator, followed by a field name when the `object:gather` tag is looking for variables in the environment.

fieldlist (optional)

Specifies a comma-separated list of field names that should be gathered. Any field in the list that does not exist in the environment is cleared. If this parameter is not specified, then the `object:gather` tag looks at all fields in the environment. If they are not found the existing value is maintained.

Description

This tag retrieves name/value pairs from the environment, and sets them into a loaded object. The name/value pairs are grouped together using a prefix and a prefix separator.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

Example

The following example gathers the name and description fields from the environment into `object1`, using the prefix `sample`:

```
<object:gather name="object1"
               prefix="sample"
               fieldlist="name,description"/>
```

See Also

[portal:logdep](#)

[object:scatter](#)

[object:stream](#)

object:get

Copies a field from an object.

Syntax

```
<object:get
  name="object name"
  field="field"
  [output="output"]/>
```

Parameters

name (required)

Name of the object.

field (required)

Name or type of field to retrieve.

output (optional)

Specifies the name of the output variable, which should be created in the environment. If this parameter is not specified, the variable is given the same name as the field.

Description

This tag copies a field value from a loaded object into the environment.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

Example

The following example copies the name column into a variable called name:

```
<object:get name="object1"
           field="name"
           output="name"/>
```

See Also

[object:gather](#)

[object:scatter](#)

[object:stream](#)

object:install

Installs an object.

Syntax

```
<object:install
  classname="some classname"
  [dir="some directory"]
  [acl="some acl"]
  track="revision tracking true or false"
  [arg1="argument 1"]
  [arg2="argument 2"]/>
```

Parameters

classname (required)

Classname of the object.

dir (optional)

Directory of the object.

acl (optional)

ACL of the object.

track (optional)

This indicates if the table associated with the implementing class should be revision tracked.

arg1 (optional)

Argument 1.

arg2 (optional)

Argument 2.

Description

This tag creates an installation of the implementing class. This typically involves creating the catalogs to allow for storage of instances of the class. The installation method should use one of the following prototypes:

- `public static int Install (ICS ics, Vector vector, String sUploadDir, String sAcList, boolean bTrack)`
- `public static int Install (ICS ics, Vector vector, String sUploadDir, String sAcList, boolean bTrack, String sArg1)`
- `public static int Install (ICS ics, Vector vector, String sUploadDir, String sAcList, boolean bTrack, String sArg1, String Arg2)`

The argument in the second prototype is a vector of a column elements, represented by the `ColumnElement` class. These typically represent column definitions using the `ColumnDefinition` class.

errno

The possible values of `errno` include:

Value	Description
-10001	Implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing
-10010	Invalid attribute value.

Example

The following example creates an object table with the additional columns `name` and `description`.

```
<object:install
classname="com.openmarket.framework.objects.AbstractObject"
arg1 ="sample">
  <object:argument name="name" value="varchar(32)NOT NULL"/>
  <object:argument name="description" value="varchar (128)NOT
NULL"/>
</object:install>
```

See Also

[object:create](#)

[object:deinstall](#)

object:isempty

Determines whether a field in an object is empty.

Syntax

```
<object:isempty  
  name="name"  
  field="field or column name"  
  output="variable name" />
```

Parameters

name (required)

Name used when the object was created using `object:create` or loaded using `object:load`.

field (required)

Name of the field or column in the object.

output (required)

Name of output variable.

Description

The result of this tag (held in the output variable) is either the string “true” or the string “false.” If the tag results in an error, no output variable is created and `errno` is set.

errno

The possible values of `errno` include:

Value	Description
-10004	Missing required attribute.
-10005	No object by specified name.

Release Introduced

This tag was introduced in CSEE release 5.0.

object:list

Retrieves a list of objects

Syntax

```
<object:list
  classname="some classname"
  list="name of list"
  [order="list of field names"]
  [arg1="argument 1"]
  [arg2="argument 2"]/>
<object:argument name="name" value="value"/>
```

Parameters

classname (required)

Classname of the object.

list (required)

Specifies the name of the list in the list pool. This is the same list pool used by other Content Server tags that generate lists. If a list exists with the same name, it is replaced by the new list.

order (optional)

Specifies a comma separated list of names that should be used to order the list of objects.

arg1 (optional)

Argument 1.

arg2 (optional)

Argument 2.

Description

This tag builds a list of objects that meet the specified criteria. The criteria are a list of name/value pairs, which must be present in the object. The implementing class must implement a list method. The list method should have one of the following prototypes:

- `public static IList List (ICS ics, FTValList aArgument, String sOrder)`
- `public static IList List (ICS ics, FTValList aArgument, String sOrder, String sArg1)`

errno

The possible values of errno include:

Value	Description
-101	No rows returned.
-105	Database error.
-10001	The implementing class is invalid.

Value	Description
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing.
-10010	Invalid attribute value.

Example

The following example lists all objects in the sample table.

```
<object:list
  classname="com.openmarket.framework.objects.AbstractObject"
    list="list1"
    arg1="sample"/>
```

See Also

[ics:listget](#)

object:load

Loads an object.

Syntax

```
<object:load
  classname="classname"
  name="object name"
  [objectid="object id"]
  [objectversion="object version"]
  [checkout="checkout info"]
  [nodeid="node ID"]
  [field="field"]
  [value="object value"]
  [arg1="argument 1"]
  [arg2="argument 2"]
  [list="list name"] />
```

Parameters

classname (required)

Classname of the object.

name (required)

Name of the object.

objectid (optional)

ID of the object.

objectversion (optional)

Version of the object.

checkout (optional)

Checkout information for the object.

nodeid (optional)

Node ID for the object.

field (optional)

This parameter specifies the name of a field argument that can be used to uniquely identify an object. When using this parameter the, value parameter should also be specified to indicate the value of the specified field.

value (optional)

Specifies the value of a field for unique identification. The value parameter must be used in conjunction with the field parameter.

arg1 (optional)

Argument 1.

arg2 (optional)

Argument 2.

list (optional)

Name of existing list. This parameter allows you to load an object from an existing list rather than from the database.

Description

This tag creates a new instance of the implementing class, loads a specific object into the object instance, and saves the object instance in the object pool using the specified name. The object can be requested by referencing the following information:

- object identifier
- object version
- node object identifier
- key field and key value

The implementing class must implement a load method. The load method should have one of the following prototypes:

- `public static MyClass Load(ICS ics, Long lObjectId, Integer iObjectVersion, boolean bLock)`
- `public static MyClass Load(ICS ics, Long lObjectId, Integer iObjectVersion, boolean bLock, String sArg1)`
- `public static MyClass Load(ICS ics, Long lNodeId)`
- `public static MyClass Load(ICS ics, Long lNodeId, String sArg1)`
- `public static MyClass Load(ICS ics, String sField, String sValue)`
- `public static MyClass Load(ICS ics, String sField, String sValue, String sArg1)`

errno

The possible values of errno include:

Value	Description
-10026	No list by specified name exists.

Example

The following example loads object1 from the sample catalog.

```
<object:load
  classname="com.openmarket.framework.objects.AbstractObject"
  name="object1"
  objectid='<%=ics.GetVar("id"%>'
  arg1="sample"/>
```

The following example uses the optional list parameter.

```
<object:list
  classname="com.openmarket.framework.objects.AbstractObject"
  arg1="obj03"
  list="list1"/>
  <ics:listloop listname="list1">
    <object:load
      classname="com.openmarket.framework.objects.AbstractObject"
      arg1="obj03"
      list="list1"
      name="object1"/>
    <object:scatter name="object1"
      prefix="object1"/>
    id: <ics:getvar name="object1:id"/><br/>
  </ics:listloop>
```

See Also

[object:save](#)

object:rollback

Rolls back an object to a previous revision.

Syntax

```
<object:rollback
  classname="some classname"
  objecttype="type of object"
  objectid="ID of object"
  [objectversion="version of object"]
  revision="revision of object"/>
```

Parameters

classname (required)
Classname of the object.

objecttype (required)
Type of object.

objectid (required)
ID of the object.

objectversion (optional)
Version of the object.

revision (required)
Specifies the target revision number of the object for the roll back.

Description

This tag rolls back an object to a specified revision. The implementing class must implement a rollback method. The rollback method should have the following prototype:

```
public static int RollBack (ICS ics, String sObjectCatalog, Long
10objectid Interger iObjectVersion, int iRevision)
```

errno

The possible values of errno include:

Value	Description
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing.
-10006	The object ID is not valid.
-10007	The version of the object is not valid.
-10011	Table is not revision tracked
-10012	Object is not checked out

Value	Description
-10017	Unable to rollback specified object
-10019	Table is not an object table

Example

The following example rolls backs the sample object to revision1.

```
<object:rollback
  classname="com.openmarket.framework.objects.AbstractObject"
  objecttype="sample"
  objectid='<%=ics.GetVar("id")%>'
  revision="1"/>
```

See Also

[object:track](#)

[object:save](#)

object:root

Retrieves a list of root nodes.

Syntax

```
<object:root
  classname="classname"
  list="list"
  [objecttype="object type"]
  [objectid="ID of object"]
  [objectversion="version of object"]
  [code="code"]
  [order="order of names"]
  [arg1="argument 1"] />
```

Parameters

- classname** (required)
Classname of the object.
- list** (required)
Specifies the name of the list in the list pool.
- objecttype** (optional)
Type of object.
- objectid** (optional)
ID of the object.
- objectversion** (optional)
Version of the object.

code (optional)

Allows selection of nodes based on their associated code.

order (optional)

Specifies a comma separated list of field names that should be used to order the list of objects.

arg1 (optional)

Argument one.

Description

This tag builds a list of node objects that have no parent, and are therefore considered root nodes.

errno

The possible values of `errno` include:

Value	Description
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10006	The object ID is not valid.
-10007	The version of the object is not valid.

Example

The following example retrieves a list of root nodes from the `com.openmarket.framework.objects.AbstractObject` class.

```
<object:root
  classname="com.openmarket.framework.objects.AbstractObject"
  list="list1"
  arg1="Tree1"/>
```

See Also

[object:setobject](#)

[object:setparent](#)

object:save

Saves an object.

Syntax

```
<object:save  
    name="object name"  
    [flush="true|false"]/>
```

Parameters

name (required)

Name of the object to be saved.

flush (optional)

Specifies that upon success, the object should be flushed from the object pool. A `true` value indicates the object should be flushed. The default value is `false`.

Description

Saves a loaded object into storage. The object must have been previously created or loaded into the object pool. Saving an object either creates a new object in storage or updates an existing object in storage.

errno

The possible values of `errno` include:

Value	Description
-105	Database error.
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

Example

The following example saves `object1` and removes it from memory:

```
<object:save name="object1"  
    flush="true"/>
```

See Also

[object:delete](#)

[ics:listget](#)

[user:su](#)

object:scatter

Scatters a field from an object.

Syntax

```
<object:scatter  
    name="object name"  
    [prefix="prefix"]/>
```

Parameters

name (required)

Name of the object.

prefix (optional)

Specifies the the name of a string for the variable names that is created in the environment. The prefixes are separated from the field names with a colon (:) when creating variables in the environment.

Description

This command allows fields of a previously loaded object to be copied to the environment as variables with a specified prefix. This tag is the preferred alternative to using multiple `object:get` tags.

errno

The possible values of errno include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

Example

The following example creates a variable in the environment with the prefix `sample`:

```
<object:scatter name="object1"  
    prefix="sample"/>
```

See Also

[object:gather](#)

[portal:logdep](#)

object:set

Sets a field in an object.

Syntax

```
<object:set  
  name="object name"  
  field="field"  
  value="value"/>
```

Parameters

name (required)
Name of the object.

field (required)
Field to be set.

value (required)
Value to be set in field.

Description

This tag sets the value of a field in a previously loaded object.

errno

The possible values of errno include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

Example

The following example sets the value to red in the name field of object1.

```
<object:set name="object1"  
  field="name"  
  value="red"/>
```

See Also

[portal:logdep](#)

object:setobject

Sets an object in a node.

Syntax

```
<object:setobject  
    name="node name"  
    object="object"/>
```

Parameters

name (required)

Name of the node in which the object is placed.

object (required)

Name of the object.

Description

This tag sets the referenced object of a node object. Node objects are used with tree catalogs in the Content Server system. The node object must implement the IAbstractNode interface and the referenced object must implement the IAbstractObject interface.

errno

The possible values of errno include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

Example

The following example maps node1 (residing in a tree) to object1 from an object table.

```
<object:setobject name="node1"  
    object="object1"/>
```

See Also

[object:root](#)

[object:setparent](#)

object:setparent

Sets the parent of a node.

Syntax

```
<object:setparent  
    name="name of node"  
    parentnode="name of parent node" />
```

Parameters

name (required)

Name of the child node.

parentnode (required)

Name of the parent node.

Description

This tag sets the parent node of a node object. Node objects are used with tree catalogs in the Content Server system.

errno

The possible values of errno include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

Example

The following example sets parent1 as the parent node for the node object node1.

```
<object:setparent name="node1"  
    parentnode="parent1" />
```

See Also

[object:root](#)

object:stream

Streams the value of a field from an object.

Syntax

```
<object:stream
  name="object name"
  field="field"
  [resolve="true|false"]/>
```

Parameters

name (required)

Name of the object.

field (required)

Field to be used for streaming.

resolve (optional)

Indicates if variable references in the field should be resolved. The default value is false.

Description

This tag streams the field value of a loaded object.

errno

The possible values of errno include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

Example

The following example streams the name field from object1.

```
<object:stream name="object1"
  field="name"
  resolve="false"/>
```

See Also

[portal:logdep](#)
[object:scatter](#)

object:track

Tracks an object.

Syntax

```
<object:track
  classname="classname"
  [dir="directory"]
  [revisions="revision information"]
  [arg1="argument 1"]/>
```

Parameters

classname (required)
Classname of the object.

dir (optional)
Directory of the object.

revisions (required)
Revision information for the object.

arg1 (optional)
Argument 1.

Description

This tag activates revision tracking for a previously untracked table. The track method should have one of the following prototypes:

- `public static int Track(ICS ics, string sUploadDir, int iRevisions)`
- `public static int Track(ICS ics, string sUploadDir, int iRevisions, String sArgs1)`

errno

The possible values of errno include:

Value	Description
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing.

Example

The following example activates tracking for the sample table:

```
<object:track
  classname="com.openmarket.framework.objects.AbstractObject"
  dir="C:\Content\sample"
```

```
revisions="10"
arg1="sample"/>
```

See Also

[user:su](#)

[object:untrack](#)

object:undocheckout

Retracts the checkout of an object.

Syntax

```
<object:undocheckout
    name="some object name"
    [revert="true|false"]
    [flush="true|false"]/>
```

Parameters

name (required)

Name of the object.

revert (optional)

Specifies that the object should revert back to its last committed state. The default value `true` causes the object to revert back to its last committed state. Specifying the value `false` removes the checkout lock with out reverting the object back to the last committed state.

flush (optional)

Flushes the object from the object pool. A `true` value indicates the object should be flushed. The default value is `false`.

Description

This tag is used for retracting a checkout made on an object in the Content Server system. The object can be reverted to its last committed state, or have the checkout lock removed.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-10011	Database table is not revision tracked.
-10012	Object is not checked out
-10018	Unable to undo check out specified object

Example

The following example retracts the checkout for object1, reverts it to it the last committed state, and flushes it from memory.

```
<object:undocheckout name="object1"
                    revert="true"
                    flush="true"/>
```

See Also

[object:checkin](#)

[object:checkout](#)

object:untrack

Removes tracking on an object.

Syntax

```
<object:untrack
    classname="some classname"
    [arg1="argument 1"]/>
```

Parameters

classname (required)
Classname of the object

arg1 (optional)
Argument 1.

Description

This tag deactivates revision tracking on for a previously tracked table. The untrack method should have one of the following prototypes:

- `public static int Untrack (ICS ics)`
- `public static int Untrack (ICS ics, String sArgs1)`

errno

The possible values of errno include:

Value	Description
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing.
-10011	Database table is not revision tracked.
-10022	Unable to untrack specified object.

Value	Description
-10023	Invalid database table name.
-10024	User not allowed access for request.

Example

The following example deactivates revision tracking on the sample table.

```
<object:untrack
    classname="com.openmarket.framework.objects.AbstractObject"
    arg1="sample"/>
```

See Also

[object:track](#)

portal:logdep

Logs an article as a dependency on an item.

Syntax

```
<portal:logdep
    id="string"
    date="JDBC-formatted date" />
```

Parameters

id (required)

A string that uniquely identifies the item you want to record. For example, Article-9876835, where 9876835 is a unique ID for an asset of type Article.

date (required)

The SQL date format for the date and time when the item was last modified. If this parameter is left null, the current date and time are used.

Description

The name of the output variable created is `randval`.

errno

None.

Example

This example is taken from the New Portal example included with Content Server.

```
<ics:if condition='<%=ics.GetVar("articleid")!=null%>'>
<ics:then>
    <!-- get the article from the database -->
    <ics:setvar name='id' value='<%=ics.GetVar("articleid")%>' />
```

```

    <ics:selectto listname='articlelist' table='NewPortalArticle'
where='id'
limit='1' what='*'/>
    <portal:logdep>
        <portal:parameter name='id' value='<%=ics.GetVar("id")%>'/>
    </portal:logdep>

    <ics:if condition='<%=ics.GetList("articlelist",false)!=null &&
ics.GetList("articlelist",false).hasData()%>'>
    <ics:then>
        <!-- display the article -->
        <div class="articleHeadline">
            <ics:listget listname="articlelist" fieldname="headline"/>
        </div>
        <div class="articleSubHead">
            <ics:listget listname="articlelist"
fieldname="subheadline"/>
        </div>
        <div class="articleCopyright">
            <ics:listget listname="articlelist" fieldname="byline"/>
            <ics:listget listname="articlelist"
fieldname="creditline"/>
        </div>
        <div class="articleBody">
            <ics:listget listname="articlelist" fieldname="@urlbody"/>
        </div>
        <div class="articleCopyright">
            <ics:listget listname="articlelist"
fieldname="copyright"/>
        </div>
    </ics:then>
    <ics:else>
        Error fetching Article <%=ics.GetVar("articleid")%>
    </ics:else>
    </ics:if>
</ics:then>
<ics:else>
    No article id.
</ics:else>
</ics:if>

```

portal:randomval

Generates a random number between 1 and 10 to select an ad image for the banner.

Syntax

```
<portal:randomval
    minval="integer"
    maxval="integer"
    output="variable name" />
```

Parameters

minval (required)

Minimum integer value of the random number.

maxval (required)

Maximum integer value of the random number.

output (required)

Name of the variable in which to store the number generated.

errno

The possible values of errno include:

Value	Description
-2	minval is greater than maxval.

time:get

Retrieves the elapsed time.

Syntax

```
<time:get
    name="name"
    [output="output"]/>
```

Parameters

name (required)

Specifies a name that is associated with the initial time.

output (optional)

Specifies a variable name to use for output of the elapsed time. If this parameter is not specified, the elapsed time is streamed.

Description

This tag streams the elapsed time in milliseconds, from when the time was last set.

errno

The value of `errno` is unchanged.

Example

The following example streams the elapsed time from `time1` to present:

```
<time:get name="time1"
          output="elapsed"/>
<ics:getvar name="elapsed"/>
```

See Also

[user:su](#)

time:set

Names the current time.

Syntax

```
<time:set
      name="time" />
```

Parameters

`name` (required)

Specifies a name to be associated with the initial time.

Description

This tag associates the current time with a name.

errno

None.

Example

The following example saves the correct time as `time1`.

```
<time:set name="time1"/>
```

See Also

[portal:logdep](#)

user:add (Deprecated)

As of Content Server version 4.0, replaced by [object:create](#).

Syntax

```
<user:add
  username="username"
  password="user's password"
  acl="user's ACL"/>
```

Parameters

username (required)
Name of the user to be added.

password (required)
Password for the user.

acl (required)
ACL assigned to the new user.

Description

Adds a Content Server user to the system.

errno

The possible values of errno include:

Value	Description
-10004	A required parameter is missing.

Example

The following example adds a user to the Content Server system.

```
<user:add username="Whitney"
  password="rosebud"
  acl="xceleditor"/>
```

See Also

[user:modify \(Deprecated\)](#)
[user:delete \(Deprecated\)](#)

user:changepassword (Deprecated)

As of Content Server version 4.0, replaced by [dir:replaceattrs](#).

Syntax

```
<user:changepassword  
    password1="new password"  
    password2="new password" />
```

Parameters

password1 (required)

The user's new password.

password2 (required)

The new password confirmation.

Description

The `user:changepassword` tag changes the password of an existing user. This tag confirms that the password entries match, and then changes the password. When a user submits a form that changes a password, this tag verifies that change.

errno

The possible values of `errno` include:

Value	Description
-105	Database error
-10004	A required parameter is missing.

Example

The following example confirms the changes to the password of the current user.

```
<user:changepassword password1="newpw"  
    password2="newpwconfirm"/>
```

See Also

[ics:argument](#)

[user:modify \(Deprecated\)](#)

user:delete (Deprecated)

As of Content Server version 4.0, replaced by [object:delete](#).

Syntax

```
<user:delete
  username="username"/>
```

Parameters

username (required)
Name of the user to be deleted.

Description

The `user:delete` tag deletes a user from the Content Server system.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.

Example

The following example deletes a user named Whitney from the Content Server system.

```
<user:delete username="Whitney"/>
```

See Also

[ics:argument](#)

[user:modify \(Deprecated\)](#)

user:login

Log in as a particular user.

Syntax

```
<user:login
  username="username"
  password="user's password"/>
```

Parameters

username (required)
User's login name.

password (required)
User's password.

Description

The `user:login` tag allows you to log in as a particular Content Server user.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.
-10010	Invalid attribute value.

Example

The following example logs in a user named Whitney.

```
<user:login username="Whitney" password="rosebud"/>
```

See Also

[user:logout](#)

[user:su](#)

user:logout

Logs out the current user.

Syntax

```
<user:logout/>
```

Parameters

None.

Description

The `user:logout` tag logs out the current Content Server user.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.
-10010	Invalid attribute value.

Example

The following example logs a user out of the Content Server system.

```
<user:logout/>
```

See Also

[user:login](#)

[user:su](#)

user:modify (Deprecated)

As of Content Server version 4.0, replaced by [dir:replaceattrs](#).

Syntax

```
<user:modify
  username="username"
  [password="user password"]
  [acl="user ACL"]/>
```

Parameters

username (required)
Username of the user to be modified.

password (optional)
Password of the user to be modified.

acl (optional)
ACL of the user to be modified.

errno

The possible values of `errno` include:

Value	Description
-105	Database error.
-10004	A required parameter is missing.
-10010	Invalid attribute value.

Example

The following example modifies the ACL for the user.

```
<user:modify username="Whitney"
  acl="ContentEditor"/>
```

See Also

[object:children](#)

[user:delete \(Deprecated\)](#)

user:su

Log in as a particular user.

Syntax

```
<user:su
  username="user name"
  password="user's password"/>
```

Parameters

username (required)

Name of the user.

password (required)

User's password.

Description

Allows you to log in as a particular so you can make changes. This tag is useful when combined with the `user:modify` tag to make administrative user profile changes.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

Example

The following example logs in a user as Percy, changes the password, and logs out and back to the original user:

```
<user:su username="Percy"
  password="rosebud"/>
  <user:modify username="Percy"
    password="other"/>
</user:su/>
```

See Also

[user:login](#)

[user:logout](#)

Chapter 10

CS-Direct JSP Tags

This chapter describes the functionality and JSP syntax of the CS-Direct JSP object methods. Use this chapter as a reference for the object tags that you need to do the following:

- Retrieve instances of assets from the database, storing them as objects in memory, so that you can display their content to your site visitors by rendering it in on-line pages.
- Extract information about the design or structure of your on-line site so that you can display that information to your site visitors by rendering it in on-line pages.

This chapter contains the following sections:

- [Asset Tags](#)
- [Render Tags](#)
- [Site Plan Tags](#)

Asset Tags

Assets are the core of CS-Direct. Assets are objects that can be created, edited, inspected, deleted, duplicated, placed into workflow, searched for, checked out and in with revision tracking, previewed, and published. You use the `asset` tags to extract the information from the CS-Direct database about your assets that you want to display on your on-line site.

When you use an `asset:load` tag to retrieve an asset from the database, CS-Direct loads an instance of the asset into memory as an object. You then have access to that instance of the asset with the other `asset` tags until either the session is flushed (the root element exits) or the name that is used for the asset object is overwritten.

Assets can have parent-child relationships that are set up either through **named associations** or through **unnamed relationships**. Named associations are defined, asset-type-specific relationships represented as fields in the asset forms. Unnamed relationships are established, for example, when you select assets from the generic Select list boxes on the Page form.

CS-Direct stores information about assets in the following database tables:

- Primary database table that holds assets of its type. For example, each page asset has a row in the Page table and each article asset has a row in the Article table. These tables store information such as the asset's name, its object ID, who created it, which template it uses, and so on. You use the `asset` tags to extract information from these tables.
- `AssetRelationTree` table, if the asset has named associations or unnamed relationships with other assets. Use the `asset:children` tag to extract information from this table.
- `AssetPublication` table, which specifies which management **sites** (publications) give you access to the asset through CS-Direct's asset entry and edit forms. If the asset is shared between more than one site, there is a row entry for each **pubid**. A **pubid** is a unique value that identifies a site.
- `SitePlanTree` table, if the asset is a page asset. For information about this table, see [Site Plan Tags](#).

To create types of assets, you use the AssetMaker forms in the Content Server interface, accessed from the Admin tab. For more information about developing asset types, see the *dCS Developer's Guide*.

Note

Each JSP file containing CS-Direct JSP asset tags must have the following directive at the top:

```
<%@ taglib prefix="asset" uri="futuretense_cs/asset.tld" %>
```

asset:addchild

Designates an asset to be added as a child asset in the AssetRelationTree table when the `asset:save` tag is called.

Syntax

```
<asset:addchild
    name="asset object name"
    type="asset type"
    childid="ID"
    [code="association name"]
    [rank="integer"] />
```

Parameters

- name (required)**
Name of the asset instance object.
- type (required)**
Asset type of child asset. Stored in OTYPE field.
- childid (required)**
ID of child asset to add. Stored in OID field.
- code (optional)**
Association name. Stored in NCODE field.
- rank (optional)**
Integer indicating rank of child asset. Stored in NRANK field.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-10007	Non-numeric rank.
-10014	Asset cannot be its own child.
-10017	Non-numeric rank.

asset:canedit

Checks whether a publish operation, which prevents the specified asset from being edited, is in progress.

Syntax

```
<asset:canedit  
  name="asset name" />
```

Parameters

name (required)
Name of previously loaded asset.

Description

The `asset:canedit` tag checks the PubSession table for any running (status='R') publish operations. If the specified asset is approved for publishing in the target for any of those running publish sessions, the asset cannot be edited.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-12034	Publish in progress.

asset:childgather

Gathers information from variables about which child relationships in the AssetRelationTree table should be added, deleted, or updated for a loaded asset.

Syntax

```
<asset:childgather  
  name="asset instance object name"  
  prefix="prefix"  
  codelist="list name" />
```

Parameters

name (required)
Name of the asset instance object.

prefix (required)
Prefix of the variable name; the first part of the variable name.

codelist (required)

List of asset association names which serves both as the second part of the variables name and the value of the `ncode` field that are written to the `AssetRelationTree`.

Description

The `asset:childgather` tag uses variables whose name is created in either of the following ways:

- Concatenating the prefix argument, a colon (:), and a value from the `codelist` argument, followed by `:id` to define the child asset's ID.
- Concatenating the prefix argument, a colon (:), a value from the `codelist` argument, followed by `:asset type` to define the child asset's asset type. The `AssetRelationTree` is updated when the [asset:save](#) tag is called.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

asset:children

Queries the `AssetRelationTree` table and builds a list of child assets for the specified parent asset.

Syntax

```
<asset:children
  name="assetName"
  list="listName"
  [code="NameOfAssociation"]
  [objecttype="typeOfObject"]
  [objectid="objectID"]
  [order="nrank"]/>
```

Parameters

name (required)

Input. Name assigned to the parent asset whose children you want to list. This parent asset must be loaded and assigned a name before you can pass its name to this tag. (See [asset:load](#).)

list (required)

Input. Name to assign to the list that stores the results. The columns in the list match the columns in the `AssetRelationTree` table, and, if you use the `object type` parameter, the asset table. If a list with this name already exists, it is replaced with this new list.

code (optional)

Input. Restricts the list to include only the child assets that have the relationship (association) specified by this parameter. You can restrict the list by named associations or by unnamed relationships. This value comes from the `ncode` field of the `AssetRelationTree`.

For example, if you include `objecttype="Image"` and `code="MainImage"` for an article asset, CS-Direct lists only the image asset that is related to that article asset by the Main Image named association. Without the code entry, CS-Direct lists all the images associated with the article.

To list only the child assets that are associated with the parent asset by unnamed relationships, use a hyphen (-). For example: `code= "-"` If you do not specify `objecttype` or `code`, the list includes all children with named associations or unnamed relationships to the parent asset.

objecttype (optional)

Input. Restricts the list of children by asset type. If you supply an `objecttype`, the list of children is a join of the `AssetRelationTree` and the asset table for the type specified.

To request a specific child asset, combine this parameter with `objectid`. Note that if you include an `objectid`, `objecttype` is not optional.

objectid (optional)

Input. Restricts the list to include only the child asset that you specify. You must provide both an `objectid` and an `objecttype` to request a specific child.

order (optional)

Input. The fields to sort the list by and whether the sort on those fields is ascending (`asc`) or descending (`desc`). By default, the sort is ascending. If you specify more than one field, separate the field names with a comma.

For example, if you specify `order="nrank"`, the list is sorted by rank starting at number 1. If you want the list sorted by descending rank, use `order="nrank desc"`.

Description

This tag queries the `AssetRelationTree` table for a list of the children of the asset that you specify, listing each child with a value for all of the fields from that table (`nid`, `nparentid`, `nrank`, `otype`, `oid`, and `ncode`). You must load the parent asset (using [asset:load](#)) before you can invoke this tag to query for and list its children. The list is a standard Content Server list.

Use this tag to retrieve the assets referred to by a collection asset, the image assets associated with an article asset, and so on. You can then loop through the list, or reference the data returned in the list to display the relevant information from those assets.

You can restrict the list of children by association name (`code`), object type, object ID, and rank (`order`).

If you use the `objecttype` parameter, the resulting list of children is a join of the `AssetRelationTree` and the asset table for the type specified and contains data from both tables. In such a case, you do not need to use [asset:load](#) for a child asset of that type if that asset type stores all of its asset data in the primary asset table—you can get that information from the list.

errno

The possible values of `errno` include:

Value	Description
-111	The asset has no children.
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-10006	The object ID is not valid.
-10007	The version of the object is not valid.

Example

This code retrieves the query asset that has been associated with a parent asset through the LatestNews association and writes it to a list named LatestNewQuery:

```
<asset:children name="HomePage" list="LatestNewQuery"
objecttype="Query" code="LatestNews"/>
```

This code retrieves a collection of articles, determines the members of the collection, and then displays the Description field of each article:

```
<asset:load name="NewsCollection" type="Collection"
objectid='<%=ics.GetVar("id")%>' />
<asset:get name="NewsCollection" field="name"/>
<asset:children name="NewsCollection" code="-" order="nrank"
objecttype="Article" list="articleList"/>
```

```
Collection <ics:getvar name="name"/><br/>
<%
IList aList=ics.GetList("articleList");
if (aList.hasData())
{
    int numRows=aList.numRows();
    %> <OL> <%
    for (int i=0;i<numRows;i++)
    {
        %><LI><%
        aList.moveTo(i+1);
        try {out.write(aList.getValue("description"));}
        catch (Exception e) {}
        %></LI><%
    } %> </OL> <%
}
%>
```

See Also

[asset:getsubtype](#)

asset:childtypes

Builds a list of asset types that are children of this asset.

Syntax

```
<asset.childtypes
  name="asset name"
  list="list name" />
```

Parameters

name (required)

Name of previously loaded asset.

list (required)

Name of asset type list. The list includes the asset type in the OTYPE field.

Description

The `asset:childtype` tag helps to improve performance. For example, you can loop through the asset types and use an [asset:loadall](#) tag for each asset type to load all the children.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

asset:create

Creates an asset object instance.

Syntax

```
<asset:create
  name="asset instance name"
  type="asset type" />
```

Parameters

name (required)

Name to assign the new asset instance.

type (required)

Name of the asset type.

Description

Data from the object is written to the database by `asset:save`.

errno

The possible values of `errno` include:

Value	Description
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing.
-10010	Invalid attribute value.

asset:def

Returns a list of an asset type's attributes and indicates whether each attribute is optional, required, read only, or editable.

Syntax

```
<asset:def
    type="asset type name"
    [pubid="publication ID"]
    [subtype="asset subtype"]
    list="list name" />
```

Parameters

type (required)

Name of the asset type.

pubid (optional)

Site (formerly publication) ID.

subtype (optional)

Name of the asset subtype.

list (required)

Name of the list to be created. Columns are:

- **name** - Attribute name, description.
- **editable** - Set to `true` if the attribute can be edited in the UI; set to `false` for read-only.
- **required** - Set to `true` if this is a required field in the database table; set to `false` if the field is optional.
- **description** - Name that can be displayed on the screen.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

asset:export

Gathers Content Server variables to create an XML rendition of an asset.

Syntax

```
<asset:export
  name="asset instance object"
  prefix="prefix"
  [pubid="publication ID"]
  [subtype="asset subtype"]
  output="variable name" />
```

Parameters

name (required)

Name of the asset instance object.

prefix (required)

First part of the name of all variables and/or lists used by this operation. Variables are typically named with the prefix, a delimiter, then the field name. For example, `Variables.prefix:fieldname`.

pubid (optional)

Site (formerly publication) ID.

subtype (optional)

Name of the asset subtype.

output (required)

Name of the output variable. This variable contains the resulting XML.

errno

The possible values of `errno` include:

Value	Description
-12066	No subtype supplied.

asset:gather

Gathers all variables, where the variable matches a column name for the asset, and sets data in the asset instance.

Syntax

```
<asset:gather
    name="asset instance object"
    [prefix="prefix"]
    [fieldlist="field names"]
    [exclude="true|false"] />
```

Parameters

name (required)

Name of the asset instance object.

prefix (optional)

First part of the name of all variables and/or lists used by this gather operation.

Variables are named with the prefix together with the field name. If a delimiter, such as a colon, was used in the creation of the variables that you want to gather, you must use the same delimiter in the gather operation. For example, `Variables.prefix:fieldname`.

fieldlist (optional)

Specifies a comma-separated list of field names that are gathered. Any field in the list that does not exist in the environment is cleared. If this parameter is not specified, then the gather operation searches all local fields in the environment; but if none are found, the existing value is maintained. The following keywords represent groups of attributes:

- `LocalFields` - Fields of the asset's primary table.
- `Association-named` - Named associations.
- `Association-unnamed` - Unnamed associations.
- `Publist` - Site (formerly publication) IDs

exclude (optional)

When set to `true`, this parameter specifies to scatter all fields except those specified in the fieldlist.

Description

The `asset:gather` tag performs multiple [asset:set](#) operations based on a variable naming convention.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

asset:get

Retrieves the value of a single field from a loaded asset object and stores it in a Content Server variable.

Syntax

```
<asset:get
    name="assetName"
    field="fieldName"
    [output="outputVariable"]/>
```

Parameters

name (required)

Input. The name assigned to the asset whose field value you want to get. This asset must be loaded and assigned a name before you can pass its name to this tag. (See [asset:load](#).)

field (required)

Input. The name of the field whose value you want to retrieve.

output (optional)

Input. The name of the output variable to create to store the value of the field. If you do not supply a name for this variable, CS-Direct uses one with the same name as the field.

Description

This tag obtains the value (or contents) from the specified field of a loaded asset and stores it in the specified output variable.

You typically use this tag to extract the asset information (from an asset object) that you want to display on-line. If you want to extract information from more than one field, use `asset:scatter`. (See [asset:saveall](#).)

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

Example

This code retrieves and displays the contents (value) of the description field:

```
<asset:get name="Article1" field="description"
output="headline"/>
<ics:getvar name="headline"/>
```

This code retrieves the value of the abstract field, writes it to the variable named `abstract`, and then displays the contents of the variable:

```
<asset:get name="Article1" field="abstract" />
<ics:getvar name="abstract"/>
```

See Also

[asset:load](#)
[asset:saveall](#)

asset:getassettype

Loads the asset type object of an asset instance.

Syntax

```
<asset:getassettype
  name="asset instance object"
  outname="assettype instance object" />
```

Parameters

name (required)
 Name of the asset instance object.

outname (required)
 Name to assign the asset type instance object.

errno

The possible values of errno include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

asset:getexportfile

Gets the export file name for an asset and writes it to a Content Server variable.

Syntax

```
<asset:getexportfile
  name="asset instance object"
  [parent="parent asset instance object"]
  [target="publish destination ID"]
  outname="variable name" />
```

Parameters

name (required)
 Name of the asset instance object.

parent (optional)

Name of the parent asset instance object, if any.

target (optional)

ID of the publish destination. If the ID is provided, the tag checks the AssetExportData for any target-specific override to the file name and/or path.

outname (required)

Name of the variable into which to write the file name.

Description

If this tag is called during a publish operation, it looks for the publishing session named pubsess and the suffix argument of that publishing session to determine the extension for the file name. If it is not called during a publish operation, it uses .html.

errno

The possible values of errno include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

asset:getlegalsubtypes

Generates a list of asset subtypes that are valid for this asset type.

Syntax

```
<asset:getlegalsubtypes
  list="variable name"
  type="asset type"
  [pubid="publication ID"] />
```

Parameters

list (required)

Name of the output variable.

type (required)

Name of the asset type.

pubid (optional)

Site (formerly publication) ID. The list of asset subtypes can be restricted by specifying the site ID.

Description

The generated list has a SUBTYPE column.

errno

The possible values of `errno` include:

Value	Description
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing.

asset:getpubdeps

Gets the publish dependencies for an asset.

Syntax

```
<asset:getpubdeps
  name="asset instance object"
  context="pubcontext object"
  list="variable name"
  [depth="-1 or 1"] />
```

Parameters

name (required)

Name of the asset instance object.

context (required)

Name of PubContext object.

list (required)

Name of the list to create containing dependent assets.

depth (optional)

Levels to search for dependency. A value of -1 indicates to search for the all dependencies to all levels. The default value is 1.

Description

The default implementation of this tag includes any child asset in the AssetRelationTree table. The generated list contains the following columns: OTYPE, OID, OVERSION.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

asset:getsitenode

Queries the SitePlanTree table and returns the node ID of the specified page asset.

Syntax

```
<asset:getsitenode  
    name="pageAssetName"  
    output="nodeIDVariableName"/>
```

Parameters

name (required)

Input. Name assigned to the page asset whose node ID you want to retrieve. This page asset must be loaded and assigned a name before you can pass its name to this tag. (See [asset:load](#).) If this asset is not a page, CS-Direct cannot return a node ID.

output (required)

Input. Variable name to assign to the page asset's node ID.

Description

The relationships set up between page assets on the site tree in the CS-Direct main window are stored in the SitePlanTree table. This tag uses the object ID of a page asset to retrieve its node ID from that table.

You can then use the node ID to acquire information about the site's hierarchy to use for display: for example, to create a navigation bar with links to section pages or a link back to the parent page.

For more information about the SitePlanTree table, see [Site Plan Tags](#).

errno

The possible values of errno include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory)

Example

This code loads a page asset and then determines the site node of that page:

```
<asset:load name="NewsPage" type="Page"  
objectid='<%=ics.GetVar("id")%>' />  
<asset:getsitenode name="NewsPage" output="PageNodeId"/>
```

See Also

[siteplan:children](#)

[siteplan:get](#)

[siteplan:getpagename](#)

[siteplan:listpages](#)[siteplan:load](#)[Release Introduced](#)

asset:getsubtype

Gets the subtype of the asset.

Syntax

```
<asset:getsubtype
    name="loaded asset"
    [output="variable name"] />
```

Parameters

name (required)

Name of the loaded asset.

output (optional)

Name of the variable in which to set the results. If the name is not supplied, subtype is used.

Description

If no subtype has been specified for this asset, Standard (the generic subtype) is returned by default.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

asset:gettemplaterootelement

Calculates the name of an asset's template and gets that template asset's root element, without loading the template.

Syntax

```
<asset:gettemplaterootelement
  name="asset instance object"
  [type="asset type"]
  [template="template name"]
  output="variable name" />
```

Parameters

name (required unless type and template arguments provided)

Name of the asset instance object.

type (optional unless template argument used)

Asset type of the asset.

template (optional)

Name of the template. If none is provided, the template name is retrieved from the loaded asset instance.

output (required)

Variable in which to store the root element of the calculated template.

errno

The possible values of errno include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-12004	An asset was not found for a specified ID.
-12006	There is no such asset type.

asset:gettemplatefortarget

Returns a template name in the designated variable.

Syntax

```
<asset:gettemplatefortarget
    name="asset object name"
    target="publish id"
    output="variable name" />
```

Parameters

name (required)

Name of the asset object.

target (required)

ID of the Publish target.

output (required)

Name of the output variable to create.

Description

This tag checks to see if there is a default template in the `AssetDefaultTemplate` table (configured by using the `Admin/Publishing/[Publish Destination]/SetDefaultTemplate` function) for the asset type /subtype combination. If there is no default template, the tag gets the template from the asset. If the asset has no assigned template, it returns the system default template (`OpenMarket/DefaultTemplate`). This tag displays the template which is used when approving the asset for export.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

asset:import

Parses XML and sets Content Server variables which can then be gathered to set values in the asset.

Note

If you use `asset:import` to gather values into a flex asset, you must provide an asset subtype using the `subtype` parameter.

Syntax

```
<asset:import
  name="asset instance object"
  prefix="prefix"
  xml="XML code"
  [pubid="publication id"]
  [subtype="asset subtype"] />
```

Parameters

name (required)

Name of the asset instance object.

prefix (required)

Prefix to use when setting ICS variable names. The variable names are `prefix:attribute`.

xml (required)

XML code that is parsed.

pubid (optional)

Publication ID.

subtype (optional, unless using this tag for a flex asset)

Asset subtype.

errno

The possible values of `errno` include:

Value	Description
-408	Bad XML syntax.
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-12004	An asset was not found for a specified ID.
-12037	Required field missing.
-12038	Incorrect data type.

Value	Description
-12043	Data out of range.
-12060	A field sent to the save asset function (after XML parsing) was too long.
-12066	No subtype supplied.

asset:inspect

Creates an object with the asset's attributes and the specification for each attribute.

Syntax

```
<asset:inspect
  list="list name"
  type="asset type"
  [pubid="publication ID"]
  [subtype="asset subtype"] />
```

Parameters

list (required)
Name of the list.

type (required)
Asset type.

pubid (optional)
Publication ID.

subtype (optional)
Asset subtype.

Description

The returned list object has columns NAME and SPEC.

errno

The possible values of errno include:

Value	Description
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-12004	An asset was not found for a specified ID.

asset:list

Queries the database and retrieves a list of assets that meet the specified criteria. It creates a list of assets of one type.

Syntax

```
<asset:list
  type="assetType"
  list="nameOfList"
  [order="order"]>
  <asset:argument name="fieldName" value="fieldValue"/>
</asset:list>
```

Parameters

type (required)

Input. Asset type of the assets to list.

list (required)

Input. Name to assign to the list of assets. If a list with this name already exists, it is replaced with this new list.

asset:argument (optional)

Input. Field name/value pairs to pass to the tag to restrict the list. For example:

```
<asset:argument name="Category" value="Sports"/>
```

order (optional)

Input. Fields to sort the list by and whether the sort on those fields is ascending (asc) or descending (desc). By default, the sort is ascending. If you specify more than one field, separate the field names with a comma.

For example, if you specify `order="nrank"`, the list is sorted by rank starting at number 1. If you want the list sorted by descending rank, use `order="nrank, desc"`.

pubid (optional)

Specifies the ID of the publication to which the query is restricted. When this optional parameter is used, the query performs a database join operation against the AssetPublication table.

Description

This tag queries the database table for assets of this type and then builds a list of assets, selecting those that meet certain criteria. The criteria you can use include asset type and field/value pairs that define assets of that type. You pass in field/value pairs with embedded `asset:argument` statements.

Use this tag to retrieve multiple assets of the same type that do not necessarily have associations or relationships with each other.

errno

The possible values of `errno` include:

Value	Description
-10001	The implementing class is invalid.

Value	Description
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing.

Example

This code creates a list named "ArticleList" composed of all the article assets that have the keyword "President":

```
<asset:list type="Article" list="ArticleList">
  <asset:argument name="keyword" value="President"/>
</asset:list>
```

See Also

[asset:addchild](#)

asset:load

Queries the database for the specified asset and then loads an instance of the asset into memory as an object.

Note

This tag can not be used to load a flex asset. However it can be used to load attributes of type assets within flex assets. To load flex assets, see [Assetset Tags](#).

Syntax

```
<asset:load
  name="assetName"
  type="assetType"
  objectid="object.id"
  [field="fieldName"]
  [value="fieldValue"]
  [deptype="exact|exists|greater"]/>
```

Parameters

name (required)

Input. Name to identify the asset after it is retrieved and stored as an object in memory. All references to the asset by other asset tags ([asset:children](#) and [asset:get](#), for example) during this session use this name to identify the asset.

type (required)

Input. Asset type of the asset that you want to retrieve from the database. Provide both type and objectid.

objectid (required)

Input. Object ID of the asset. It is not required if you use a `field/value` pair (defined below).

field (optional)

Input. Used in a pair of values with `value` to identify the asset by supplying a field name whose corresponding value uniquely identifies the asset. Note that if the `field/value` pair that you supply identifies more than one asset, CS-Direct loads the first one that it finds.

value (optional)

Input. Used in a pair of values with `field` to identify the asset by supplying a value for a field that uniquely identifies the asset.

deptype (optional)

When the using the `asset:load` tag in a template, and the assets using this template have been approved for export, this attribute specifies whether approval dependency is `exact`, `exists` or `greater`.

Legal values are:

- **exact**(default) – Exact specifies a version dependency. The version of the dependent asset must be equal to the version of the asset when it was approved.
- **exists** – Exists specifies that any version of the asset satisfies the dependency condition.
- **greater** – Greater specifies the asset must be at a or above a certain version.

Description

This tag executes a database query to retrieve an instance of the specified asset and then saves the instance into memory (session information), assigning it the name provided by the `name` parameter. That object is then available until the session is flushed (the root element exits) or the name is overwritten. Object names are global in scope.

You can identify the asset that you want to load with its asset type and object ID, or the asset type combined with a `field/value` pair that uniquely identifies the asset.

Typically, each JSP template element starts with an `asset:load` tag so that subsequent code can extract and display the loaded data on pages for your on-line site.

Note that `asset:load` loads information from the asset's storage table only. It does not load information from the `AssetPublication` table or the `AssetRelationTree` table, for example. If you need information from the `AssetRelationTree` table, use the [asset:children](#) tag. If you need information from other auxiliary tables, you can issue queries using the object ID of the loaded asset. The only exception to this rule is for page assets. When you load a page asset, you also have access to information about that asset's node in the `SitePlanTree` table.

errno

The possible values of `errno` include:

Value	Description
-108	Tree selector error.
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.

Value	Description
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-10006	The object ID is not valid.
-10007	The version of the object is not valid.
-10008	The node ID of the object is not valid.
-10009	More than one object met the specified criteria.

Example

This code loads an article asset, identifying it by type and object ID:

```
<asset:load name="FirstArticle" type="Article"
objectid="<%=ics.GetVar("id")%>" />
```

This code loads a page asset, identifying it by type and a field/value pair:

```
<asset:load name="NewsSectionFront" type="Page" field="category"
value="News" />
```

See Also

[asset:getsubtype](#)

asset:loadall

Calls [asset:load](#) for all IDs passed in the input list and creates objects with names made up of the prefix name plus a counter.

Syntax

```
<asset.loadall
    type="asset type"
    [list="list object name"]
    [ids="asset IDs"]
    [idfield="field name"]
    [prefix="prefix"]
    [initcount="integer"]
    [deptype="exact, exists, or greater"] />
```

Parameters

type (required)

Type of the asset to load.

list (optional)

You must use either the LIST or IDS parameter. Name of an object containing the object IDs of assets to load.

ids (optional)

You must use either the LIST or IDS parameter. A variable containing a semicolon-separated list of asset IDs to load.

idfield (optional)

Name of the field in the list object which specifies the object ID of the asset. The default value is ID.

prefix (required)

Base name of the loaded asset. The object name consists of this value plus a counter incremented by 1 for each asset loaded. A variable named `Prefix+Total` holds the number of assets loaded.

initcount (optional)

Initial value of the counter to be appended to the PREFIX value to determine the object name of the loaded asset. The default value is 1.

deptype (optional)

When the [asset:loadall](#) tag is used in a template and assets using this template are approved for export, this parameter specifies whether approval dependency is exact, exists, or greater. The default value is exists.

Description

If a list is passed as input and the list contains at least as many columns as the primary storage table for the asset type, the asset is loaded from the list without further querying of the database. If a list is passed with fewer columns than the number of columns in the asset type's primary storage table, then the ID is extracted and used to query the asset type's storage table to load the table. If a string is passed with a list of IDs, those are used to load the asset type table. A variable, with a name consisting of the `prefix+Total` indicates how many assets were loaded.

errno

A list of error messages is written to `Variables.errdetails $n$` , where n is an iterator beginning with 1.

The possible values of `errno` include:

Value	Description
-1000	Revision tracking error.
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing.

asset:loadrevision

Loads a previous revision of a revision-tracked asset using the [asset:load](#) tag.

Syntax

```
<asset:loadrevision
  type="asset type"
  name="object name"
  objectid="asset ID"
  revision="integer" />
```

Parameters

type (required)
Type of the asset to load.

name (required)
Object name to assign the loaded asset.

objectid (required)
Asset ID.

revision (required)
Integer indicating which asset revision to load.

errno

The possible values of `errno` include:

Value	Description
-1000	Revision tracking error.
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing.
-12004	An asset was not found for a specified ID.

More specific error numbers are set in `Variables.errdetail`. See [errdetail Error Values](#) for more information.

asset:referencedby

Builds a list of assets that reference this asset.

Syntax

```
<asset:referencedby
  name="asset name"
  list="list name" />
```

Parameters

name (required)

Name of previously loaded asset.

list (required)

Name of the list of assets that reference the asset. The list contains a column called OTYPE, which contains the asset type and the OID for the ID of the referencing asset.

Description

The list contains the fields of the AssetRelationTree.

errno

The possible values of errno include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

asset:removechildren

Designates which entries to remove from the AssetRelationTree.

Syntax

```
<asset:removechildren
  name="asset instance object"
  type="asset type"
  childid="child asset ID"
  code="association name" />
```

Parameters

name (required)

Name of the asset instance object.

type (optional)

Asset type of children to remove. If this parameter is omitted, children of all types are removed.

`childid` (optional)

ID of a child asset to remove.

`code` (optional)

Association name of child asset to remove.

Description

The `AssetRelationTree` is updated during an `asset:save` operation.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

asset:save

Saves an asset.

Syntax

```
<asset:save
  name="asset instance object"
  [flush="true|false"] />
```

Parameters

`name` (required)

Name of the asset instance object.

`flush` (optional)

If set to `true`, flushes the asset from the object pool.

Description

This tag saves an asset to its storage and, if it is a new asset, to the `AssetPublication` table. Associations defined by using the `AddChild` `RemoveChildren` or `ChildGather` methods are also saved there.

The following fields in the asset are updated by default:

- `status` – If no value is supplied, the status is `m_sStatusCreated` ('PL').
- `createdby` – If no value is supplied and this is a new asset set to the session variable `username`.
- `createddate` – If no value is supplied and this is a new asset, set to the current date.
- `updatedby` – If no value is supplied, set to session variable `username`.
- `createddate` – If no value is supplied, set to the current date.

Saving to the AssetPublication table uses the following logic and variables:

- `primarypubid` (variable) – An entry is made based on the variable `primarypubid`.
- `pubid` (session variable) – If `primarypubid` does not exist, the session variable `pubid` defines the publication in which to put the asset.

errno

The possible values of `errno` include:

Value	Description
-105	Database error.
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-12018	Missing publication.
-12034	Publish in progress.
-12065	Error updating AssetPublication.

asset:saveall

Calls [asset:save](#) for all assets named with the prefix name plus a counter.

Syntax

```
<asset.saveall
  type="asset type"
  prefix="loaded asset name"
  [initcount="integer"]
  [flush="true|false"] />
```

Parameters

`type` (required)

Type of the asset to save.

`prefix` (required)

Base name of the loaded asset. The object name consists of this value plus a counter incremented by 1.

`initcount` (optional)

Initial value of the counter to be appended to the `PREFIX` value to determine the object name of the loaded asset. The default value is 1.

`flush` (optional)

If set to `true`, the object is flushed from the object pool. The default value is `false`.

errno

When an error is encountered, other assets are saved. A list of error messages is written to `Variables.errdetailsn`, where *n* is an iterator beginning with 1.

The possible values of `errno` include:

Value	Description
-105	Database error.
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-12018	Missing publication.
-12034	Publish in progress.
-12065	Error updating AssetPublication.

See Also

[asset:loadall](#)

asset:scatter

Retrieves and makes variables of (“scatters”) the values from all the fields of a loaded asset object. (To obtain just one field value, use [asset:get](#).)

Syntax

```
<asset:scatter
    name="assetName"
    [prefix="variablePrefix"]/>
```

Parameters

name (required)

Input. Name assigned to the asset whose fields you want to make variables of (scatter). This asset must be loaded and assigned a name before you can pass its name to this tag. (See [asset:load](#).)

prefix (optional)

Input. Optional but rarely omitted. Specifies a string to be included in the name of the variables that store the asset’s information. When you specify a prefix, you then create your variables by using a colon to separate the prefix from the field name. For example, if you specify `prefix="article"`, your variables are `article:fieldname`.

Description

This tag retrieves values from the fields of a previously loaded asset and then writes them into memory (session information) as variables. The variables are constructed with the

prefix that you supply with the `prefix` parameter. After you use this tag, you can display the information contained in the asset's fields.

Use this tag rather than multiple [asset:get](#) commands when you want to retrieve more than a couple of fields.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

Example

This code retrieves values from a loaded asset named `Article1`, writes variables for those values into memory, and then displays the data in the asset's `description` and `abstract` fields:

```
<asset:scatter name="Article1" prefix="article"/>
  <%=ics.GetVar("article:description")%>
  <%=ics.GetVar("article:abstract")%>
```

See Also

[asset:load](#)
[asset:childtypes](#)

asset:search

Locates a list of asset primary table rows based on the asset type and a set of search criteria.

Syntax

```
<asset:search
  type="asset type"
  [subtype="asset subtype"]
  prefix="prefix"
  [fieldlist="field name(s)"]
  localfields="fields"
  publist="publication ID"
  [exclude="true|false"]
  [limit="maximum assets"]
  [order="order-by string"]
  list="list variable name" />
```

Parameters

type (required)

Asset type of assets for which to search.

subtype (optional)

Asset subtype.

prefix (required)

Prefix for variable names used to specify search criteria .

fieldlist (optional)

A comma-separated list of field names that may have search criteria specified. If this argument is not specified, then all local fields are considered.

localfields

Fields of the asset's primary table.

publist

Site ID (Publication ID).

exclude (optional)

When set to true, it specifies to scatter all fields except those specified by the fieldlist parameter.

limit (optional)

Maximum number of assets to return.

order (optional)

An Order-By string to add to the end of the query. You can only use fields in the asset type's primary table for the order-by string.

list (required)

Name of resulting list.

Description

The search criteria are gathered from Content Server variables. For attributes of simple types, the variable search criteria is gathered from the following:

- *prefix:attribute_op* – Search operation - Valid values are =, !=, <, >, <=, >=, like, or not like.
- *prefix:attributeTotal* – Number of criteria for this attribute to string together with ORs.
- *prefix:attributen* – Where *n* is a number between 0 and *prefix:attributeTotal* -1, indicating the search value.

If there is no *prefix:attributeTotal* variable:

- *prefix:attribute* – Search value

errno

The possible values of errno include:

Value	Description
-105	Database error.
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10001	The implementing class is invalid.
-12006	There is no such asset type.

asset:set

Sets a field in an asset.

Syntax

```
<asset:set
  name="asset instance object"
  field="field name"
  value="value" />
```

Parameters

- name** (required)
Name of the asset instance object.
- field** (required)
Field to set.
- value** (required)
Value to which to set the field.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-12065	Error updating AssetPublication.

asset:share

Updates the AssetPublication table to have an entry for each of the requested sites.

Syntax

```
<asset:share
  name="asset instance object"
  publist="publication IDs" />
```

Parameters

name (required)

Name of the asset instance object.

publist (required)

Semicolon-separated list of site (formerly publication) IDs from which the asset is accessible.

asset:siteparent

Queries the SitePlanTree table and then loads the parent page of the specified page asset into memory as an object.

Syntax

```
<asset:siteparent
  name="pageName"
  outname="parentPageName" />
```

Description

This tag queries the SitePlanTree table and then loads as an object the parent page of the specified page asset. It acts like [asset:load](#).

Use this tag to display information about the hierarchical position of the current page or to create a link to the current page's parent page. This tag determines the parent page and then loads the parent page in one database query.

Parameters

name (required)

Input. Name of the page asset whose parent page you want to load. This page asset must be loaded and assigned a name before you can pass its name to this tag. (See [asset:load](#).)

outname (required)

Input. Name to assign to the parent page object.

errno

The possible values of errno include:

Value	Description
-112	No parent exists.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-10007	The version of the object is not valid.
-10008	The node ID of the object is not valid.
-10009	More than one object met the specified criteria.
-12007	The specified value is not valid.

Example

This code loads a page asset, loads its parent page, extracts the name of the parent page asset, and then displays the name:

```
<asset:load name="myPage" type="Page"
  objectid=<ics:getvar name="grandparent:name"/>
<asset:siteparent name="myPage" outname="grandparent"/>
<asset:get name="grandparent" field="name"
output="grandparent:name"/>
Up one level to <ics:getvar name="grandparent:name"/>.
```

See Also

[Site Plan Tags](#).

[asset:getassettype](#)

asset:sites

Creates a list object of sites (formerly known as publications) to which the asset is shared. The list contains the columns of the Publication table.

Syntax

```
<asset:sites
    name="asset name"
    list="list name"
    pubid="publication ID" />
```

Parameters

name (required)
Name of previously loaded asset.

list (required)
Name of list that is created.

pubid (optional)
Site (formerly publication) ID.

errno

The possible values of errno include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

asset:stream

Streams the value of a field from an object.

Syntax

```
<asset:stream
    name="asset instance object"
    field="field name"
    [resolve="true|false"] />
```

Parameters

name (required)
Name of the asset instance object.

field (required)
Field to get.

resolve (optional)
If true, resolves Content Server variable names. The default value is true.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

asset:void

Sets the status code of the asset to V0 and removes any child relations from the AssetRelationTree.

Syntax

```
<asset:void
    name="asset instance object" />
```

Parameters

`name` (required)
Name of the asset instance object.

Description

If there were children in the AssetRelationTree, the root reference remains.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-12018	Missing publication.
-12034	Publish in progress.

InSite Editor Tag

The following section describes the InSite editor tag, [insite:edit](#).

insite:edit

Edits assets from the user interface.

Syntax

```
<insite:edit
  assetid="ID of asset"
  assetfield="Attribute_name of asset's field"
  assetfieldvalue="field value to be edited"
  assettype="type of asset"
  [ewebeditpro="true|false"]
  [width="pixels or percentage"]
  [height="pixels"]
  [variable="variable name"]
  [ssvariable="session variable name"]
  [property="property name"]
  [list="list name"]
  [column="column name"]/>
```

Parameters

assetid (required)

ID of the asset to be edited.

assetfield (required)

Name of the field of the asset to be edited.

assettype (required)

Type of asset to be edited.

ewebeditpro (optional)

Legal values are true or false. The default value is true.

width (optional)

Width of the asset in percentage or pixels. The default value is 100%.

height (optional)

Height of the asset in pixels only. The default value is 200 pixels. Percentage values are unsupported.

variable (optional)

You must use at least one of the following four parameters: `variable`, `ssvariable`, `property`, or `list` and `column` (`list` and `column` must be used together). Name of the Content Server variable that you want to stream to a page. For example, for a variable called `foo`, set the parameter equal to `foo` rather than to `Variables.foo`.

ssvariable (optional)

You must use at least one of the following four parameters: `variable`, `ssvariable`, `property`, or `list` and `column` (`list` and `column` must be used together). Name of the Content Server session variable that you want to stream to a page.

property (optional)

You must use at least one of the following four parameters: `variable`, `ssvariable`, `property`, or `list` and `column` (`list` and `column` must be used together). A Content Server property from the `futuretense.ini` property file.

list (optional)

You must use at least one of the following four parameters: `variable`, `ssvariable`, `property`, or `list` and `column` (`list` and `column` must be used together). Name of the list.

column (optional)

You must use at least one of the following four parameters: `variable`, `ssvariable`, `property`, or `list` and `column` (`list` and `column` must be used together). Name of the field in the list.

Description

This tag allows you to edit assets in the Content Server system. When editing standard assets (the `assetfileid` is the column name, for example), the `Headline` field is the description column for Article assets. When editing flex assets, the attribute name should be preceded by the prefix `Attribute_`; for example, the body attribute would be `Attribute_body`. For more information about editing assets, see the *dCS Developer's Guide*.

Example

```
<insite:edit
  assetid="Variables.cid"
  assetfield="Attribute_headline"
  assetfieldvalue="assetlist.SORT_headline"
  assettype="AArticles"
  ewebeditpro="true"
  width="100%"
  height="200"/>
```


Property Tags

The following section describes the property tag, [property:get](#).

property:get

Gets a property value from the currently loaded property files.

Syntax

```
<property:get  
  name="propertyname"  
  [file="property file name"]  
  [output="variable name"] />
```

Parameters

name (required)

Name of the property to retrieve.

file (optional)

A semicolon-separated list of property file names.

output (optional)

Specifies an optional output variable. If this parameter is not specified, the output is streamed to the browser.

Description

Retrieves a property value from the currently loaded or optionally specified property file. The property may then be used in conjunction with other XML tags.

errno

None.

Render Tags

CS-Direct renders content in three ways:

- By rendering pages and pagelets into static HTML files through the Export to Disk publishing method.
- By rendering pages and pagelets dynamically when a visitor requests them on a dynamic public site.
- By rendering assets in response to the **Preview** function on a CS-Direct management or development system.

CS-Direct determines how to render content by using an argument named `rendermode`, which has three valid settings:

- `export` – CS-Direct uses the Export to Disk publishing method to create static HTML files
- `live` – CS-Direct invokes Content Server to serve content dynamically
- `preview` – CS-Direct invokes Content Server to render a selected asset in a browser pointed at a CS-Direct management or development system

`rendermode` is set through the `resargs` of page entries in the SiteCatalog table. When CS-Direct creates page entries for templates, it sets `rendermode` as an argument in the `resargs2` column for that entry and sets it to `live`. If you export or preview an asset that uses that template, CS-Direct sets `rendermode` to `export` or `preview`, as appropriate.

The render tags process your CS-Direct page and element requests and create URLs and arguments for URLs. Some of the tags ([render:callelement](#) and [render:satellitepage](#), for example) function similarly to tags that are available through Content Server or CS-Satellite. However, the render tags preserve the CS-Direct rendering environment and the variables needed in that environment without requiring additional code.

Note

Each JSP file that includes CS-Direct JSP render tags must have the following directive at the top:

```
<%@ taglib prefix="render" uri="futuretense_cs/render.tld" %>
```

If you use either [render:satelliteblob](#) or [render:satellitepage](#) tags, you open and close the file with a pair of [satellite:tag](#) tags. A JSP file that includes CS-Satellite JSP tags must have the following directive at its top:

```
<%@ taglib prefix="satellite" uri="futuretense_cs/satellite.tld" %>
```

The render tags (with the exception of [render:encode](#)) are for use on dynamic sites only.

render:callelement

Calls an element from the ElementCatalog table.

Syntax

```
<render:callelement
    elementname="nameOfElement">
    [<render:argument name="variable1" value="value1"/>]
</render:callelement>
```

Parameters

elementname (required)

Input. Name of the element to invoke.

render:argument (optional)

Input. Name/value pairs to pass to the called element. For example, CS-Direct sets several variables through the `resargs2` column in a page entry. To pass a value for one of these variables to a `render:callelement` tag, you use an argument. For example:

```
<render:argument name="c" value="Article"/>
<render:argument name="cid" value='<%=ics.GetVar("id")%>' />
```

Description

This tag is the CS-Direct equivalent of the Content Server XML [CALLELEMENT](#) tag with the following differences:

- Its scope is always local.
- `rendermode` is passed to the element if `rendermode` is set to `export` or `preview`.

Use this tag to process the content of an element that you wrote for CS-Direct and you want the scope of that element to be local. The element must exist in the ElementCatalog.

As with the XML [CALLELEMENT](#) tag, variables that exist prior to `render:callelement` are available to tags within the called element. Variables passed to an element are subject to the following order of precedence:

1. Variables set in arguments.
2. Variables set in scripts.
3. Variables set in the `resdetails1` and `resdetails2` columns in the ElementCatalog.

However, because the scope of an element called with this tag is local, variables set within the called element are not available within the parent (calling) element.

errno

The possible values of `errno` include:

Value	Description
-10	The element does not exist.
-12	The element cannot be evaluated.

Example

The following code sample calls an element named `TopPageBreadcrumb`. It displays the path you took to get to the currently displayed page. For example, if you visited the News page directly from a link on the Home page, the `TopPageBreadcrumb` element displays "Home > News".

```
<render:callelement
    elementname="BurlingtonFinancial/Site/TopPageBreadcrumb"/>
    <render:argument name="c" value="Article"/>
</render:callelement>
```

See Also

[render:contentserver](#)

render:contentserver

Calls and renders a page or a pagelet from the SiteCatalog table.

Syntax

```
<render:contentserver
    pagename="nameOfPageEntry"/>
    [<render:argument name="variable1" value="value1"/>]
</render:contentserver>
```

Parameters

pagename (required)

Input. Name of the page entry from the SiteCatalog to render.

render:argument (optional)

Input. Name/value pairs to pass to the called element. For example, CS-Direct sets several variables through the `resargs2` column in a page entry. To pass a value for one of these variables to a `render:contentserver` tag, you use an argument. For example:

```
<render:argument name="c" value="Article"/>
<render:argument name="cid" value='<%=ics.GetVar("id")%>' />
```

Description

This tag is the CS-Direct equivalent of the Content Server XML [CONTENTSERVER \(Deprecated\)](#) tag with the following differences:

- Its scope is as follows: variables in the calling element are not available in the called element (unless explicitly passed in); changes made to variable values in the called element are not reflected in the calling element; and new variables set in the called element are not available to the calling element.
- `rendermode` is passed to the page entry.

Use this tag to call the ContentServer module to serve a CS-Direct page or pagelet. The page or pagelet must have a page entry in the SiteCatalog. If the specified pagelet is already cached, this tag loads it. If the specified pagelet is not already cached, this tag caches it (if caching is enabled for it) and loads it.

Variables passed to the ContentServer servlet are subject to the following order of precedence:

1. Variables set in arguments.
2. Variables set in scripts.
3. Variables set in the resargs1 and resargs2 columns of the SiteCatalog.
4. Variables set in the resdetails1 and resdetails2 columns of the ElementCatalog.

If an element is coded with a [render:satellitepage](#) tag but CS-Satellite is not present, [render:satellitepage](#) passes the call to [render:contentserver](#) which then serves the page or pagelet.

errno

The value of errno is unchanged.

Example

This code calls for the page entry of a template named PlainList, passing the ID of the asset that it should display and the ID of the asset's parent page:

```
<render:contentserver pagename="BurlingtonFinancial/Collection/
PlainList">
<render:argument name="cid
value='<%=SectionHighlights.getValue("oid")%>'/>
<render:argument name="p" value='<%=ics.GetVar("asset:id")%>'/>
</render:contentserver>
```

See Also

[render:satellitepage](#)

render:encode

Produces a complete URL (from the servlet name onward) from the arguments, previously packed arguments, and the servlet name (usually ContentServer or BlobServer) provided.

Syntax

```
<render:encode
  str="servletName"
  ostr="variableName"
  [addsession="true/false"]>
  [packedargs="stringFrompackargstag"]>
  <[argument name="variable1 value="value1"]/>
</render:encode>
```

Parameters

str (required)

Input. Name of the servlet to use when rendering the URL, either ContentServer or BlobServer. You can include all the information for the URL to the left of the page name (which is supplied dynamically). For example:

`http://localhost/servlet/ContentServer?pagename=index/home`

outstr (required)

Input. Name of the variable that stores the URL generated by this tag.

addsession (optional)

Input. Specifies whether to include session IDs in the URL when a browser is set to reject cookies. `true` means to encode session IDs in the URL. If this parameter is not specified, it is set to `true` by default.

render:argument (optional)

Input. Name/value pairs to pass to the tag. The names and values of these arguments are URL-encoded and incorporated into the final URL.

An argument passed in with this parameter takes precedence over an argument passed in with the `packedargs` parameter. If the same argument is passed in with both, the `packedargs` version is overridden by the argument version.

packedargs (optional)

Input. Previously created URL-encoded packed arguments, in standard form; that is, the output from a previous [render:packargs](#) tag.

Description

This tag creates a URL based on the arguments and packed arguments that are passed to it. All characters in the URL are URL-encoded.

Call the standard CS-Direct `GetPageURL` element to obtain URLs for hyperlinks on a page. That element is coded to preserve and resolve the various CS-Direct variables that are necessary for rendering pages in the CS-Direct environment. However, in special cases, you might need to use this tag to obtain a URL instead.

For example, if your site is static but you want to present a dynamic URL inside a static HTML file, use this tag. A URL created with this tag remains dynamic, even when passed through the Export to Disk publishing method. That is, Export to Disk does not translate a dynamic URL created by this tag into a static HTML file.

On a dynamic site, you can use this tag if your page or pagelet exists completely outside of the CS-Direct environment and you do not need to preserve any of the CS-Direct variables (`c`, `cid`, `ct`, `p`) other than `rendermode` when creating a URL for it.

This tag completes the appropriate URL encoding required for all the arguments passed to it—with the exception of arguments passed in as `packedargs` which are assumed to be already URL-encoded. (The `render:packargs` tag URL-encodes arguments passed to it, which are the arguments you pass in to `render:encode` as `packedargs`.)

This tag also includes session IDs in the URL, according to the conventions of the local application server, if a browser is set to reject cookies.

errno

The possible values of `errno` include:

Value	Description
-1004	An argument is missing.

Example

This code uses `render:packargs` to URL-encode the name of the site as a string and then uses `render:encode` to create a URL that includes that encoded string:

```
<render:packargs outstr="myPackedArgs">
  <render:argument name="sitename" value="MySite"/>
<render:encode outstr="myURL" str="servlet/ContentServer?"
packedargs='<%=ics.GetVar("myPackedArgs")%>' />
```

See Also

[render:packargs](#)

render:export

Begins the export process. It starts with the specified page and recursively exports all links (both pages and blobs).

Syntax

```
<render:export
  pagename="page_name"
  rootfolder="root_directory"
  virtualfolder="url_prefix"
  cid="asset_id"
  [c="assettype"]
  [p="parent_page_id"]
  [packedargs="stringfrompackargstag"]
  [args_var1="value"] />
```

Parameters

pagename (required)

Name of the page to export.

rootfolder (required)

Export root directory. Its value equals the base directory (`cs.pgexportfolder`) plus optional subdirectory (DIR from publish destination arguments).

virtualfolder (required)

URL prefix to add to the beginning of all assets' URLs contained in exported files.

cid (required)

Asset ID.

c (optional)

Asset type.

p (optional)

Asset ID of the parent Page asset.

packedargs (optional)

Contains already URL-encoded packed arguments in standard form, with ampersands (&) separating the arguments, and equals signs (=) separating the argument name from value.

args_var1

A family of generic arguments. The argument name is `xxx` and its value is `yyy`. The names and values of these arguments are URL encoded and put into the final URL.

errno

The possible values of `errno` include:

Value	Description
-1004	An argument is missing.

render:filter

Filters single assets or lists of assets which have not been approved for publishing from being exported into the generated file.

Syntax

```
<render:filter
  list="list name"
  listvarname="output list name"
  listidcol="assetID column"
  [listtypecol="assettype column"]
  [type="asset type"]
  [id="asset id"]
  [varname="output variable"/>
```

Parameters

list (required)

Name of the list to filter.

listvarname (required)

Name of the list that contains the filtered assets.

listidcol (required)

Name of the asset ID column in the list to be filtered.

type (optional)

Asset type to filter.

id (optional)

Asset ID to filter.

varname (optional)

Name of the output variable that includes a value of `true` or `false`. A `true` value indicates it is safe to include the asset.

Description

This tag filters a single asset or list of assets based on whether the asset is approved for the destination being exported. Use this tag within templates that generate a list of assets, including unapproved assets. Templates used during an export publish operation which employ query assets, [EXECSQL](#) commands, [asset:list](#), and [ics:selectto](#) may be candidates for the [render:filter](#) tag.

errno

The possible values of errno include:

Value	Description
-10004	An argument is missing.

Example

```
<render:filter list="ArticlesFromTheQuery"
               listvarname="ArticlesFromTheQuery"
               listidcol="id"/>
<if COND="ArticlesFromTheQuery.#numRows!=0">
<then>
  <LOOP LIST="ArticlesFromTheQuery">

    <CALLELEMENT NAME="OpenMarket/Xcelerate/Actions/Publish/
GetPageURL">
      <ARGUMENT NAME="pagename" VALUE="BurlingtonFinancial/
Article/Variables.ct"/>
      <ARGUMENT NAME="cid" VALUE="ArticlesFromTheQuery.id"/>
      <ARGUMENT NAME="ctype" VALUE="Article"/>
    </CALLELEMENT>
    <A class="wirelink" HREF="Variables.referURL"
REPLACEALL="Variables.referURL"><csvar
NAME="ArticlesFromTheQuery.subheadline"/></A><P/>
  </LOOP>
</then>
</if>
```

Release Introduced

This tag was introduced in dCS release 4.0.

render:bloburl

Creates a BlobServer URL without embedding it in an HTML tag (see also [render:satelliteblob](#)).

Syntax

```
<render:bloburl
  outstr="VariableName"
  blobtable="blobTable"
  blobkey="primaryKeyName"
  blobwhere="primaryKeyValue"
  blobcol="column name"
  [blobheader="MIMEtype"]
  [blobnocache="false"]
  [addsession="true"]
  [dynamic="true"]
  [blobheadernamen="blob header name"]
  [blobheadervalue="blob header value"]
  [csblobid="session variable value" />
```

Parameters

outstr (required)

Input. Name of the variable that stores the URL generated by this tag.

blobtable (required)

Input. Name of the CS-Direct table that stores assets of this type. For example, `blobtable="ImageFile"`.

blobkey (required)

Input. Name of the column used as the primary key. Typically this is `id`.

blobwhere (required)

Input. Object ID of the asset.

blobcol (required)

Input. Name of the column that contains the binary data. For example, for the `imagefile` asset type, `blobcol="urlpicture"`.

blobheader (optional)

Input. Mimetype for returned data, in the form *description/extension*. For example, the mimetype for a GIF image is `image/gif`.

blobnocache (optional)

Input. Specifies whether to disable blob caching. A value of `true` prevents the blob from being cached.

addsession (optional)

Input. Specifies whether to include session IDs in the URL when a browser is set to reject cookies. `true` means to encode session IDs in the URL. If this parameter is not specified, it is set to `true` by default.

dynamic (optional)

Input. Specifies whether to create a static or a dynamic URL for the asset when the page is being rendered by the Export to Disk publishing method. `true` means to create a dynamic URL even if `rendermode` is set to `export`. `false` means to create

a static URL if `rendermode` is set to `export`. If this parameter is not specified, it is set to `false` by default.

blobheadername (optional)

Specifies HTTP header variables to suit your needs. The `blobheadername` parameter is used in conjunction with the `blobheadervalue` parameter to produce name/value pairs. This parameter represents the variable name.

blobheadervalue (optional)

Used in conjunction with `blobheadername` to specify and represent HTTP header variable values.

csblobid (optional)

Value of this parameter must correspond to a session variable of the same name when BlobServer security is on. When a request is made to the BlobServer, the `img src` service parameter sends a new request to the application server after the browser reads the containing page. BlobServer then verifies if the `csblobid` parameter was set. When BlobServer security is set to true and the `csblobid` value matches the session variable value, the blob is served. If the values do not match, a security violation is reported and no blob is served.

Description

This tag creates a BlobServer URL.

If `rendermode` is set to `export`, the tag creates a static URL; that is, a file name for an HTML file. It creates the URL based on the parameters described in the following section. These are a set of arguments that specify the table the blob is stored in, the name of the column that contains the binary data, the name of the primary key of the table, the value in that column to use to identify the object, and so on.

If `rendermode` is set to `live`, the tag creates a dynamic URL by concatenating the following:

- Name of the server (which is the same server as the one serving the page that is requesting the URL).
- Approot (application root) of the application server software, which it obtains from the `ft.approot` property in the `futuretense.ini` file.
- CGI path, which it obtains from the `ft.cgipath` property in the `futuretense.ini` file. For example, for WebLogic this path is `/servlet/` and for iAS6, this path is `/NASapp/cs/`.
- String `BlobServer?blobtable=`
- Name of the table the blob is stored in, passed in through a parameter that is described in the following section.
- Other arguments, passed in as parameters that are described in the following section.

Based on this information, `render:getbloburl` creates a URL for the blob asset and returns it to the element that requested it in a variable specified by the `outstr` parameter. The requesting element can then use the variable in an HTML tag to create the link.

To create BlobServer URL embedded in an HTML tag, use the [render:satelliteblob](#) tag.

You can use the `blobheadername` and `blobheadervalue` parameters to specify http headers in the BlobServer response. For example, you can specify the content-type as follows:

```
blobheadername1="Content-Type" blobheadervalue1="image/gif"
```

If you want to use http headers to prevent browser caching of BlobServer requests for all browsers, use both HTTP 1.0 and HTTP 1.1 cache control directives. In the following example, the first header (Cache-Control:no-cache) is the HTTP 1.1 directive and the second header (Pragma:no-cache) is the HTTP 1.0 directive.

```
blobheadername1="Cache-Control" blobheadervalue1="no-cache"
blobheadername2="Pragma" blobheadervalue2="no-cache"
```

errno

None.

Example

This code creates a BlobServer URL for an ImageFile asset identified through an ID passed in with blobwhere="Variables.asset:id" and stores it in a variable named "theURL":

```
<render:getbloburl blobtable="ImageFile" blobcol="urlpicture"
blobheader="Variables.asset:mimetype" blobkey="id"
blobwhere="Variables.asset:id" outstr="theURL"/>
```

See Also

[render:satelliteblob](#)

[render:packargs](#)

render:getpageurl

Creates a URL for an asset, processing the arguments passed to it from the calling element into a URL-encoded string and returning it in a variable.

Syntax

```
<render:getpageurl
  outstr="theURLVariable"
  pagename="SiteCatalogPageEntry"
  cid="Assets ID"
  [p="NameOfParentPage"]
  [c="AssetType"]
  [addsession="true/false"]
  [dynamic="true/false"]
  [packedargs="stringFromPACKARGStag"]>
  [<render:argument name="xxx" value="y"/>]
</render:getpageurl>
```

Parameters

outstr (required)

Input. Name of the variable that stores the URL generated by this tag.

pagename (required)

Input. Page name from a SiteCatalog entry. The page name can be referred to by a specific name or it can be determined through some combination of the four standard CS-Direct page criteria variables (c, cid, ct, and p). For example:
pagename="BurlingtonFinancial/Article/Variables.ct"

cid (required)

Input. ID of the asset that the URL represents.

p (optional)

Input. ID of the parent page. If a value for `p` is not explicitly provided, the tag tries to determine whether a value exists.

c (optional)

Input. Type of the asset. This parameter is optional, but if `c` (asset type) is not supplied, `rendermode` is set to `export`, and assets of this type have a **Filename** field, then the file name that the Export to Disk publishing method creates for the asset does not include the information from the **Filename** field of the asset. FatWire strongly recommends that you supply a value for `c`.

addsession (optional)

Input. Whether or not to include session IDs in the URL when a browser is set to reject cookies. `true` means to encode session IDs in the URL. If this parameter is not specified, it is set to `true` by default.

dynamic (optional)

Input. Whether to create a static or a dynamic URL for the asset when the page is being rendered by the Export to Disk publishing method. `true` means to create a dynamic URL even if `rendermode` is set to `export`. `false` means to create a static URL if `rendermode` is set to `export`. If this parameter is not specified, it is set to `false` by default.

packedargs (optional)

Input. Previously created URL-encoded packed arguments, in standard form; that is, the output of a previous [render:packargs](#) tag.

render:argument (optional)

Input. Name/value pairs to be passed in and included in the URL.

Description

This tag creates URLs for assets that are not blobs.

If `rendermode` is set to `export`, the tag creates a static URL (that is, a file name for an HTML file) and returns it as a variable. It creates the URL based on the standard CS-Direct page criteria variables `c`, `cid`, and `p`, a page name from a SiteCatalog entry, and other parameters that are described in the following section.

If `rendermode` is set to `live`, the tag creates a dynamic URL by concatenating and encoding the following:

- Name of the server (which is the same server as the one serving the page that is requesting the URL).
- Approot (application root) of the application server software, which it obtains from the `ft.approot` property in the `futuretense.ini` file.
- CGI path, which it obtains from the `ft.cgipath` property in the `futuretense.ini` file. For example, for WebLogic this path is `/servlet/` and for iAS6, this path is `/NASapp/cs/`.
- `String ContentServer?pagename=`
- Name from a SiteCatalog entry, passed in as a parameter that is described in the following section.
- Information passed in with parameters for the the standard CS-Direct page criteria variables, `c`, `cid`, and `p`, that are also described in the following section.

Based on this information, the tag creates a URL for the asset and returns it to the element that requested it in the variable specified by the `outstr` parameter. The requesting element can then use the variable to create a hyperlink for the asset.

Note

You use `render:getpageurl` to create URLs for CS-Direct assets. You cannot use it to create links to other Web sites, HTML files that are created outside of the Export publishing process, and so on.

errno

None.

Example

This code creates a URL for an article asset and stores it in a variable named “theURL”:

```
<render:getpageurl pagename="BurlingtonFinancial/Article/Full"
cid='<%=ics.GetVar("asset.id")%>' c="Article" ostr="theURL"/>
```

See Also

[render:packargs](#)

[render:getbloburl](#)

render:logdep

Handles an asset dependency during the export and approval process.

Syntax

```
<render:logdep
    [asset="asset name"]
    [cid="asset id"]
    [c="asset type"]
    [deptype="exact|exists|greater"]/>
```

Parameters

asset (optional)
Name of the asset.

cid (optional)
ID of the asset. When using the `cid` parameter you must also specify the `c` parameter. Either `asset` or `cid` and `c` must be set.

c (optional)
Type of asset. This parameter must be used in conjunction with the `cid` parameter. Either `asset` or `cid` and `c` must be set.

deptype (optional)
Specifies approval dependency. Legal values are as follows:

- **exact** – Specifies a version dependency. The version of the dependent asset must be equal to the version of the asset when it was approved.
- **exists** – Specifies that any version of the asset satisfies the dependency condition.
- **greater** – Specifies the asset must be at a or above a certain version.

Description

This tag keeps track of asset dependencies during publishing. When the `render : logdep` tag is encountered during an export publishing operation it marks assets as dependent on the asset being approved. Use the `render : logdep` tag when methods other than [asset:load](#) are employed to insert data into templates.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-10006	The object ID is not valid.

Release Introduced

This tag was introduced in dCS release 4.0.

render:packargs

Encodes in the URL the name/value argument pairs passed in and writes them to an output string, in standard form, with ampersand characters (&) separating each name/value pair.

Syntax

```
<render:packargs
  outstr="myPackedArguments"
  [packedargs="cid=1234&c=Article"]>
  [<render:argument name="arg1" value="value1"/>]
</render:packargs>
```

Parameters

outstr (required)

Input. Name of a variable to create that stores the encoded output string.

render:argument (optional)

Input. Name/value pairs to be packed by the tag. The names and values of these arguments are URL- encoded and incorporated into the final URL.

An argument passed in with this parameter takes precedence over an argument passed in with the `packedargs` parameter. If the same argument is passed in with both, the `packedargs` version is overridden by the `argument` version.

packedargs (optional)

Input. Previously created URL-encoded packed arguments, in standard form; that is, the output of a previous [render:packargs](#) tag.

Description

The `GetPageURL` element encodes (“packs”) the four standard CS-Direct arguments (c, cid, ct, p) before creating a dynamic URL with them. If you need to include additional arguments in a URL, use this tag to URL-encode them (“pack” them) before passing them to the `GetPageURL` element.

Use this tag to pack up the arguments and store them as a string. Then, you pass that string to the `GetPageURL` element through the `packedargs` parameter of a [render:callelement](#) tag.

errno

The possible values of `errno` include:

Value	Description
-10004	An argument is missing.

Example

This code packs the name/value pair `foo=bar` into a URL-encoded string, calls the `GetPageURL` element, and passes in several arguments, including the URL-encoded (packed) one :

```
<render:packargs outstr="myPackedArgs">
  <render:argument name="foo" value="bar"/>
</render:packargs>

<render:callelement name="OpenMarket/Xcelerate/Actions/Publish/
GetPageURL"/>
  <render:argument name="packedargs"
value='<%=ics.GetVar("myPackedArgs")%>'/>
  <render:argument name="c" value="Article"/>
  <render:argument name="cid" value='<%=ics.GetVar("id")%>'/>
</render:callelement>
```

See Also

[render:encode](#)

render:satelliteblob

Retrieves a blob from a table or from the CS-Satellite cache.

Syntax

```
<render:satelliteblob
  service="HTML tagName"
  blobtable="blobTable"
  blobkey="primaryKeyName"
  blobwhere="primaryKeyValue"
  blobcol="column name"
  blobheader="MIMEtype"
  [blobheadernamen="blob header name"]
  [blobheadervalue="blob header value"]
  [cachecontrol="expirationDateAndTime"]
  [csblobid="session variable value" />
  [<render:argument name="format1" value="5"/>]
</render:satelliteblob>
```

Parameters

service (required)

Input. Start of the HTML tag to hold the reference to the blob. For example, if the blob is an image, use `service="IMG SRC"`.

blobtable (required)

Input. Name of the CS-Direct table that stores assets of this type. For example, `blobtable="ImageFile"`.

blobkey (required)

Input. Name of the column used as the primary key. Typically, this is `"id"`.

blobwhere (required)

Input. Object ID of the asset.

blobcol (required)

Input. Name of the column that contains the binary data. For example, for the imagefile asset type, `blobcol="urlpicture"`.

blobheader (required)

Mimetype for returned data, in the form *description/extension*. For example, the mimetype for a GIF image is `image/gif`.

blobheadernamen (optional)

Specifies HTTP header variables to suit your needs. The `blobheadernamen` parameter is used in conjunction with the `blobheadervalue` parameter to produce name/value pairs. This parameter represents the variable name.

bloheadervalue (optional)

Used in conjunction with `blobheadername` to specify and represent HTTP header variable values.

render:argument (optional)

Input. Name/value modifiers for the HTML tag. For example:

```
<render:argument name="vspace" value="5"/>
<render:argument name="hspace" value="5"/>
```

cachecontrol (optional)

Input. Determines for how long to cache the blob. See [satellite:blob](#) for details about this parameter.

csblobid (optional)

Value of this parameter must correspond to a session variable of the same name when BlobServer security is on. When a request is made to the BlobServer, the `img src` service parameter sends a new request to the application server after the browser reads the containing page. BlobServer then verifies if the `csblobid` parameter was set. When BlobServer security is set to true and the `csblobid` value matches the session variable value, the blob is served. If the values do not match, a security violation is reported and no blob is served.

Description

This tag is the CS-Direct equivalent of the CS-Satellite [satellite:blob](#) tag, with the following exception:

- `rendermode` is passed in automatically.

You can use the `blobheadername` and `blobheadervalue` parameters to specify http headers in the BlobServer response. For example, you can specify the content-type as follows:

```
blobheadername1="Content-Type" blobheadervalue1="image/gif"
```

If you want to use http headers to prevent browser caching of BlobServer requests for all browsers, use both HTTP 1.0 and HTTP 1.0 cache control directives. In the following example, the first header (`Cache-Control:no-cache`) is the HTTP 1.1 directive and the second header (`Pragma:no-cache`) is the HTTP 1.0 directive.

```
blobheadername1="Cache-Control" blobheadervalue1="no-cache"
blobheadername2="Pragma" blobheadervalue2="no-cache"
```

To determine how long to cache a page or pagelet, these criteria are evaluated in this order:

1. A value passed in using the `cachecontrol` parameter.
2. If there is no value in `cachecontrol`, the value in the `satellite.page.cachecontrol.default` property, found in the `futuretense.ini` property file, is used.
3. If there is no value in `satellite.page.cachecontrol.default`, the value of the `expiration` property, located in the `satellite.ini` property file, is used.

FatWire recommends that you set values for the `satellite.page.cachecontrol.default` properties that make sense for the majority of your pages and pagelets. If you want to override the default expiration time, use the `cachecontrol` parameter. Use the `expiration` property to determine the timeout for pages that are not generated from this tag; for example, pages that are generated directly from a URL.

errno

None.

Example

This code creates an HTML `<IMG SRC>` tag. The SRC is the blob in the ImageFile table identified through the ID passed in with `blobwhere='<%=ics.GetVar("asset:id")%>'` and both its horizontal and vertical spacing are at five pixels:

```
<render:satelliteblob blobtable="ImageFile" blobkey="id"
blobcol="urlpicture" blobwhere='<%=ics.GetVar("asset:id")%>'
blobheader='<%=ics.GetVar("asset:mimetype")%>' service="IMG SRC">
  <render:argument name="alt"
    value='<%=ics.GetVar("asset:alttext")%>' />
  <render:argument name="hspace" value="5"/>
  <render:argument name="vspace" value="5"/>
</render:satelliteblob>
```

See Also

[render:satellitepage](#)

[satellite:blob](#)

render:satellitepage

Retrieves a page or pagelet identified through a page name and a set of arguments and specifies its cache expiration.

Syntax

```
<render:satellitepage
  pagename="nameOfPageEntry"
  [<cachecontrol="expiration_date_and_time"/>]
  [<render:argument name="variable1" value="value1"/>]
</render:satellitepage>
```

Parameters

pagename (required)

Input. The name of the page to invoke.

cachecontrol (optional)

Input. Determines for how long to cache the pagelet. See [satellite:page](#) for details about this parameter.

render:argument (optional)

Input. Name/value pairs to pass to the page entry. For example:

```
<render:argument name="c" value="Article"/>
```

Description

This tag is the CS-Direct equivalent of the `satellite:page` tag with the following exception:

- `rendermode` is passed to the element automatically.

Use this tag to serve a CS-Direct page or pagelet when you are using CS-Satellite on your system. The page or pagelet must have a page entry in the SiteCatalog. If the specified

pagelet is already cached on a CS-Satellite, this tag loads it. If the specified pagelet is not already cached, this tag caches it and loads it.

If an element is coded with a `render:satellitepage` tag but CS-Satellite is not present, `render:satellitepage` passes the call to [render:contentserver](#) which then serves the page or pagelet.

To determine how long to cache a page or pagelet, these criteria are evaluated in this order:

1. A value passed in using the `cachecontrol` parameter.
2. If there is no value in `cachecontrol`, the value in the `satellite.page.cachecontrol.default` property, found in the `futuretense.ini` property file, is used.
3. If there is no value in `satellite.page.cachecontrol.default`, the value of the `expiration` property, located in the `satellite.ini` property file, is used.

FatWire recommends that you set values for the `satellite.page.cachecontrol.default` properties that make sense for the majority of your pages and pagelets. If you want to override the default expiration time, use the `cachecontrol` parameter. Use the `expiration` property to determine the timeout for pages that are not generated from this tag; for example, pages that are generated directly from a URL.

errno

None.

Example

The following code calls for the page entry of a template named `LeadSummary`, passing the ID of the asset that it should display, the ID of the asset's parent page, and the name of a child template:

```
<render:satellitepage
  pagename="BurlingtonFinancial/Article/LeadSummary">
  <render:argument name="cid" value='<%=theArticles.getValue(oid)%>' />
  <render:argument name="ct" value="Full" />
  <render:argument name="p" value='<%=ics.GetVar("p")%>' />
</render:satellitepage>
```

See Also

[render:contentserver](#)

[render:satelliteblob](#)

[satellite:page](#)

render:stream

Use this tag as an alternative to [CSVAR](#) to display asset field that may have embedded links or inclusions.

Syntax

```
<render:stream
  value="variable_string"
  variable="variable name"
  ssvariable="session variable name"
  property="futuretense.ini_property"
  list="list_name"
  column="column_name" />
```

Parameters

value (optional)

You must use at least one of the following four parameters: `variable`, `ssvariable`, `property`, or `list` and `column` (`list` and `column` must be used together). Name of the inline string that you want to stream to a page. For example, for a variable called `foo`, set the parameter equal to `variables.foo`.

variable (optional)

You must use at least one of the following four parameters: `variable`, `ssvariable`, `property`, or `list` and `column` (`list` and `column` must be used together). Name of the content server variable that you want to stream to a page. For example, for a variable called `foo`, set the parameter equal to `foo` rather than to `variables.foo`.

ssvariable (optional)

You must use at least one of the following four parameters: `variable`, `ssvariable`, `property`, or `list` and `column` (`list` and `column` must be used together). Name of the Content Server session variable that you want to stream to a page.

property (optional)

You must use at least one of the following four parameters: `variable`, `ssvariable`, `property`, or `list` and `column` (`list` and `column` must be used together). A Content Server property from the `futuretense.ini` property file.

list (optional)

You must use at least one of the following four parameters: `variable`, `ssvariable`, `property`, or `list` and `column` (`list` and `column` must be used together). Name of the list

column (optional)

You must use at least one of the following four parameters: `variable`, `ssvariable`, `property`, or `list` and `column` (`list` and `column` must be used together). Name of the field in the list.

render:unknowndeps

Describes the current asset dependencies as incomplete.

Syntax

```
<render : unknowndeps />
```

Parameters

None.

Description

When the `render : unknowndeps` tag is used during an export publish operation it marks the asset's dependencies as incomplete. This tag makes every publish operation republish the specified asset rather than retrieving it from cache to ensure the data contained in the asset is current. When a template using this tag is employed during rendering on the live mirror site or during preview it instructs the cache manager not to cache the pagelet.

This tag should be used in cases when pages have undeterminable connections to dependents. For example use this tag when employing a query in a template which yields a different resultset each time it is generated.

errno

None.

Release Introduced

This tag was introduced in dCS release 4.0.

Site Plan Tags

The CS-Direct site tree is an outline or site plan, located on the SitePlan tab, that displays the page assets in your site. You use the `siteplan` tags to extract information from the CS-Direct database about a page asset's position in the site tree hierarchy. You can then display that information on your on-line site in site maps, navigation bars, and so on.

CS-Direct stores information about page assets in several database tables:

- Page table, which stores information about the page asset as an asset (its name, its object ID, who created it, which template it uses, and so on). Use the `asset` tags to obtain information from that table.
- AssetRelationTree table, which stores information about the other assets that page assets have relationships with through named or unnamed associations. Use the [asset:children](#) tag to obtain information from that table.
- AssetPublication table, which specifies which management sites (publications) give you access to the page. Page assets cannot be shared between sites so there is only one row entry for a page because it can have only one **pubid**. A pubid is a unique value that identifies a site.
- SitePlanTree table, which stores information about the page asset's hierarchy in the site structure. Use the `siteplan` tags to obtain information from that table.

The SitePlanTree table is a standard Content Server object tree with nodes and children. It stores information such as a page asset's node ID, the node ID of its parent, whether it is placed or unplaced, and so on.

If you want to display information about a page asset as an asset, use the `asset` tags to retrieve it. (See [Asset Tags](#).) If you want to display information about a page asset's hierarchy in the site structure, use the `siteplan` tags to retrieve it. See also [asset:search](#).

Note

Each JSP file containing CS-Direct JSP `siteplan` tags must have the following directive at the top:

```
<%@ taglib prefix="siteplan" uri="futuretense_cs/
siteplan.tld" %>
```

The following section describes each site plan tag, beginning with [siteplan:children](#).

siteplan:children

Queries the SitePlanTree table and then builds a list of the child nodes (pages) for the specified node (page).

Syntax

```
<siteplan:children
  name="nodeName"
  list="listName"
  [objecttype="Page"]
  [objectid="ObjectIDofSpecificChildNode"]
  [code="Placed" or "Unplaced"]
  [order="fieldName"]/>
```

Parameters

name (required)

Input. Name of the parent node whose children you want to list. This parent node must be loaded and assigned a name before you can pass its name to this tag. (See [siteplan:load](#).)

list (required)

Input. Name to assign to the list of the node's children. This list holds values for all of the fields in the SitePlanTree table. If a list with this name already exists, it is replaced.

objecttype (optional)

Input. Restricts the list to nodes of a specific type and that type should always be "Page." You can combine this parameter with `objectid` to request one specific node.

objectid (optional)

Input. Object ID of a specific child node. If you supply `objectid`, you must also supply `objecttype`.

code (optional)

Input. Name of the association that describes the relationship between the child node and the parent node. Valid codes are `Placed` and `Unplaced`.

order (optional)

Input. Fields to sort the list by and whether the sort on those fields is ascending (`asc`) or descending (`desc`). By default, the sort is ascending. If you specify more than one field, separate the field names with a comma.

For example, if you specify `order="nrank"`, the list is sorted by rank starting at number 1. If you want the list sorted by descending rank, use `order="nrank, desc"`.

Description

This tag queries the SitePlanTree table for a list of the child nodes of the node that you specify, listing each child with a value for all of the fields from that table (`nid`, `nparentid`, `nrank`, `otype`, `oid`, and `ncode`). You must load the parent node ([siteplan:load](#)) before you can invoke this tag to query for and list its children. The list is a standard Content Server list.

You can restrict the list of children nodes by association name (`code`), object type, object ID, and rank (`order`).

If you specify `objectype="Page"`, the list of children is a join of the SitePlanTree and the Page table. In other words, you do not need to perform a subsequent [siteplan:load](#) or [asset:load](#) (both of which are direct database queries that take time) to obtain information about the child pages. You do need to use more [siteplan:load](#) tags if you want to continue traversing the hierarchy in the SitePlanTree table; that is, if you want to extract grandchildren pages, and so on.

errno

The possible values of `errno` include:

Value	Description
-111	The asset has no children.
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-10006	The object ID is not valid.
-10007	The version of the object is not valid.

Example

This code extracts the child pages for the page asset identified as “parentNode,” stores them in the list named “PlacedPages,” orders them by rank, and restricts the list to include only placed pages:

```
<siteplan:children name="parentNode" objectype="Page"
  list="PlacedPages" code="Placed" order="nrnk"/>
```

See Also

[siteplan:listpages](#)

siteplan:get

Retrieves and lists the value of a single field from a loaded site plan node.

Syntax

```
<siteplan:get
  name="nodeName"
  field="fieldName"
  [output="variableName"]/>
```

Parameters

`name` (required)

Input. Name of the site plan node whose field value you want to retrieve. This node must be loaded and assigned a name before you can pass its name to this tag. (See [siteplan:load](#).)

field (required)

Input. Name of the field whose value you want to retrieve.

output (optional)

Input. Name to assign to the output variable that stores the value of the field. If you do not supply a name, CS-Direct creates a variable with the same name as that of the field.

Description

This tag obtains the value (or contents) from the specified field of a loaded node, and stores it in the specified output variable.

Use this tag to extract the parent node ID in order to trace the parentage of a page, which enables you to then display a link to the parent of the page that is currently displayed.

errno

The possible values of errno include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

Example

This code extracts the value of the “oid” field for the site plan node identified as “pageNode” and stores it in an output variable named “objectid.”

```
<siteplan:get name="pageNode" field="oid" output="objectid"/>
```

See Also

[siteplan:load](#)

siteplan:getpagename

Retrieves the page asset name of the specified node. This is an asset name, not a page name from the SiteCatalog table.

Syntax

```
<siteplan:getpagename
  name="nodeName"
  output="variableName" />
```

Parameters**name (required)**

Input. Name assigned to the site plan node whose page name you want to retrieve. This node must be loaded and assigned a name before you can pass its name to this tag. (See [siteplan:load](#).)

output (required)

Input. Name to assign to the output variable that stores the name of the page.

Description

This tag returns the page name of the specified node; that is, the value entered into the **Name** field on the CS-Direct **New/Edit Page** form for the page asset that corresponds to this node.

You can use this tag to obtain the page name of the node so that you can display that name in a navigation bar.

errno

The possible values of errno include:

Value	Description
-108	Tree selector error.
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-10006	The object ID is not valid.
-10007	The version of the object is not valid.
-10008	The node ID of the object is not valid.
-10009	More than one object met the specified criteria.

Example

This code loads the site plan and then retrieves the page asset name of a specific site plan node:

```
<siteplan:load name="BusinessNewsNode"
  id='<%=ics.GetVar("nodeid")%>' />
<siteplan:getpagename name="BusinessNewsNode"
output="PageAssetName"/>
```

See Also

[siteplan:get](#)

[asset:getassettype](#)

siteplan:listpages

Queries the SitePlanTree table and then creates a list of pages, starting with the page node that you specify.

Syntax

```
<siteplan:listpages
  name="nodeName"
  placedlist="PlacedPages"
  [level="1"]/>
```

Parameters

name (required)

Input. Name assigned to the site plan node whose unplaced or placed pages you want to list. This node must be loaded and assigned a name before you can pass its name to this tag. (See [siteplan:load](#).)

placedlist (required)

Input. Name to assign to the list of placed pages.

level (optional)

Input. Number of node levels to list, starting with the node identified by the name parameter. The default is 10000.

Description

This tag creates a hierarchical list of the pages in the site plan. You can determine which page node to start with and how many levels in the hierarchy to list.

The list contains the following column names:

- **PageName** – Name of the page.
- **ParentName** – Name of the parent in the tree.
- **NodeLevel** – Level in the tree relative to the parent.
- **PlacedValue** – Specifies if the page is placed or unplaced.
- **AssetType** – Type of asset.
- **Id** – Asset's ID.
- **HasChildren** – Specifies if the asset has children; possible values are `true` or `false`.
- **Nid** – Node ID of SitePlan tree entry.

errno

The possible values of `errno` include:

Value	Description
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-10006	The object ID is not valid.
-10007	The version of the object is not valid.
-10008	The node ID of the object is not valid.
-10009	More than one object met the specified criteria.

Example

This code lists all the placed pages under the site plan node identified as “thePubNode” to one level in the tree; that is, it lists the child pages for “thePubNode” but not its grandchildren, or great-grandchildren, and so on.

```
<siteplan:listpages name="thePubNode" placedlist="placedPages"
level="1"/>
```

siteplan:load

Queries the database for the specified site plan tree node and then loads an instance of the site plan into memory as an object.

Syntax

```
<siteplan:load
    name="name"
    nodeid="nodeid"/>
```

Parameters

name (required)

Input. Name of top-level node of the site plan to load.

nodeid (required)

Input. Node ID of the top-level node in the site plan.

Description

This tag executes a database query to retrieve an instance of the specified site plan tree node and then saves the instance into memory (session information), assigning it the name provided by the name parameter. That object is then available until the session is flushed or the name is overwritten.

To determine the nodeid of the site plan tree node that you want to load, you can either use the [asset:getsitenode](#), or [siteplan:listpages](#) tags.

You must load the site plan before you can use any of the other `siteplan` tags, except [siteplan:listpages](#) and [siteplan:root](#).

errno

The possible values of `errno` include:

Value	Description
-108	Tree selector error.
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-10006	The object ID is not valid.
-10007	The version of the object is not valid.
-10008	The node ID of the object is not valid.
-10009	More than one object met the specified criteria.

Example

This code loads a page asset, extracts the identity of its site node, and then loads the site plan with that page asset as the top-level page:

```
<asset:load name="BusinessNews" type="Page"
objectid='<%=ics.GetVar("id")%>' />
<asset:getsitenode name="BusinessNews" output="PageNodeId"/>
<siteplan:load name="ParentNode"
nodeid='<%=ics.GetVar("PageNodeId")%>' />
```

See Also

[asset:load](#)

[asset:getassettype](#)

siteplan:nodepath

Returns a list of the currently loaded site node's ancestors in the SitePlanTree.

Syntax

```
<siteplan:nodepath  
    name="node"  
    list="list name" />
```

Parameters

name (required)

Name of the node whose children you want to get.

list (required)

Name of the list in which to put the results.

Description

Returns a list of the currently loaded site node's ancestors in the SitePlanTree. The list has all the columns of the SitePlanTree and does not include the current object. The first item in the list is its immediate parent, the second is that node's parent, and so on.

errno

The possible values of `errno` include:

Value	Description
-3	No access privileges to table.
-103	No table.
-105	Database error.
-110	Invalid tree name.

Release Introduced

This tag was introduced in dCS release 5.0.

siteplan:place

Marks the page as Placed and assigns it a place in the hierarchy.

Syntax

```
<siteplan:place
  name="node"
  [nodeid="page node ID"]
  [childid="page ID"]
  [type="asset type"]
  [rank="page rank"] />
```

Parameters

name (required)

Name of the node under which to place a page.

nodeid (optional)

You can specify either **nodeid** or **childid** and **type**. Node ID of the page to place under named page.

childid (optional)

Use this parameter when the **type** parameter is specified. With **type**, this parameter identifies the page asset to place under the loaded Site Plan node.

type (optional)

Used when the **childid** is specified. With **childid**, this parameter identifies the page asset to place under the loaded Site Plan node.

rank (optional)

Specifies the rank under the parent.

errno

The possible values of **errno** include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-10007	Rank is not specified.

siteplan:root

Queries the SitePlanTree table for a root node of the site plan.

Syntax

```
<siteplan:root
    list="rootNode"
    [objectid="object ID"]/>
```

Parameters

list (required)

Input. Name to assign to the list that stores the root node. The columns of the list match those from the SitePlanTree table. If a list with this name already exists, it is replaced with this new list.

objectid (optional)

Input. Pubid of a site (formerly publication).

Description

Use this tag to extract and display information about the site structure from the top down.

If you use the `objecttype` parameter, the result is a join of the SitePlanTree and the Publication table (only a publication can be a root). The name and description fields in the Publication table are then accessible without requiring a subsequent [siteplan:load](#) for the site plan.

errno

The possible values of `errno` include:

Value	Description
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing.
-10006	The object ID is not valid.
-10007	The version of the object is not valid.
-10008	The node ID of the object is not valid.

Example

This code retrieves the root node identified by the object ID and stores it in a list named "PubRoot":

```
<siteplan:root list="PubRoot" objectid="968251170475"/>
```

siteplan:setobject

Sets an object in a node.

Syntax

```
<siteplan:setobject
  name="node name"
  object="page asset object"/>
```

Parameters

name (required)
Name of a loaded Site Plan node.

object (required)
Name of a loaded page asset.

Description

The OTYPE and OID fields of the SitePlanTree row are updated.

errno

The possible values of errno include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

siteplan:setparent

Sets the parent of a node.

Syntax

```
<siteplan:setparent
  name="node"
  parent="loaded node"/>
```

Parameters

name (required)
Name of a loaded Site Plan node.

parent (required)
Name of a loaded node of the Site Plan node.

Description

Returns a list of the currently loaded site node's ancestors in the SitePlanTree. The NPARENTID field of the SitePlanTree row is updated.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

siteplan:unplace

Removes a Site Plan node from the `Placed` part of the tree and sets the `NCODE` field to `Unplaced`.

Syntax

```
<siteplan:unplace
  name="node name"
  nodeid="child page NID" />
```

Parameters

`name` (required)

Name of a loaded Site Plan node.

`nodeid` (required)

NID (node ID) of the child page to be removed from `Placed`.

Description

Returns a list of the currently loaded site node's ancestors in the `SitePlanTree`.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

siteplan:unplacedancestor

Returns whether the specified node has any pages above it in the hierarchy that have an Unplaced ncode.

Syntax

```
<siteplan:unplacedancestor  
  name="node name"  
  output="variablename" />
```

Parameters

name (required)

Name of a loaded Site Plan Node.

output (required)

Name of the output variable. This variable is set to `true` if the page asset in question has an unplaced ancestor.

Description

Returns a list of the currently loaded site node's ancestors in the SitePlanTree.

errno

The possible values of `errno` include:

Value	Description
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).

Chapter 11

CS-Direct Advantage JSP Tags

This chapter describes the functionality and JSP syntax of the object methods. These methods are implemented as Content Server custom application tags, which invoke application business logic for applications built with Content Server. It serves as a reference for the tags that you use to perform the following tasks:

- Create catalog pages
- Create shopping carts
- Implement catalog searching
- Display selected product information

Use the CS-Engage object tags, described in [Chapter 12, “CS-Engage JSP Tags,”](#) to gather data about visitors to your site, and to display recommendations or promotions (merchandising information) based on that data.

Assetset Tags

An **assetset** is a collection of assets that can be described by either a list or a search constraint. An assetset can include different types of assets. It is not limited to assets of one type.

If you are using CS-Engage, there are additional assetset tags available. See [Chapter 12, “CS-Engage JSP Tags”](#) for information.

Note

Each JSP file containing JSP assetset tags must have the following directive at the top:

```
<%@ taglib prefix="assetset" uri="futuretense_cs/
assetset.tld" %>
```

The following section describes each assetset tag, beginning with [assetset:getassetcount](#).

assetset:getassetcount

Retrieves an asset count.

Syntax

```
<assetset:getassetcount
    name="assetsetname"
    [list="attriblist"]
    varname="varname"/>
```

Parameters

name (required)

Input parameter. Name of the assetset object.

list (optional)

Input parameter. Name of the list that determines sort order. This list has three columns:

- **attributetypename**
- **attributename** – Name of attribute, sort by or has one of the following special values:
 - _RATING_** – order by asset rating (CS-Engage only)
 - _ASSETTYPE_** – order by asset type
- **direction** – Either ascending or descending

varname (required)

Input and output parameter. As input, name of the variable that holds the assigned count on output.

Description

When called, `assetset:getassetcount` creates the specified variable with the count of table rows that results if [assetset:getassetlist](#) were called with similar parameters.

assetset:getassetlist

Retrieves an ordered list of assets, given optional sort criteria.

Syntax

```
<assetset:getassetlist
    name="assetsetname"
    [list="attriblist"]
    [maxcount="rowcount"]
    [method="random|highest"]
    listvarname="varname"/>
```

Parameters

name (required)

Input parameter. Name of an assetset object.

list (optional)

Input parameter. Name of the list that determines sort order. This list has three columns:

- **attributetypename**
- **attributename** – Name of attribute, sort by or has one of the following special values:
 - _RATING_** – order by asset rating (CS-Engage only)
 - _ASSETTYPE_** – order by asset type
- **direction** – Either ascending or descending

maxcount (optional)

Input parameter. Maximum number of rows to return in the list. A value of 0 (zero) indicates all. If count is less than the number that otherwise is returned, then items are selected according to the specified method argument.

method (optional)

Input parameter. Required only if the value of maxcount is less than the number of items described. method can have one of the following values:

- **random** – Random weighted selection based on rating.
- **highest** – Best selection based on rating.

This parameter is meaningful only in the context of CS-Engage.

listvarname (required)

Input and output parameter. As input, name of the list that holds the assigned values on output. The list has the following columns:

- **assettype**
- **assetid**

A column for each sort attribute name, prefixed by the string SORT_.

Description

Each row in the ordered list holds a value for each attribute that is used for sorting. For example, if an assetset represents a product that has two authors, (say Fred and Sam), and the result is specified to sort by author, then the result set has a column called SORT_author, with two rows (one for Fred and one for Sam).

Note

You can sort only on required values. If you sort on an attribute that does not exist for some assets, those assets are filtered from the results.

Example

```
<!-- Apply a sort list to an assetset and retrieve a resultset
list -->
<assetset:getassetlist name="myasset" list="listout"
listvarname="aslist"/>
```

assetset:getattributevaluecount

Gets the count of distinct attribute values for a specific attribute associated with the products represented by an assetset.

Syntax

```
<assetset:getattributevaluecount
  name="assetsetname"
  attribute="attribname"
  [typename="assettypename"]
  varname="varname"/>
```

Parameters

name (required)

Input parameter. Name of the assetset object.

attribute (required)

Input parameter. Name of an attribute whose values are to be looked up.

typename (optional)

Input parameter. Asset type name of the attribute. The default is set in the gator.ini file: mwb.defaultattributes=PAttributes.

varname (required)

Input and output parameter. As input, name of the variable that holds the count of the unique values requested on output.

assetset:getattributevalues

Gets the list of attribute values that are associated with the assets represented by an assetset.

Syntax

```
<assetset:getattributevalues
  name="assetsetname"
  attribute="attribname"
  [typename="assettypename"]
  listvarname="varname"
```



```
[ordering="ascending|descending"]/>
```

Parameters

name (required)

Input parameter. Name of the assetset object.

attribute (required)

Input parameter. Name of an attribute whose value should be looked up.

typename (optional)

Input parameter. Asset type name of the attribute. The default is set in the `gator.ini` file: `mwb.defaultattributes=PAtributes`.

listvarname (required)

Input and output parameter. As input, name of a list that holds a list of string values assigned on output and obtained from the assetset's attributes.

ordering (optional)

Input parameter. Indicates whether the result list should be in ascending or descending order; value can be either ascending or descending.

Example

```
<!-- Iterate through a list of attribute values. -->
<!-- First, create a blank searchstate. -->
<searchstate:create name="ss"/>
<!-- Associate the blank searchstate with a new asset set. -->
<assetset:setsearchedassets name="as" constraint="ss"/>
<!-- Get a set of all values for the given attribute. -->
<assetset:getattributevalues ordering="descending" name="as"
    attribute="price" listvarname="results"/>
```

assetset:getmultiplevalues

Scatters attribute values from several attributes (and potentially more than one product) into several specified lists.

Syntax

```
<assetset:getmultiplevalues
    name="assetsetname"
    list="listname"
    [byasset="true|false"]
    prefix="prefix"/>
```

Parameters

name (required)

Input parameter. Name of the assetset object.

list (required)

Input parameter (see as output, below). List of attribute names to scatter. This list has three columns:

- attributetypename
- attributename

- **direction** – Possible values include: none, ascending, or descending. If the **byasset** parameter is set to **true**, the value of this column must be set to none.

byasset (optional)

Input parameter. Boolean value indicating whether to scatter a different instance of each attribute for each individual asset in the assetset (**true**), or to group all the attribute values for all assets together into one list per attribute (**false**). The default is **false**.

prefix (required)

Input parameter. Starting part of the list names that this method creates.

list

Output parameter. If **byasset** is **false**, creates a set of lists with the names: **prefix:attrname**. If **byasset** is **true**, creates a set of lists with the names: **prefix:assetid:attrname**. If any list is empty (that is, has no values), does not create, change, or clear the list of that name at all.

Description

FatWire recommends using `assetset:getmultiplevalues` when the goal is to display a fixed-format table of assets, or to obtain many attributes of a single asset (such as for a product detail page).

`assetset:getmultiplevalues` has the following limitations:

- Only non-foreign attributes can be scattered.
- Text-type attributes cannot be scattered.

assetset:setasset

Builds an asset set from a single asset.

Syntax

```
<assetset:setasset
  name="assetsetname"
  type="assettype"
  id="assetid"
  [locale="localeobject"]
  [deptype="exact|exists|none"]/>
```

Parameters

name (required)

Input and output parameter. As input, name of the assetset object that holds the specified values on output.

type (required)

Input parameter. Internal asset name.

id (required)

Input parameter. Asset identifier.

locale (optional)

Input parameter. Name of a locale object, which determines sort order.

deptype (optional)

Specifies whether approval dependency is `exact`, `exists` or `greater`.

Legal values are:

- `exact` – Specifies a version dependency. The version of the dependent asset must be equal to the version of the asset when it was approved.
- `exists` – Specifies that any version of the asset satisfies the dependency condition.
- `none` (default) – Specifies no approval dependency.

Example

```
<!-- Get attributes for the specified product. -->
String aid = ics.GetVar("id");
<assetset:setasset name="as" type="products" id="id"/>
```

assetset:setempty

Creates an empty or null assetset object.

Syntax

```
<assetset:setempty
    name="assetsetName"
    [locale="localeobject"]
    [deptype="exact|exists|none"]/>
```

Parameters

name (required)

Input and output parameter. As input, name of the assetset object to create on output.

locale (optional)

Input parameter. Name of a locale object, which determines sort order.

deptype (optional)

Specifies whether approval dependency is `exact`, `exists` or `greater`.

Legal values are:

- `exact` – Specifies a version dependency. The version of the dependent asset must be equal to the version of the asset when it was approved.
- `exists` – Specifies that any version of the asset satisfies the dependency condition.
- `none` (default) – Specifies no approval dependency.

assetset:setlistedassets

Builds an assetset from a list of assets.

Syntax

```
<assetset:setlistedassets
  name="assetsetname"
  [assettypes="assettype"]
  assets="listname"
  [locale="localeobject"]
  [deptype="exact|exists|none"]/>
```

Parameters

name (required)

Input and output parameter. As input, name of the assetset object to create on output, representing the list of passed assets.

assettypes (optional)

Input parameter. Name of a list of asset types to include in building the set. The list has one column called `assettype`. If null, then all assets in the system are considered.

assets (required)

Input parameter. Name of a list containing the columns `assettype` and `assetid`. This list is used to create the assetset.

locale (optional)

Input parameter. Name of a locale object, which determines sort order.

deptype (optional)

Specifies the approval dependency. Legal values are:

- **exact** – Specifies a version dependency. The version of the dependent asset must be equal to the version of the asset when it was approved.
- **exists** – Specifies that any version of the asset satisfies the dependency condition.
- **none** (default) – Specifies no approval dependency.

Description

The `assetset:setlistedassets` tag builds an assetset where the assets are described by their type name and asset identifier.

assetset:setsearchedassets

Creates an assetset object which represents all assets of specific types narrowed by specified search criteria.

Syntax

```
<assetset:setsearchedassets
    name="assetsetname"
    [assettypes="assettype"]
    [constraint="searchstateobject"]
    [locale="localeobject"]
    [site="siteidentifier"]
    [deptype="exact|exists|none"]/>
```

Parameters

name (required)

Input and output parameter. As input, name of the assetset object to create on output, representing the specific asset types and search criteria.

assettypes (optional)

Input parameter. A string of asset types; for example, "Products, AArticles, AImages", to include in building the set. Identifying specific assets is recommended, since the search becomes much slower for each asset type searched. If null, then all assets in the system are considered.

constraint (optional)

Input parameter. Name of a searchstate object describing the search constraint.

locale (optional)

Input parameter. Name of a locale object, which determines sort order.

site (optional)

Input parameter. Site identifier to search.

deptype (optional)

Specifies approval dependency. Legal values are:

- **exact** – Specifies a version dependency. The version of the dependent asset must be equal to the version of the asset when it was approved.
- **exists** – Specifies that any version of the asset satisfies the dependency condition.
- **none** (default)– Specifies no approval dependency.

Example

```
<!-- Iterate through a list of attribute values. -->
<!-- First, create a blank searchstate. -->
<searchstate:create name="ss"/>
<!-- Associate the blank searchstate with a new asset set. -->
<assetset:setsearchedassets name="as" constraint="ss"
assettypes="products"/>
<!-- Get a set of all values for the given attribute. -->
<assetset:getattributevalues ordering="descending" name="as"
    attribute="price" listvarname="results"/>
```

Blob Service Tags

The following section describes each blob service tag, beginning with [blob service:getIdcolumn](#).

blob service:getIdcolumn

Returns the name of the key column to be passed to other blob management tags.

Syntax

```
<blob service:getIdcolumn  
  varname="variable name"/>
```

Parameters

varname (required)
Name of the variable that contains the column.

Example

The output from the following code sample is: id.

```
<blob service:getIdcolumn varname="blobIdColumn"/>
```

blob service:gettablename

Returns the name of the blob table to be passed to the blob management tags.

Syntax

```
<blob service:gettablename  
  varname="variable name"/>
```

Parameters

varname (required)
Name of the variable that contains the table.

Example

The output from the following code sample is: MungoBlobs.

```
<blob service:gettablename varname="blobTableName"/>
```

blobservice:geturlcolumn

Returns the name of a URL column to be passed to other blob management tags.

Syntax

```
<blobservice:geturlcolumn  
    varname="variable name"/>
```

Parameters

varname (required)

Name of the variable that contains the name of the column.

Example

The output from the following code sample is: urldata.

```
<blobservice:geturlcolumn varname="blobURLName"/>
```

blobservice:readdata

Uses an ID to load the file name and URL data of the specific blob.

Syntax

```
<blobservice:readdata  
    id="blob id"  
    listvarname="output list"/>
```

Parameters

id (required)

Identifies the blob to be read.

listvarname (required)

Name of the list containing the data. The list is created with one column called urldata.

Calculator Tags

The calculator tag provides basic calculator and Boolean functionality.

Note

Each JSP file containing JSP calculator tags must have the following directive at the top:

```
<%@ taglib prefix="calculator" uri="futuretense_cs/
calculator.tld" %>
```

The following section describes the calculator tag, [calculator:go](#).

calculator:go

Calculates the value of an expression.

Syntax

```
<calculator:go
    value="postfixstring"
    varname="result"/>
```

Parameters

value (required)

Input parameter. Postfix expression as a string with the following format:

"opreand1 operand2 ... operandn opchar"

where operand1, operand2, ..., operandn are one or more operands, and opchar is one of the following operator characters or strings:

+	-	*	/	%	=	!=
LT	GT	LTE	GTE	AND	OR	NOT

All values are separated by spaces, and the entire value input string is contained within double quotes.

varname (required)

Input and output parameter. As input, name of a string that holds the output that results from evaluating the expression.

Description

The value is computed using reverse polish notation. Numbers in the expression string are pushed onto a stack. Operations pop numbers off the stack as needed and push the result. The final answer is the top element of the stack.

Example

```
<!-- Divide total price by item quantity to get the item price. --
>
<calculator:go value="100 20 /" varname="calcout"/>
```



```
<!-- Convert the above item price to currency. -->  
<currency:create name="currencyxml"/>  
<currency:getcurrency name="currencyxml" value="10.08"  
  varname="gotcurrencyxml"/>
```

Cart Tags

A shopping cart object contains the following information:

- A list of product IDs, the quantity, the purchase amount, and the storeID for each product.

Note

The storeID number is assigned by your Transact administrator when you register an on-line store with Transact.

- A list of discount descriptions and amounts, which result from applying a campaign discount to the cart.
- A set of cart and item parameters required by the Commerce Connector.

See [Chapter 16, “CS-Direct Advantage Cart Parameters”](#) for cart parameter names and values. See [Chapter 17, “CS-Direct Advantage Order Item Parameters”](#) for individual item parameter names and values.

Note

Each JSP file containing JSP cart tags must have the following directive at the top:

```
<%@ taglib prefix="cart" uri="futuretense_cs/cart.tld" %>
```

The following section describes each cart tag, beginning with [cart:additem](#).

cart:additem

Adds an order item to the shopping cart.

Syntax

```
<cart:additem
  name="cartname"
  storeid="storeid"
  [type="products"]
  id="assetid"
  quantity="unitspurchased"
  price="totalprice"
  [list="orderitems"]
  [merge="true|false"]
  [insertbefore="rowid"]
  [varname="rowid"]/>
```

Parameters

name (required)

Input parameter. Name of the cart object.

storeid (required)

Input parameter. Positive integer, not more than eight digits, that identifies the store to CS- Direct Advantage. This number is assigned by your Transact administrator.

type (optional)

Input parameter. Type of item to add. The default is `Products`.

id (required)

Input and output parameter. As input, asset id of the order item that is added to the cart on output.

quantity (required)

Input parameter. Number of units of this item that the buyer is purchasing. If items are sold in packages, then `Quantity` should represent the number of packages, not the total number of items in those packages.

price (required)

Input parameter. Total price of this product, specified as an integer or a number with two decimal places. `PRICE` cannot be negative, `price` cannot hold a currency character such as a dollar sign (\$), and `price` cannot hold a comma (,) as used in many European countries. For example, 42.99 is an acceptable value of `price`, but \$42.99 and 42,999 are not.

For both tangible goods and digital goods, set `price` to be the total price of the item. If `quantity` is greater than 1, then `price` is the total price, not the price of an individual unit. For example, if you sell a gift set of three shirts for \$60, you set `quantity` to 3 and `price` to \$60. You do not set `Price` to \$20, the price of each individual shirt.

For digital subscriptions, set `price` to be the price of each installment payment. For example, if a subscription costs \$100, payable in four equal installments, you set `price` to 25.

list (optional)

Input parameter. List of order item parameters for the item.

merge (optional)

Input parameter. Boolean: whether to attempt to merge quantity, price with (possibly) existing item with the input ID (asset ID).

insertbefore (optional)

Input parameter. Row identifier of the item before which to insert this item if no merge occurs. The default is to insert at end.

varname (optional)

Input parameter. Row identifier for the item. This is always set by `additem`, not by the caller, and uniquely identifies the line item in the cart.

Description

See [Chapter 17, “CS-Direct Advantage Order Item Parameters”](#) for individual item parameter names and values.

cart:check

Checks the cart against Transact via the Commerce Connector.

Syntax

```
<cart:check
    name="cartname"
    varname="varname"/>
```

Parameters

name (required)

Input parameter. Name of the cart object.

varname (required)

Input parameter. Name of the variable that holds one of the following values:

- **true** – Cart is acceptable and can be committed to Transact.
- **false** – Cart is not acceptable to Transact.

Description

As a result of the check, `cart:check` updates the following parameters with legal values:

- **subTotal** – Total cost of items
- **shippingInfo-shippingCost** – Total cost of shipping.
shippingInfo-shipperID – Transact shipper GUID.
shippingInfo-shipperName – Transact shipper name.
- **tax-taxType** – Type (USSalesTax, Canadian, VAT)
tax-taxAmount – Total tax amount (if taxType is not Canadian)
tax-GST – Canadian GST
tax-PST – Canadian PST
- **shippingInfo-shippingTax-taxType** – Type (USSalesTax, Canadian, VAT)
shippingInfo-shippingTax-taxAmount – if taxType is not Canadian
shippingInfo-shippingTax-GST – Canadian GST
shippingInfo-shippingTax-PST – Canadian PST
- **totalSalesAmount** – Sum of subTotal, shippingCost, and tax-taxAmount.
- **store** – Set of parameters that come from the `transact.ini` file. (The Web developer is only concerned with `storeID` and `currencyCode`.)

The `cart:check` tag also returns an error code and error message if Transact cannot resolve any other field in the cart (for example, a missing postal code, or a postal code that is not legal for a specified city).

As a result of the check, the cart object is updated to reflect any errors or notifications retrieved from the Transact. Choices for fields that have choices also become available.

Example

```
<!-- Get the current cart. -->
<commercecontext:getcurrentcart varname="thiscart"/>
<!-- Check the cart to enumerate payment methods. -->
<cart:check name="thiscart" varname="cartstatus"/>
```

```
<!-- Now get the legal values for the payment types. -->  
<cart:getlegalvalues name="thiscart" field="paymentinfo-brandid"  
    listvarname="paylist"/>
```

cart:clear

Empties the cart.

Syntax

```
<cart:clear  
    name="cartname" />
```

Parameters

name (required)

Input parameter. Name of the cart object.

Description

Use `cart:clear` after a successful commit, or whenever the contents of the cart need to be cleared. When called, `cart:clear` clears the cart, including all contents, items, and discounts.

cart:cleardiscounts

Clears the cart's current discounts.

Syntax

```
<cart:cleardiscounts  
    name="cartname" />
```

Parameters

name (required)

Input parameter. Name of the cart object.

Description

Use this tag whenever the discounts in a cart need to be cleared.

cart:commit

Sends a checked cart to Transact via the Commerce Connector.

Syntax

```
<cart:commit
  name="cartname"
  varname="varname"/>
```

Parameters

name (required)

Input parameter. Name of the cart object.

varname (required)

Input and output parameter. As input, name of the variable that holds a boolean value output as a result of the attempted commit. True means that the cart has been committed to Transact.

Description

Only a successfully checked cart can be committed. A commit of an otherwise acceptable cart may fail due to unsuccessful payment authorization or other special custom conditions associated with the CS-Direct Advantage store configuration.

Note

If you call `cart:commit` for a cart that has not previously been checked, then that cart is checked prior to being committed.

The cart object is updated to reflect any errors or notifications retrieved from the Commerce Connector. Choices for fields which have choices also become available.

cart:create

Creates an empty cart object.

Syntax

```
<cart:create
  name="cartname"/>
```

Parameter

name (required)

Input and output parameter. As input, the name of the cart object to create on output.

cart:deleteitem

Deletes an item from a cart.

Syntax

```
<cart:deleteitem
    name="cartname"
    id="itemid"
    [listvarname="listvar"]/>
```

Parameters

name (required)

Input parameter. Name of the shopping cart.

id (required)

Input parameter. Row identifier that specifies the item.

listvarname (optional)

Input and output parameter. As input, a list variable that, if included, holds all the parameters of the item in the deleted row upon output.

Example

```
<%
    String aid = ics.GetVar("delitem");
    if (aid!=null)
    {
%>
        <!-- If yes, delete the item from the shopping cart -->
        <cart:deleteitem name="thiscart" id="delitem"/>
        <!-- Save the cart to the commerce context -->
        <commercecontext:setcurrentcart cart="thiscart"/>
<%
    }
%>
```

See Also

[cart:getparameters](#)

cart:fromstring

Creates a cart object from a text string previously written by [cart:tostring](#).

Syntax

```
<cart:fromstring
  name="objname"
  varname="varname"/>
```

Parameters

name (required)

Input and output parameter. As input, name of the cart object to create on output.

varname (required)

Input parameter. Name of the string previously written by [cart:tostring](#).

cart:getitemdiscounts

Obtains a list of current discounts.

Syntax

```
<cart:getitemdiscounts
  name="cartname"
  id="itemid"
  listvarname="varname"/>
```

Parameters

name (required)

Name of the cart object.

id (required)

Input parameter. Row identifier that specifies the item.

listvarname (required)

Input and output parameter. As input, name of a list to create on output with the current discounts. This list has the following columns:

- bucket – Identifier for the type of discount.
- storeid – Store for which to apply this discount.
- description – Description of the discount.
- value – Currency value of the discount.

cart:getcartdiscounts

Obtains a list of the current discounts.

Syntax

```
<cart:getcartdiscounts
    name="cartname"
    listvarname="varname"/>
```

Parameters

name (required)

Input parameter. Name of the cart object.

listvarname (required)

Input and output parameter. As input, name of a list to create on output with the current discounts. This list has the following columns:

- **bucket** – Identifier for the type of discount.
- **storeid** – Store for which to apply this discount
- **description** – Description of the discount.
- **value** – Currency value of the discount.

cart:getcartdiscounttotal

Obtains a total of all discounts applied to the cart.

Syntax

```
<cart:getcartdiscounttotal
    name="cartname"
    [storeid="storeid"]
    varname="varname"/>
```

Parameters

name (required)

Input parameter. Name of the cart object.

storeid (optional)

Input parameter. ID of the store for which a discount total should be retrieved, if the cart represents more than one store. If unspecified, discount totals for all stores are retrieved.

varname (required)

Input and output parameter. As input, name of a variable that holds the total, which is assigned as a double value on output.

Description

The cart total obtained is discount only and does not include any item, shipping, or tax amounts.

cart:geterrors

Obtains the cart parameter errors reported the last time the cart was checked.

Syntax

```
<cart:geterrors  
    name="cartname"  
    [storeid="storeid"]  
    listvarname="listname"/>
```

Parameters

name (required)

Input parameter. Name of the cart object.

storeid (optional)

Input parameter. ID of the store for which errors should be retrieved, if the cart represents more than one store. If unspecified, errors for all stores are retrieved.

listvarname (required)

Input and output parameter. As input, name of the list to create on output with the error information. This list has three columns:

- field – Name of the parameter with which the error is associated.
- description – Locale-specific error text.
- errorcode – Integer corresponding to the error condition.

cart:getfinaltotal

Obtains the final cart total.

Syntax

```
<cart:getfinaltotal  
    name="cartname"  
    varname="varname"  
    [storeid="storeid"]/>
```

Parameters

name (required)

Input parameter. Name of the cart object.

varname (required)

Input and output parameter. As input, name of a variable that holds the total, which is assigned as a double value on output.

storeid (optional)

Input parameter. ID of the store where the total applies, if the cart represents more than one store. If storeid is not included, returns the item total for all stores.

Description

When called, `cart:getfinaltotal` returns a final total that includes items and discounts, tax, and shipping. Always returns 0 (zero) until the cart has been checked or committed.

cart:getitemdiscounttotal

Obtains a list of the item discount totals.

Syntax

```
<cart:getitemdiscounttotal
    name="cartname"
    varname="varname"/>
```

Parameters

name (required)

Input parameter. Name of the cart object.

varname (required)

Input and output parameter. As input, name of a variable to hold the total of the item discount on output.

cart:getitemerrors

Obtains the errors for an order item.

Syntax

```
<cart:getitemerrors
    name="cartname"
    id="itemid"
    listvarname="varlist"/>
```

Parameters

name (required)

Input parameter. Name of the shopping cart.

id (required)

Input parameter. Row identifier that specifies the item.

listvarname (required)

Input and output parameter. As input, name of the list to create on output with the following columns:

- **field** – Item parameter where the error occurred.
- **description** – Information about the error.
- **errorcode** – Error encountered.

cart:getitemlegalvalues

Obtains an item parameter's legal values.

Syntax

```
<cart:getitemlegalvalues
  name="cartname"
  id="itemid"
  field="fieldid"
  listvarname="varlist"/>
```

Parameters

name (required)

Input parameter. Name of the shopping cart.

id (required)

Input parameter. Row identifier that specifies the item.

field (required)

Input parameter. Name of the item parameter.

listvarname (required)

Input and output parameter. As input, name of the list to create on output with the following columns:

- **name** – Presented to the end user.
- **value** – Returned to set the cart parameter **field**.

Description

This tag obtains an item parameter's legal values. There are no fixed choices for item parameters.

cart:getitemparameter

Obtains a single item parameter.

Syntax

```
<cart:getitemparameter
  name="cartname"
  id="itemid"
  field="fieldid"
  varname="varparam"/>
```

Parameters

name (required)

Input parameter. Name of the shopping cart.

id (required)

Input parameter. Row identifier that specifies the item.

field (required)

Input parameter. Name of the item parameter.

`varname` (required)

Input and output parameter. As input, name of a variable that holds the parameter value returned on output for the specified order item.

Description

This tag retrieves a single item parameter from the shopping cart.

See Also

[Chapter 17, “CS-Direct Advantage Order Item Parameters”](#)

cart:getitemparameters

Obtains all item parameters for the item.

Syntax

```
<cart:getitemparameters
    name="cartname"
    id="itemid"
    listvarname="varlist"/>
```

Parameters

`name` (required)

Input parameter. Name of the shopping cart.

`id` (required)

Input parameter. Row identifier that specifies the item.

`listvarname` (required)

Input and output parameter. As input, name of the list to create on output with the following columns:

- `name` – Name of item parameter.
- `value` – Value of item parameter.

Description

This tag retrieves all item parameters for the item in the shopping cart.

See Also

[Chapter 17, “CS-Direct Advantage Order Item Parameters”](#)

cart:getitems

Obtains all items in the cart.

Syntax

```
<cart:getitems
    name="cartname"
    [storeid="storeid"]
    [list="inputlist"]
    listvarname="varlist"/>
```

Parameters

name (required)

Input parameter. Name of the shopping cart.

storeid (optional)

Input parameter. ID of the store where the items exist, if the cart represents more than one store. If *storeid* is not included, returns all items for all stores.

list (optional)

Input parameter. Describes the minimum input parameters. By default, only the six fixed columns are returned, and any values set for items.

listvarname (required)

Input and output parameter. As input, name of the list to create on output with the following columns: rowid, assettype, assetid, storeid, quantity, price, *parametername*

Description

This tag retrieves all items in the shopping cart.

Example

```
<!-- Build a resultset list that holds all item parameters. -->
<listobject:create name="itemlist" columns="value"/>
<listobject:addrow name="itemlist">
    <listobject:argument name="value" value="name"/>
</listobject:addrow>
<listobject:addrow name="itemlist">
    <listobject:argument name="value" value="uniqueid"/>
</listobject:addrow>
<listobject:tolist name="itemlist" value="itemlist"/>
<!-- Get any existing items from the cart. -->
<cart:getitems name="thiscart" list="itemlist"
listvarname="cartlist"/>
```

See Also

[Chapter 17, “CS-Direct Advantage Order Item Parameters”](#)

cart:getitemtotal

Obtains the item total of the cart.

Syntax

```
<cart:getitemtotal
    name="cartname"
    varname="varname"
    [storeid="storeid"]/>
```

Parameters

name (required)

Input parameter. Name of the shopping cart.

varname (required)

Input and output parameter. As input, name of a variable that holds the total, which is assigned as a double value on output.

storeid (optional)

Input parameter. ID of the store where the total applies, if the cart represents more than one store. If storeid is not included, returns the item total for all stores.

Description

When called, `cart:getitemtotal` returns the total of items only; the total does not include discounts, tax, or shipping.

Example

```
<!-- Get the cart item total -->
<cart:getitemtotal name="cart" varname="carttotal"/>
<!-- Convert the total to currency. -->
<currency:getcurrency name="curval" value="carttotal"
    varname="curout"/>
```

cart:getlegalvalues

Obtains the legal values for a parameter from the cart.

Syntax

```
<cart:getlegalvalues
    name="cartname"
    field="fieldname"
    [storeid="storeid"]
    listvarname="varname"/>
```

Parameters

name (required)

Input parameter. Name of the shopping cart.

field (required)

Input parameter. Name of the cart parameter for which to obtain legal values.

storeid (optional)

Input parameter. ID of the store for which legal values should be retrieved, if the cart represents more than one store. Use it to retrieve shipping-related legal values. May also be used to retrieve payment-related legal values; otherwise it is ignored.

listvarname (required)

Input and output parameter. As input, name of the list to create on output with the following columns:

- name – Presented to the end user.
- value – Returned to set the cart parameter field.

Description

The legal values returned by `cart:getlegalvalues` may be supplied by Transact, and may be incomplete until the cart is checked. No list is created if the cart has not yet been checked, or if there are no fixed possibilities for the parameter.

Example

```
<!-- Get the current cart. -->
<commercecontext:getcurrentcart varname="thiscart"/>
<!-- Check the cart to enumerate payment methods. -->
<cart:check name="thiscart" varname="cartstatus"/>
<!-- Now get the legal values for the payment types. -->
<cart:getlegalvalues name="thiscart" field="paymentInfo-brandID"
  listvarname="paylist"/>
```

See Also

[Chapter 16, “CS-Direct Advantage Cart Parameters”](#)

cart:getparameter

Obtains the current value for a specified generic cart-level parameter.

Syntax

```
<cart:getparameter
  name="cartname"
  field="fieldname"
  [storeid="storeid"]
  varname="varname"/>
```

Parameters**name (required)**

Input parameter. Name of the cart object.

field (required)

Input parameter. Name of the cart parameter for which to the current value.

storeid (optional)

Input parameter. ID of the store for which a parameter should be retrieved, if the cart represents more than one store. Use it to retrieve shipping-related values that are store-dependent; otherwise it is ignored.

varname (required)

Input and output parameter. As input, name of the variable to hold the value assigned on output.

Description

The variable created to hold the value is cleared to null if the field does not exist in the cart.

See Also

[Chapter 16, “CS-Direct Advantage Cart Parameters”](#)

cart:getparameters

Obtains the current parameter selections for the cart.

Syntax

```
<cart:getparameters
    name="cartname"
    listvarname="listname"/>
```

Parameters

name (required)

Input parameter. Name of the cart object.

listvarname (required)

Input and output parameter. As input, name of the list to create on output with name and value columns.

Description

This tag obtains the parameter selections for the shopping cart object.

See Also

[Chapter 16, “CS-Direct Advantage Cart Parameters”](#)

cart:getpreliminarytotal

Obtains the preliminary cart total.

Syntax

```
<cart:getpreliminarytotal  
  name="cartname"  
  varname="varname"  
  [storeid="storeid"]/>
```

Parameters

name (required)

Input parameter. Name of the cart object.

varname (required)

Input and output parameter. As input, name of a variable that holds the preliminary total, which is assigned as a double value on output.

storeid (optional)

Input parameter. ID of the store where the total applies, if the cart represents more than one store. If storeid is not included, returns the item total for all stores.

Description

The total returned by `cart:getpreliminarytotal` includes all items and discounts, but not tax and shipping.

cart:getshippingtotal

Obtains the shipping total.

Syntax

```
<cart:getshippingtotal  
  name="cartname"  
  varname="varname"  
  [storeid="storeid"]/>
```

Parameters

name (required)

Input parameter. Name of the cart object.

varname (required)

Input and output parameter. As input, name of a variable that holds the shipping total, which is assigned as a double value on output.

storeid (optional)

Input parameter. ID of the store where the shipping total applies, if the cart represents more than one store. If storeid is not included, returns the shipping total for all stores.

Description

The total returned by `cart:getshippingtotal` includes shipping, but not items and discounts and tax. Always returns 0 until the cart has been either checked or committed.

cart:getstores

Obtains a list of stores for which items in the cart are assigned.

Syntax

```
<cart:getstores
    name="cartname"
    listvarname="listname"/>
```

Parameters

name (required)

Input parameter. Name of the cart object.

listvarname (required)

Input and output parameter. As input, name of the list to create on output, containing store identifiers in the single column value.

Description

When called, `cart:getstores` returns transact store identifiers for each store for which items exist.

cart:gettaxtotal

Obtains the tax total for the cart.

Syntax

```
<cart:gettaxtotal
    name="cartname"
    varname="varname"
    [storeid="storeid"]/>
```

Parameters

name (required)

Input parameter. Name of the cart object.

varname (required)

Input and output parameter. As input, name of a variable that holds the tax total, which is assigned as a double value on output.

storeid (optional)

Input parameter. ID of the store where the tax total applies, if the cart represents more than one store. If `storeid` is not included, returns the tax total for all stores.

Description

When called, `cart:gettaxtotal` returns a total that includes taxes, but not items, discounts, and shipping. Always returns zero until the cart has been checked or committed.

cart:gettransactionid

Obtains the Transact transaction identifier from the Commerce Connector for this cart.

Syntax

```
<cart:gettransactionid
  name="cartname"
  varname="varname"/>
[storeid="storeid"]/>
```

Parameters

name (required)

Input parameter. Name of the cart object.

varname (required)

Input and output parameter. As input, name of a variable that holds the transaction identifier assigned on output.

storeid (optional)

Input parameter. ID of the store to which the transaction applies, if the cart represents more than one store. Ignored otherwise.

Description

The returned value is valid only if the cart has been committed.

cart:gettransactionids

Obtains the transaction identifiers from the Commerce Connector for this cart.

Syntax

```
<cart:gettransactionids
  name="cartname"
  listvarname="varname"/>
```

Parameters

name (required)

Input parameter. Name of the cart object.

listvarname (required)

Input and output parameter. As input, name of the list to create on output, containing transaction identifiers in the single column value.

Description

This tag is similar to `cart:gettransactionid`, but applies to multistore shopping carts. The returned values are valid only if the cart has been committed.

cart:merge

Merges an older cart into the current cart object.

Syntax

```
<cart:merge
    name="currentcart"
    cart="oldcart"/>
```

Parameters

name (required)
Input parameter. Name of the current cart object.

cart (required)
Input parameter. Name of the older cart object.

cart:setitemparameters

Sets order item parameters.

Syntax

```
<cart:setitemparameters
    name="cartname"
    id="itemid"
    list="listname"/>
```

Parameters

name (required)
Input parameter. Name of the cart object.

id (required)
Input parameter. Row identifier that specifies the item.

list (required)
Input parameter. List of name/value pairs to set on output for the specified order item.

Description

This tag sets order item parameters for the shopping cart.

See Also

[Chapter 17, “CS-Direct Advantage Order Item Parameters”](#)

cart:setitemquantity

Sets an item's quantity and price information.

Syntax

```
<cart:setitemquantity
  name="cartname"
  id="itemid"
  quantity="itemquantity"
  price="itemprice"/>
```

Parameters

name (required)

Input parameter. Name of the cart object.

id (required)

Input parameter. Row identifier that specifies the item.

quantity (required)

Input parameter. Number of the specified item to be set on output.

price (required)

Input parameter. Price of the specified item to be set on output.

cart:setparameter

Sets a single shopping cart parameter.

Syntax

```
<cart:setparameter
  name="cartname"
  field="field"
  [value="value"]
  [storeid="storeid"]/>
```

Parameters

name (required)

Input parameter. Name of the cart object.

field (required)

Input parameter. Name of the cart parameter to set.

value (optional)

Input parameter. Value of the cart parameter to be set on output. If not included, the field value is undefined.

storeid (optional)

Input parameter. Store for which the parameter is to be set on output. It is ignored otherwise.

Description

The parameter set by this tag is a name/value pair whose interpretation is determined by the Commerce Connector. See [Chapter 16, “CS-Direct Advantage Cart Parameters”](#) for cart parameter names and values.

cart:setparameters

Sets the specified shopping cart parameters.

Syntax

```
<cart:setparameters
    name="cartname"
    list="listname"/>
```

Parameters

name (required)

Input parameter. Name of the cart object.

list (required)

Input parameter. Name of the list that holds the requested parameter values. The list has name and value columns.

Description

The parameters set by this tag are name/value pairs that are interpreted by the Commerce Connector. See [Chapter 16, “CS-Direct Advantage Cart Parameters”](#) for cart parameter names and values.

cart:tostring

Writes the cart object into a text string.

Syntax

```
<cart:tostring
    name="objname"
    varname="varname"/>
```

Parameters

name (required)

Input parameter. Name of the cart object.

varname (required)

Input and output parameter. As input, name of the string to which to write the cart object on output. The string is long, that is, measured in kilobytes.

Cart Set Tags

You use cart set tags to store, retrieve, delete, and list stored cart objects for a registered buyer. Cart set tags cannot be used for unregistered or walk-in buyers.

The visitor or buyer for whom the cart set is maintained is identified by a user ID, a string normally obtained from [vdm:getcommerceid](#) or [csuser:lookup](#).

Note

Each JSP file containing JSP cart set tags must have the following directive at the top:

```
<%@ taglib prefix="cartset" uri="futuretense_cs/cartset.tld"%>
```

The following section describes each cart set tag, beginning with [cartset:addcart](#).

cartset:addcart

Adds a new cart to the cart set.

Syntax

```
<cartset:addcart  
    name="cartset"  
    cartid="cartid"  
    cart="newcart"/>
```

Parameters

name (required)

Input parameter. Name of the cart set to interrogate. A previous call to [cartset:getset](#) is required to get this parameter.

cartid (required)

Input parameter. Identifier to use for the cart. If a cart with the same identifier already exists in the cart set, it is replaced on output.

cart (required)

Input parameter. Cart object to add to the cart set on output.

cartset:cartids

Returns a list of identifiers for carts within the cart set.

Syntax

```
<cartset:cartids
    name="cartset"
    listvarname="listname"/>
```

Parameters

name (required)

Input parameter. Name of the cart set to interrogate. A previous call to [cartset:getset](#) is required to get this parameter.

listvarname (required)

Input and output parameter. List variable that holds the identifiers for the carts in the cart set. The list has a single column, value.

cartset:deletecart

Deletes a cart from the cart set.

Syntax

```
<cartset:deletecart
    name="cartset"
    cartid="cartid"/>
```

Parameters

name (required)

Input parameter. Name of the cart set to interrogate. A previous call to [cartset:getset](#) is required to get this parameter.

cartid (required)

Input parameter. Identifier of the cart to be deleted. The specified cart is removed from the cart set on output.

cartset:getcart

Retrieves a stored cart from the cart set.

Syntax

```
<cartset:getcart
    name="cartset"
    cartid="cartid"
    cart="cartvar"/>
```

Parameters

name (required)

Input parameter. Name of the cart set to interrogate. A previous call to [cartset:getset](#) is required to get this parameter.

cartid (required)

Input parameter. Identifier of the cart to be retrieved.

cart (required)

Input and output parameter. Name of the variable to fill with the retrieved cart object on output.

cartset:getset

Retrieves the current visitor's cart set, or else creates a new empty cart set.

Syntax

```
<cartset:getset
    name="cartset"
    userid="userid"/>
```

Parameters

name (required)

Input and output parameter. Variable name in which to store the retrieved or created cart set. This parameter is required for calls to all other cart set tags.

userid (required)

Input parameter. Identifier for the user.

Commerce Context Tags

During a visitor session, CS-Direct Advantage creates and saves a **commerce context** for the visitor. Current shopping cart is maintained by the commerce context.

You use the commerce context JSP tags to save cart objects to and retrieve objects from the commerce context. Shopping cart objects, when present, are persistent throughout the visitor session; that is, they are stored in the Content Server database and then flushed when the session ends.

The commerce context maintains the following information about CS-Engage assets:

- List of segments for which the visitor qualifies
- List of promotions for which the visitor qualifies
- Time object that is used for calculating time-based rules
- Utility object that gives you access to product attributes

There are additional tags available with CS-Engage for displaying recommendations and promotions on your application pages. See [Chapter 12, “CS-Engage JSP Tags,”](#) in this guide for more information, as well as the *dCS Developer's Guide*.

Note

Each JSP file containing JSP commerce context tags must have the following directive at the top:

```
<%@ taglib prefix="commercecontext" uri="futuretense_cs/
commercecontext.tld" %>
```

The following section describes each commerce context tag, beginning with [commercecontext:getabandonedcartsessions](#).

commercecontext:getabandonedcartsessions

Gets a list of sessions that have abandoned carts.

Syntax

```
<commercecontext:getabandonedcartsessions
    [startdate="date"]
    enddate="date"
    listvarname="listvar"/>
```

Parameters

startdate (optional)

Input parameter. Starting date, as a epoch-based timestamp. The default value is 0 if this parameter is not provided.

enddate (required)

Input parameter. Ending date, as a epoch-based timestamp.

listvarname (required)

Input and output parameter. As input, name of a list that holds the sessions that have abandoned carts. The list is created on output with two columns: odate (which is the timestamp describing the last time the cart was updated for that session) and osession.

Description

Abandoned carts are carts that were last modified for the session between a specified date range.

commercecontext:getcurrentcart

Retrieves and names a copy of the current cart from the visitor context.

Syntax

```
<commercecontext:getcurrentcart  
  varname="cartname"/>
```

Parameters

varname (required)

Input or output parameter. Name of the cart object to retrieve or create.

Description

Retrieves a locally available copy of the current cart.

You can modify settings, data, and functions in the copy and then check it back into the visitor context with [commercecontext:setcurrentcart](#).

Example

```
<!-- Get the current cart. -->  
<commercecontext:getcurrentcart varname="thiscart"/>  
<!-- Check the cart to enumerate payment methods. -->  
<cart:check name="thiscart" varname="cartstatus"/>  
<!-- Now get the legal values for the payment types. -->  
<cart:getlegalvalues name="thiscart" field="paymentInfo-brandID"  
  listvarname="paylist"/>
```

commercecontext:getsessioncart

Obtains a cart given a session identifier (visitor id).

Syntax

```
<commercecontext:getsessioncart  
    session="sessionid"  
    varname="cartname"/>
```

Parameters

session (required)

Input parameter. Session identifier.

varname (required)

Input parameter retrieves the specified cart object and specifies a name for it.

commercecontext:logout

Discards the current visitor ID and creates a new one.

Syntax

```
<commercecontext:logout/>
```

Parameters

None.

Description

When visitors access a site implemented with CS-Engage, they are assigned a visitor ID. This tag clears the current visitor ID and assigns a new one. Use this tag to create a logout button so a new visitor can interact with the site without having to end the session.

If the current visitor is logged into Transact or any other external database through Commerce Connector, this tag clears this visitor's commerce ID or access ID as well, which logs the visitor out from Commerce Connector. The visitor must log in again to obtain the new visitor ID.

commercecontext:saveall

Saves a copy of everything in the commerce context to the Content Server database for the duration of the session; that is, to persistent storage.

Syntax

```
<commercecontext:saveall/>
```

Parameters

None.

Description

Because the visitor objects are persistent, the only use for this tag is if you want to store other custom context objects (that must be explicitly saved) that you created during the session.

commercecontext:setcurrentcart

Checks the working copy of the current cart back into the visitor context, updating the current cart that is stored there.

Syntax

```
<commercecontext:setcurrentcart
    cart="cartname"/>
```

Parameters

cart (required)

Input parameter. Name of the working copy of the cart that updates the current cart object.

Description

For more information about working with carts, see the *dCS Developer's Guide*.

Example

```
<!-- Is there an item to delete? -->
<%
    String aid = ics.GetVar("delitem");
    if (aid!=null)
    {
%>
        <!-- If yes, delete the item from the shopping cart -->
        <cart:deleteitem name="thiscart" id="delitem"/>
        <!-- Save the cart to the commerce context -->
        <commercecontext:setcurrentcart cart="thiscart"/>
    }
%>
```

CSMAC Tags

You use CSMAC tags to generate Message Authentication Codes (MACs) that can be interpreted by Transact, and to produce encrypted cookie values to identify individual visitors to Transact via the Commerce Connector.

Note that `csmac:string` and `csmac:payload` produce different varname values from the same input value:

- `csmac:string` calculates MAC based on value
- `csmac:payload` calculates MAC based on:
'value + "&kid=" + storeID + "." + currentKeyNumStr + "&ss=env"'

Note

Each JSP file containing JSP CSMAC tags must have the following directive at the top:

```
<%@ taglib prefix="csmac" uri="futuretense_cs/csmac.tld" %>
```

The following section describes each CSMAC tag, beginning with `csmac:cookievalue`.

csmac:cookievalue

Produces a cookie value that the web designer can insert into the visitor's browser to enable Transact to recognize the user without requiring additional authentication.

Syntax

```
<csmac:cookievalue
    userid="userid"
    storeid="storeid"
    [seconds="duration"]
    varname="varcookie"/>
```

Parameters

userid (required)

Input parameter. Identifier of the user as known to Transact. You can get this value by calling `vdm:getcommerceid`.

storeid (required)

Input parameter. Identifier of the Transact store whose secret keys are used to generate the MAC.

seconds (optional)

Input parameter. Duration in seconds (from now) for which Transact honors the cookie. If omitted, the default duration is 600 (10 minutes).

varname (required)

Input and output parameter. Name of a variable that holds the cookie string output with the following form:

```
"MAC:principal_id=userid&expiration=<now+durationSeconds>&kid=storeid.
<keyID>&ss=env"
```

Description

Successful usage of such a cookie is limited to configurations where Transact and CS-Direct Advantage reside in the same DNS domain. Typically, the value returned is used with the Content Server [SETCOOKIE](#) tag.

csmac:payload

Generates a MAC query string value for a string resembling a name-value pair URL query string, with a specific store's current key.

Syntax

```
<csmac:payload
  value="stringval"
  storeid="storeid"
  varname="payloadvar"/>
```

Parameters

value (required)

Input parameter. String from which the MAC should be calculated. It should be in the name-value pair format: "name1=val1&name2=val2" This is used in URL query strings.

Note

Use the character code `&` as input instead of the ampersand character `&`.

storeid (required)

Input parameter. Identifier of the Transact store whose secret keys are used to generate the MAC.

varname (required)

Input and output parameter. Name of a variable to write to with the payload string of the form "MAC:value&kid=storeid.keyID&ss=env". For example, with a value of name1=val1 and storeID of 400003, the following is returned (depending on current key values):

```
"Ha3857e216c5523843495550f4834908c:name1=val1&kid=400003.4&ss=env"
```

csmac:string

Generates a MAC string for any string with a specific store's current key.

Syntax

```
<csmac:string
    value="stringval"
    storeid="storeid"
    varname="stringvar" />
```

Parameters

value (required)

Input parameter. String from which the MAC should be calculated.

storeid (required)

Input parameter. Identifier of the Transact store whose secret key is used to generate the MAC.

varname (required)

Input and output parameter. Name of a variable to write to with the MAC string of the form `MAC:kid=storeID.keyID`. For example, with a value of `name1=val1` and storeID of 400003, the following could be returned (depending on current key values):

`Ha3857e21f48349086c5523843495550c:kid=400003.4`

csmac:transactserver

Returns a string with the form *protocol://hostname:port* that is the URL prefix of the configured Transact.

Syntax

```
<csmac:string
    [protocol="http|https"]
    varname="stringvar" />
```

Parameters

protocol (optional)

Input parameter. Either `http` or `https`. If `protocol` is omitted, the default protocol for contacting Transact is taken from the `transact.ini` file.

varname (required)

Input and output parameter. As input, name of a variable to write with the URL prefix string on output.

Description

The returned string contains the protocol, hostname, and possibly port number, but does not contain any pathname or query string portions of the URL. If a standard default port is configured for either `http` (80) or `https` (443), this is omitted from the returned string. For example, depending on values in the `transact.ini` file, the following URL string might be returned:

`https://release.openmarket.com:8081`

CSUSER Tags

You use CSUSER tags to produce, obtain, and maintain information about Transact registered buyers. The CSUSER tags require the Commerce Connector for communication with Transact.

You set and retrieve individual buyer parameters in a manner similar to that used for shopping cart parameters. See [Chapter 18, “CS-Direct Advantage Buyer Parameters”](#) for buyer parameter names and values. See [Chapter 16, “CS-Direct Advantage Cart Parameters”](#) for cart parameter names and values.

Note

Each JSP file containing JSP csuser tags must have the following directive at the top:

```
<%@ taglib prefix="csuser" uri="futuretense_cs/csuser.tld"
%>
```

The following section describes each CSUSER tag, beginning with [csuser:authenticate](#).

csuser:authenticate

Looks up and retrieves a user information from the Transact database.

Syntax

```
<csuser:authenticate
    name="csuser"
    varname="booleanvar"/>
```

Parameters

name (required)

Input parameter. Name of the csuser object.

varname (required)

Input parameter. A variable that holds a string representing boolean status:

- true – the buyer has been retrieved from Transact.
- false – the buyer could not be authenticated or, an error has occurred. Call `csuser:geterrors` to determine the problem.

Description

Typically, a form is presented to the site visitor asking for an `accessName` and `password`. Be sure to use a secure URL (https) with this form to guarantee the secrecy of the visitor's password.

Both the `accessName` parameter and the `password` parameter must be set within the csuser object to identify the user. This differs from `lookup` insofar as this tag provides a means of confirming that the buyer knows his password.

The values supplied by the visitor are set in the csuser object with a call to either `csuser:setparameter` or `csuser:setparameters`. If a call to

`csuser:authenticate` returns `true`, you then use the `userid` parameter in the `csuser` object to call `vdm:setcommerceid`.

csuser:clear

Clears the parameters and data in a `csuser` object.

Syntax

```
<csuser:clear  
  name="csuser"/>
```

Parameters

`name` (required)

Input parameter. Name of the `csuser` object to clear.

csuser:create

Creates an empty `csuser` object.

Syntax

```
<csuser:create  
  name="csuser"/>
```

Parameters

`name` (required)

Input parameter. Name of the `csuser` object to create.

csuser:createpmtacct

Creates a new payment account in Transact for the `csuser` object.

Syntax

```
<csuser:createpmtacct  
  name="csuser"  
  varname="booleanvar"  
  list="listname"/>
```

Parameters

`name` (required)

Input parameter. Name of the `csuser` object.

`varname` (required)

Input parameter. A variable that holds a string representing boolean status:

- `true` – the payment account has been registered with Transact.

- `false` – an error has occurred. Call `csuser:geterrors` to determine the problem.

The variable is set to indicate whether the payment account was created on Transact.

`list` (required)

Input parameter. List that holds the payment method parameters. The list has two columns: name and value, where each parameter name listed below is one half of a name/value pair, and each name/value pair is a row in the list.

- `name` – (optional) A user-supplied name for the account
- `brandID` – Credit card/switch brand ID (GUID) from `getAvailablePaymentBrands()`.
- `ccn` – Credit/switch card number, or equivalent
- `expirationDate` – in 'mm/yy' format.
- `billToAddress` – (optional) billing address for account. The default is the buyer's address. If any value is set, then all address fields must be set. The address must be specified in nine rows using the following standard fields:

```
billToAddress-name1
billToAddress-name2
billToAddress-address1
billToAddress-address2
billToAddress-city
billToAddress-state
billToAddress-country
billToAddress-postalCode
billToAddress-telephone
```

- `issueNumber` – (optional) Switch Card Issue Number
- `extraInstrumentInfo` – (optional) Japanese brands only
- `poNumber` – (optional) Purchase order number for PO account only
- `poComments` – (optional) Purchase order comment for PO account only

`csuser:deletepmtacct`

Deletes a payment account on Transact for the user.

Syntax

```
<csuser:deletepmtacct
  name="csuser"
  id="accountid"
  varname="listvar"/>
```

Parameters

`name` (required)

Input parameter. Name of the `csuser` object.

`id` (required)

Input parameter. The accountID within Transact of the payment account to be deleted.

varname (required)

Input parameter. Variable that holds a string representing boolean status:

- true – the payment account has been deleted from Transact.
- false – an error has occurred. Call `csuser:geterrors` to determine the problem.

csuser:fromstring

Creates a `csuser` object from a text string previously written by `csuser:tostring`.

Syntax

```
<csuser:fromstring  
    name="objname"  
    varname="varname"/>
```

Parameters

name (required)

Input and output parameter. As input, name of the `csuser` object to create on output.

varname (required)

Input parameter. Name of the string previously written by `csuser:tostring`.

csuser:getallpmtaccts

Obtains the current payment parameter selections for the `csuser` object.

Syntax

```
<csuser:getallpmtaccts  
    name="csuser"  
    listvarname="listvar"/>
```

Parameters

name (required)

Input parameter. Name of the `csuser` object.

listvarname (required)

Input and output parameter. As input, name of a list to create on output with the following columns:

- name
- accountID
- billToAddress-name
- billToAddress-address1
- expirationDate
- brandID

csuser:geterrors

Obtains the errors reported the last time any of the following tags were called for the specified csuser object:

- csuser:register
- csuser:modify
- csuser:lookup
- csuser:authenticate
- csuser:createpmtacct
- csuser:modifypmtacct
- csuser:deletepmtacct

Syntax

```
<csuser:geterrors  
    name="csuser"  
    listvarname="listname"/>
```

Parameters

name (required)

Input parameter. Name of the csuser object.

listvarname (required)

Input and output parameter. As input, name of the list to create on output that holds the error information. The list has three columns:

- field – the name of the parameter with which the error is associated
- description – locale-specific error text
- errorcode – an integer corresponding to the error condition

csuser:getlegalpmtbrands

Obtains the current parameter selections for the csuser object.

Syntax

```
<csuser:getlegalpmtbrands  
    name="csuser"  
    listvarname="listvar"/>
```

Parameters

name (required)

Input parameter. Name of the csuser object.

listvarname (required)

Input and output parameter. As input, name of the list to create on output that holds the error information. The list has two columns:

- name – is presented to the end user

- value – returned to set the parameter paymentInfo-brandID

csuser:getparameter

Obtain the current value for a specified generic user parameter.

Syntax

```
<csuser:getparameter  
  name="csuser"  
  field="paramname"  
  varname="varname"/>
```

Parameters

name (required)

Input parameter. Name of the csuser object.

field (required)

Input parameter. Name of the parameter.

varname (required)

Input and output parameter. As input, name of the variable to create on output with the result. The variable is cleared to null if the field does not exist.

Description

See [Chapter 18, “CS-Direct Advantage Buyer Parameters”](#) for parameter names and values.

csuser:getparameters

Verifies the user exists in Transact.

Syntax

```
<csuser:getparameters  
  name="csuser"  
  listvarname="listname"/>
```

Parameters

name (required)

Input parameter. Name of the csuser object.

listvarname (required)

Input and output parameter. As input, name of the list to create on output. The list has two columns: name and value.

Description

See [Chapter 18, “CS-Direct Advantage Buyer Parameters”](#) for parameter names and values.

csuser:lookup

Looks up and retrieves a buyer from the Commerce Connector.

Syntax

```
<csuser:lookup
    name="csuser"
    varname="booleanvar"/>
```

Parameters

name (required)

Input parameter. Name of the csuser object.

varname (required)

Input parameter. Variable that holds a string representing boolean status:

- true – the buyer has been retrieved with Transact.
- false – an error has occurred. Call `csuser:geterrors` to determine the problem.

Description

Either the csuser object's `accessName` or the `userID` parameter must be set to identify the buyer. If both are set, `userID` is used. The csuser object is updated with the corresponding data from Transact. After lookup, call `vdm:setcommerceid` with the `userID` parameter as input.

See Also

[csuser:authenticate](#)

csuser:modify

Modifies buyer data on Transact.

Syntax

```
<csuser:modify
    name="csuser"
    varname="booleanvar"/>
```

Parameters

name (required)

Input parameter. Name of the csuser object.

varname (required)

Input parameter. Variable that holds a string representing boolean status:

- true – the buyer has been modified with Transact.
- false – an error has occurred. Call `csuser:geterrors` to determine the problem.

Description

The user ID parameter must be set for the csuser object. Any other buyer parameters set in the csuser object replace the corresponding parameters in the buyer data record on Transact, with the following exceptions:

- The accessName parameter cannot be changed.
- If set, the password parameter must differ from the buyer's current password on Transact.
- If any address field is modified, then all address fields must be modified. The address must be specified in nine rows using the following standard fields:

billToAddress-name1	billToAddress-name2
billToAddress-address1	billToAddress-address2
billToAddress-city	billToAddress-state
billToAddress-country	billToAddress-postalCode
billToAddress-telephone	

csuser:modifypmtacct

Changes payment account data on Transact for the specified buyer.

Syntax

```
<csuser:modifypmtacct
  name="csuser"
  list="listname"
  varname="booleanvar"/>
```

Parameters

name (required)

Input parameter. Name of the csuser object.

list (required)

Input parameter. List containing the payment method input parameters. The list has two columns: name and value, where each parameter name listed below is one half of a name/value pair, and each name/value pair is a row in the list.

- **name** – (optional) A user-supplied name for the account
- **accountID** – Credit card/switch brand ID (GUID) from getAvailablePaymentBrands().
- **ccn** – Credit/switch card number, or equivalent
- **expirationDate** – in 'mm/yy' format.
- **billToAddress** – (optional) billing address for account. The default is the buyer's address. If any value is set, then all address fields must be set. The address must be specified in nine rows using the following standard fields:

billToAddress-name1
billToAddress-name2
billToAddress-address1
billToAddress-address2
billToAddress-city
billToAddress-state

billToAddress-country
 billToAddress-postalCode
 billToAddress-telephone

- issueNumber – (optional) Switch Card Issue Number.
- extraInstrumentInfo – (optional) Japanese brands only.
- poNumber – (optional) Purchase order number for PO account only.
- poComments – (optional) Purchase order comment for PO account only.

Note

The ccn and accountID parameters must not be changed.

To change the ccn value, you must delete the old payment account (with `csuser:deletepmtacct`) and create a new one (with `csuser:createpmtacct`).

varname (required)

Input parameter. Variable that holds a string representing boolean status:

- true – the buyer has been modified with Transact.
- false – an error has occurred. Call `csuser:geterrors` to determine the problem.

csuser:register

Registers a new buyer as a csuser object with Transact.

Syntax

```
<csuser:register
  name="csuser"
  varname="booleanvar"/>
```

Parameters

name (required)

Input parameter. Name of the csuser object.

varname (required)

Input parameter. Variable that holds a string representing boolean status:

- true – the buyer has been registered with Transact.
- false – an error has occurred. Call `csuser:geterrors` to determine the problem.

Description

As a result of a call to `csuser:register`, the csuser object is updated with a userID assigned by Transact. After registering, call `vdm:setcommerceid` with the userID parameter as input.

See Also

[csuser:authenticate](#)

csuser:setparameter

Sets a single buyer parameter.

Syntax

```
<csuser:setparameter
  name="csuser"
  field="paramname"
  value="paramvalue"/>
```

Parameters

name (required)

Input parameter. Name of the csuser object.

field (required)

Input parameter. Name of the buyer parameter to set.

value (optional)

Input parameter. Value of the buyer parameter. If not included, field becomes unset.

Description

The name/value pair is subject to Commerce Connector interpretation. See [Chapter 18, “CS-Direct Advantage Buyer Parameters”](#) for parameter names and values.

csuser:setparameters

Sets the specified buyer parameters.

Syntax

```
<csuser:setparameters
  name="csuser"
  list="paramlist"/>
```

Parameters

name (required)

Input parameter. Name of the csuser object.

list (required)

Input parameter. Name of the parameter list. This list has two columns: name and value.

Description

The name/value pairs are subject to Commerce Connector interpretation. See [Chapter 18, “CS-Direct Advantage Buyer Parameters”](#) for parameter names and values.

csuser:toString

Writes the csuser object into a text string.

Syntax

```
<csuser:toString  
    name="objname"  
    varname="varname" />
```

Parameters

name (required)

Input parameter. Name of the csuser object.

varname (required)

Input and output parameter. As input, name of the string to which to write the csuser object on output. The string is long, that is, measured in kilobytes.

Currency Tags

Currency tags perform conversions between floating point values and currency strings that can be displayed. Currency tags also perform formatting and rounding operations on currency strings.

Note

Each JSP file containing JSP currency tags must have the following directive at the top:

```
<%@ taglib prefix="currency" uri="futuretense_cs/
currency.tld" %>
```

The following section describes each currency tag, beginning with [currency:create](#).

currency:create

Creates a currency object for a specific currency ISO code.

Syntax

```
<currency:create
    name="curname"
    [isocurrency="isosymbol"]
    [separator="dot|comma|default"]/>
```

Parameters

name (required)

Input and output parameter. As input, name of the currency object to create on output.

isocurrency (optional)

Input parameter. The 3-character ISO currency symbol. If not specified, the default currency symbol (USD) is used.

separator (optional)

Input parameter. Specifies the currency separator character. It can have one of the following values:

- dot
- comma
- default

Example

```
<!-- Get the item price, which is stored as (price * quantity);
and where cartlist_price and cartlist_quantity are the two String
Variables that have the Values retrieved appropriately-->
<calculator:go value="cartlist_price cartlist_quantity /"
    varname="calcout"/>
<!-- Convert the above item price to currency. -->
<currency:create name="curval"/>
<currency:getcurrency name="curval" value="calcout"
    varname="curout"/>
```

currency:getcurrency

Converts a double to a displayable currency string.

Syntax

```
<currency:getcurrency
    name="curname"
    value="fpvalue"
    varname="varname" />
```

Parameters

name (required)

Input parameter. Name of the currency object.

value (required)

Input parameter. Floating-point monetary value.

varname (required)

Input and output parameter. As input, the name of the variable that holds the displayable monetary string assigned on output.

Example

```
<!-- Get the item price, which is stored as (price * quantity);
and where cartlist_price and cartlist_quantity are the two String
Variables that have the Values retrieved appropriately-->
<calculator:go value="cartlist_price cartlist_quantity /"
    varname="calcout"/>
<!-- Convert the above item price to currency. -->
<currency:create name="curval"/>
<currency:getcurrency name="curval" value="calcout"
    varname="curout"/>
```

currency:getisocode

Returns the ISO code for the currency instance.

Syntax

```
<currency:getisocode  
    name="curname"  
    varname="varname" />
```

Parameters

name (required)

Input parameter. Name of the currency object.

varname (required)

Input and output parameter. As input, the name of the variable that holds the ISO code value assigned on output.

Example

```
<currency:getisocode name="usCurrenct" varname="usISOCODE"/>
```

currency:getsymbol

Returns the symbol used to represent currencies of this type.

Syntax

```
<currency:getsymbol  
    name="curname"  
    varname="varname" />
```

Parameters

name (required)

Input parameter. Name of the currency object.

varname (required)

Input and output parameter. As input, name of the currency object that holds currency symbol assigned on output.

currency:readcurrency

Converts a number previously rendered as a monetary string back to a decimal value.

Syntax

```
<currency:readcurrency  
    name="curname"  
    value="string"  
    varname="varname" />
```

Parameters

name (required)

Input parameter. Name of the currency object.

value (required)

Input parameter. Display string.

varname (required)

Input and output parameter. As input, name of the variable that holds floating point value assigned on output.

Example

```
<currency:getsymbol name="usCurrency" varname="usCurrencySymbol"/>
```

currency:round

Rounds a currency value to the nearest legal absolute currency value.

Syntax

```
<currency:round  
    name="curname"  
    value="string"  
    varname="varname" />
```

Parameters

name (required)

Input parameter. Name of the currency object.

value (required)

Input parameter. Value to be rounded.

varname (required)

Input and output parameter. As input, name of the variable that holds the rounded floating point value assigned on output.

Example

```
<currency:round name="usCurrency" value="10.089"  
varname="usCurrencyOutput"/>
```

currency:rounddown

Rounds a currency value down to the next lower legal absolute currency value.

Syntax

```
<currency:rounddown
  name="curname"
  value="string"
  varname="varname" />
```

Parameters

name (required)

Input parameter. Name of the currency object.

value (required)

Input parameter. Value to be rounded down.

varname (required)

Input and output parameter. As input, name of the variable that holds the rounded down floating point value assigned on output.

Example

```
<currency:rounddown name="usCurrency1" value="10.085"
  varname="usCurrencyRounddown" />
```

currency:roundup

Rounds a currency value up to the next higher legal absolute currency value.

Syntax

```
<currency:roundup
  name="curname"
  value="string"
  varname="varname" />
```

Description

Here are some examples of how this tag rounds up:

- 10.011 is rounded to 10.02
- -10.011 is rounded to -10.02.

Parameters

name (required)

Input parameter. Name of the currency object.

value (required)

Input parameter. Value to be rounded up.

varname (required)

Input and output parameter. As input, name of the variable that holds the rounded up floating point value assigned on output.

Example

```
<currency:roundup name="usCurrency" value="10.085"  
varname="usCurrencyRoundup"/>
```

Date Converter Tags

Date converter tags convert dates with year, month, day, and optional hour, minute, and am/pm fields into epoch format long integers representing milliseconds since Jan 1, 1970, 0:00 GMT. Date tags also convert long integers into dates.

Note

Each JSP file containing JSP date tags must have the following directive at the top:

```
<%@ taglib prefix="date" uri="futuretense_cs/date.tld" %>
```

The following section describes each date converter tag, beginning with [date:clocklist](#).

date:clocklist

Converts an epoch format long integer representing milliseconds since Jan 1, 1970, 0:00 GMT, into a single-row list with columns representing various date/time fields and formats.

Syntax

```
<date:clocklist  
    listvarname="listvarname"  
    [clock="longinteger"]  
    [timezone="timezone"]/>
```

Note

If you take the result of `date:clocklist` and use it as an input value to `date:convert`, the epoch date returned is different from the originally converted date because `date:clocklist` is 1-based for month and day values, while `date:convert` is 0-based for month and day values.

Parameters

listvarname (required)

Input parameter (see as output, below). Name of a list to create with broken out date formats.

clock (optional)

Input parameter. Long integer value of milliseconds since Jan 1, 1970 0:00 GMT. The default is now, or the current clock value.

timezone (optional)

Input parameter. Time zone name. The default is system default.

Note

Only North American time zones are recognized. For time zones elsewhere in the world, use a plus or minus offset to GMT; for example, `timezone="-3:00"` or `timezone="+3:00"`.

listvarname

Output parameter. A list is created with the following columns:

- `fulldate` – for example, "Saturday, April 6, 2001"
- `fulltime` – for example, "1:05:00 o'clock AM EST"
- `longdate` – for example, "April 6, 2001"
- `longtime` – for example, "1:05:00 AM EST"
- `year` – four-digit year
- `month` – integer month, January=1, December=11
- `dayofmonth` – integer day of month, 1–31
- `dayofweek` – integer day of week 1–7, Sunday=1
- `hour` – integer hour 0–11
- `minute` – integer minute 0–59
- `second` – integer second 0–59
- `millisecond` – integer millisecond 0–999
- `ampm` – integer, am=0, pm=1
- `clock` – integer, milliseconds since Jan 1, 1970 0:00 GMT

date:convert

Converts a specific date to an epoch format integer.

Note

If you take the result of `date:convert` and use it as an input value to `date:clocklist`, the specific date returned is different from the originally converted date because `date:convert` is 0-based for month and day values, while `date:clocklist` is 1-based for month and day values.

Syntax

```
<date:convert
  varname="varname"
  year="year"
  month="month"
  day="day"
  [hour="hour"]
  [ampm="ampm"]
  [minute="minute"]
  [timezone="timezone"]/>
```

Parameters

varname (required)

Input and output parameter. As input, name of a variable to create with the integer date format on output.

year (required)

Input parameter. The year number.

month (required)

Input parameter. The month number (0-based; that is, 0=January, 11=December).

day (required)

Input parameter. The day number (0-based; that is, Sunday=0; Saturday=6.)

hour (optional)

Input parameter. The hour number (default is 0).

ampm (optional)

Input parameter. The am/pm specifier (default is am).

minute (optional)

Input parameter. The minute number (default is 0).

timezone (optional)

Input parameter. A specified time zone.

Note

Only North American time zones are recognized. For time zones elsewhere in the world, use a plus or minus offset to GMT. For example:

- `timezone="-3:00"`
- `timezone="+6:00"`

date:defaulttz

Returns the current time zone in standard international format.

Syntax

```
<date:defaulttz  
  varname="varname"/>
```

Parameters

varname (required)

Input and output parameter. As input, name of a variable to hold the current time zone on output.

Decimal Tags

The decimal tags are used for formatting floating point values as decimal objects in different locales.

Note

Each JSP file containing JSP decimal tags must have the following directive at the top:

```
<%@ taglib prefix="decimal" uri="futuretense_cs/decimal.tld"
%>
```

The following section describes each decimal tag, beginning with [decimal:create](#).

decimal:create

Creates a decimal mapping object, which you can use to convert between a floating point value and a displayable string.

Syntax

```
<decimal:create
    name="decimalobject"
    [ locale="localeobject" ] />
```

Parameters

name (required)

Input and output parameter. As input, name of the decimal object to create on output.

locale (optional)

Input parameter. Locale object to use for this decimal value. If unspecified, the system default applies.

decimal:getdecimal

Converts a floating point value to a displayable string.

Syntax

```
<decimal:getdecimal
    name="decimalobject"
    value="floatvalue"
    varname="decimalvalue" />
```

Parameters

name (required)

Input parameter. Name of the decimal object to convert.

value (required)

Input parameter. Floating-point value to convert.

varname (required)

Input and output parameter. As input, name of a string to hold the formatted decimal value on output.

Description

This tag uses the locale and place count information present in the provided decimal mapping object.

decimal:getmaxplaces

Gets the previously set maximum number of places to the right of the decimal point (or comma).

Syntax

```
<decimal:getmaxplaces  
    name="decimalobject"  
    varname="decimalplaces"/>
```

Parameters

name (required)

Input parameter. Name of the decimal object.

varname (required)

Input and output parameter. As input, name of a variable to hold maximum number of decimal places.

decimal:getminplaces

Gets the previously set minimum number of places to the right of the decimal point or comma.

Syntax

```
<decimal:getminplaces  
    name="decimalobject"  
    varname="decimalplaces"/>
```

Parameters

name (required)

Input parameter. Name of the decimal object.

varname (required)

Input and output parameter. As input, name of the variable to hold minimum number of decimal places on output.

decimal:getthousands

Gets the value set by [decimal:setthousands](#) for this decimal converter object.

Syntax

```
<decimal:getthousands  
    name="decimalobject"  
    varname="true|false"/>
```

Parameters

name (required)

Input parameter. Name of the decimal object.

varname (required)

Input and output parameter. As input, name of the variable to hold the true or false value set by [decimal:setthousands](#).

decimal:readdecimal

Converts a displayable string (created by [decimal:getdecimal](#)) back into a floating point value.

Syntax

```
<decimal:readdecimal  
    name="decimalobject"  
    value="decimalvalue"  
    varname="floatvalue"/>
```

Parameters

name (required)

Input parameter. Name of the decimal object.

value (required)

Input parameter. Formatted decimal value produced by [decimal:getdecimal](#).

varname (required)

Input and output parameter. As input, name of a variable to hold the floating-point value on output.

decimal:setmaxplaces

Sets the maximum number of places to the right of the decimal point or comma.

Syntax

```
<decimal:setmaxplaces  
    name="decimalobject"  
    places="decimalplaces"/>
```

Parameters

name (required)

Input parameter. Name of the decimal object.

places (required)

Input parameter. Maximum number of decimal places to set.

decimal:setminplaces

Sets the minimum number of places to the right of the decimal point or comma.

Syntax

```
<decimal:setminplaces  
    name="decimalobject"  
    places="decimalplaces"/>
```

Parameters

name (required)

Input parameter. Name of the decimal object.

places (required)

Input parameter. Minimum number of decimal places to set.

decimal:setthousands

Sets whether to include a thousands separator, where appropriate, in displayable strings.

Syntax

```
<decimal:setthousands  
    name="decimalobject"  
    value="true|false"/>
```

Parameters

name (required)

Input parameter. Name of the decimal object.

value (required)

Input parameter. Either true or false.

Hash Table Tags

Hash table tags improve CS-Direct Advantage user interface performance. These tags are useful when you are trying quickly and repeatedly to determine whether a list column holds a value.

Note

Each JSP file containing JSP hash table tags must have the following directive at the top:

```
<%@ taglib prefix="hash" uri="futuretense_cs/hash.tld" %>
```

The following section describes each hash table tag, beginning with [hash:add](#).

hash:add

Adds a value to the hash table object.

Syntax

```
<hash:add
    name="objname"
    value="stringval" />
```

Parameters

name (required)
Input parameter. Name of the hash table object.

value (required)
Input parameter. Value to add to the hash table.

hash:contains

Checks for a specified value in the hash table object.

Syntax

```
<hash:contains
    name="objname"
    value="stringval"
    varname="booleanvar" />
```

Parameters

name (required)
Input parameter. Name of the hash table object.

value (required)
Input parameter. Value to check for in the hash table.

varname (required)

Input parameter. Variable that holds a string representing boolean value: `true` or `false`; the value is or is not present in the hash table.

hash:create

Creates a hash table object.

Syntax

```
<hash:create  
  name="objname"  
  [list="listname"]  
  [column="colname"]/>
```

Parameters

name (required)

Input parameter. Name of the hash table object.

list (optional)

Input parameter. Name of a list from which to read to initialize the hash table. Table is created empty, otherwise.

column (optional)

Input parameter. Name of column within the list, if specified, from which to read to initialize the hash table.

hash:hasdata

Checks for the existence of data in hash table object.

Syntax

```
<hash:hasdata  
  name="objname"  
  varname="varname"/>
```

Parameters

name (required)

Input parameter. Name of the hash table object.

varname (required)

Input and output parameter. As input, name of the variable that holds a boolean value on output: `true` if the hash table object has data; `false` if it is empty.

hash:toString

Writes out hash table values to a string.

Syntax

```
<hash:toString  
    name="objname"  
    delim="delimiter"  
    varname="varname" />
```

Parameters

name (required)

Input parameter. Name of the hash table object.

delim (required)

Input parameter. Delimiter used to separate hash values in the output string.

varname (required)

Input and output parameter. As input, name of the variable to which the delimited values are assigned on output.

List Object Tags

You use the list object tags to construct Content Server resultset lists, which are used throughout your elements as arguments for other CS-Direct Advantage tags. For example, you call list object tags and use the subsequent resultset lists as input for shopping cart tags, and assetset tags. You can use all three list object tags at a time to do the following:

- Create an empty list object.
- Add rows to the list object.
- Convert the list object to a Content Server resultset list.

Note

Each JSP file containing JSP list object tags must have the following directive at the top:

```
<%@ taglib prefix="listobject" uri="futuretense_cs/
listobject.tld" %>
```

The following section describes each list object tag, beginning with [listobject:addrow](#).

listobject:addrow

Adds a new row to the end of a list.

Syntax

```
<listobject:addrow name="listname">
  <listobject:argument name="colname-1" value="colvalue-1"/>
  <listobject:argument name="colname-n" value="colvalue-n"/>
</listobject:addrow>
```

Parameters

name (required)

Input parameter. Name of the list object to which to add a new row.

argument (required)

Input parameter. Specify an argument line for each column name/value pair in the row to be added.

Note

The list object name and value inputs are case-sensitive.

Example

```
<!-- Create a list for a call to assetset:getassetlist -->
<listobject:create name="mylist"
columns="attributename,direction"/>
<listobject:addrow name="mylist">
  <listobject:argument name="attributename" value="name"/>
```

```

    <listobject:argument name="direction" value="descending"/>
</listobject:addrow>
<listobject:addrow name="mylist">
    <listobject:argument name="attributename" value="price"/>
    <listobject:argument name="direction" value="descending"/>
</listobject:addrow>
<listobject:tolist name="mylist" listvarname="listout"/>

```

listobject:create

Creates a new list with specific columns but no rows.

Syntax

```

<listobject:create
    name="listname"
    columns="col1,col2,...,coln" />

```

Parameters

name (required)

Input and output parameter. As input, name of the list object to create on output as an empty list with the specified columns.

columns (required)

Input parameter. A string consisting of the column names separated by commas.

Example

```

<!-- Create a list for a call to assetset:getassetlist -->
<listobject:create name="mylist"
columns="attributename,direction"/>
<listobject:addrow name="mylist">
    <listobject:argument name="attributename" value="name"/>
    <listobject:argument name="direction" value="descending"/>
</listobject:addrow>
<listobject:addrow name="mylist">
    <listobject:argument name="attributename" value="price"/>
    <listobject:argument name="direction" value="descending"/>
</listobject:addrow>
<listobject:tolist name="mylist" listvarname="listout"/>

```

listobject:tolist

Converts a list object to a resultset list.

Syntax

```
<listobject:tolist  
    name="listname"  
    listvarname="varname" />
```

Parameters

name (required)

Input parameter. Name of the list object to convert.

listvarname (required)

Input and output parameter. As input, name of the list variable into which to write the resultset list on output.

Example

```
<!-- Create a list for a call to assetset:getassetlist -->  
<listobject:create name="mylist"  
columns="attributename,direction"/>  
<listobject:addrow name="mylist">  
    <listobject:argument name="attributename" value="name"/>  
    <listobject:argument name="direction" value="descending"/>  
</listobject:addrow>  
<listobject:addrow name="mylist">  
    <listobject:argument name="attributename" value="price"/>  
    <listobject:argument name="direction" value="descending"/>  
</listobject:addrow>  
<listobject:tolist name="mylist" listvarname="listout"/>
```


Locale Tags

You use the locale tags to generate a locale object, which is used to describe the desired locale for various other tags in the system.

Note

Each JSP file containing JSP locale tags must have the following directive at the top:

```
<%@ taglib prefix="locale" uri="futuretense_cs/locale.tld"
%>
```

The following section describes the locale tag, [locale:create](#).

locale:create

Creates a locale descriptor for use in other tags.

Syntax

```
<locale:create
    varname="localeobject"
    country="isocountry"
    language="isolanguage"
    localename="locale name"/>
```

Parameters

varname (required)

Input and output parameter. As input, name of a variable to which to assign the locale object on output.

country (required)

Input parameter. ISO-standard country name.

language (required)

Input parameter. ISO-standard language name.

localename (required)

Input parameter. Name of the locale; for example, en_US for the United States, ja_JP for Japan, or fr_FR for France.

Miscellaneous Tags

The following section describes the [misc:getlistcolumnnames](#) tag.

Note

Each JSP file containing JSP miscellaneous tags must have the following directive at the top:

```
<%@ taglib prefix="misc" uri="futuretense_cs/misc.tld" %>
```

misc:getlistcolumnnames

Returns the names of all the columns in an input list.

Syntax

```
<misc:getlistcolumnnames  
    list="listobject"  
    listvarname="varname" />
```

Parameters

list (required)

Input parameter. Name of the list object to interrogate.

listvarname (required)

Input and output parameter. As input, variable list name to write with the list of column names.

Searchstate Tags

A searchstate represents a set of constraints that apply to a pool of products. A **constraint** can be either a restriction based on a single attribute, or another searchstate (called a **nested constraint**). Searchstate objects contain:

- a list of attribute names and their values
- a list of index names and their rich-text search criteria.

Note

Each JSP file containing JSP searchstate tags must have the following directive at the top:

```
<%@ taglib prefix="searchstate" uri="futuretense_cs/
searchstate.tld" %>
```

The following section describes each searchstate tag, beginning with [searchstate:addlikeconstraint](#).

searchstate:addlikeconstraint

Adds an attribute name/value constraint into a new or existing searchstate object.

Syntax

```
<searchstate:addlikeconstraint
    name="ssname"
    [bucket="bucketname"]
    [typename="assettype"]
    attribute="attribname"
    [list="listname"]
    [immediateonly="true|false"]
    [caseinsensitive="true|false"]/>
```

Parameters

name (required)

Input parameter. Name of an object representing a searchstate.

bucket (optional)

Input parameter. Bucket name. If not specified, the attribute name is used. Use this parameter to indicate which constraint you need when the same attribute is used in multiple constraints. A bucket is an alias and functions similar to an inner join in SQL; when you have multiple tables that have columns of the same name, you must specify the table name to indicate which column you are requesting.

typename (optional)

Input parameter. Internal asset name for the attribute (either CAttributes for content attribute, or PAttributes for product attribute). If you do not specify typename, a value is supplied from a property in the gator.ini property file: mwb.defaultattributes=PAtributes. The default is PAttributes and the value may be changed.

attribute (required)

Input parameter. Name of the attribute to constrain.

list (optional)

Input parameter. A list of the constrained values for the attribute. If specified, one or more of the values must match the attribute for a product to meet the constraint. The default is that all assets that have any value for the attribute match the constraint.

immediateonly (optional)

Input parameter. `true` indicates that the search is limited to values directly associated with the specified attribute; `false` (the default) extends the search to include values inherited from a parent.

caseinsensitive (optional)

Input parameter. `true` indicates that the comparison is case-insensitive; `false` (the default) considers case in the comparison.

Description

The object referred to by name is updated to include the new constraint. This tag is similar to a database LIKE operation and accepts wild cards (%). If the attribute name is already in the searchstate, then the new constraint replaces the old constraint.

searchstate:addnestedconstraint

Adds a nested searchstate as part of the constraint.

Syntax

```
<searchstate:addnestedconstraint
  name="newssname"
  bucket="bucketname"
  searchstate="ssname"/>
```

Parameters

name (required)

Input parameter. Name of the searchstate object to add.

bucket (required)

Input parameter. An invented name defined by [searchstate:findconstraint](#). Use this parameter to indicate which constraint you need when the same attribute is used in multiple constraints. A bucket is an alias and functions similar to an inner join in SQL; when you have multiple tables that have columns of the same name, you must specify the table name to indicate which column you are requesting.

searchstate (required)

Input parameter. Name of the searchstate object to nest inside of the object specified by name.

searchstate:addrangeconstraint

Adds a range constraint for a specific attribute name.

Syntax

```
<searchstate:addrangeconstraint
    name="ssname"
    [bucket="bucketname"]
    [typename="assettype"]
    attribute="attribname"
    lower="lowrange"
    upper="uprange"
    [caseinsensitive="true|false"]/>
```

Parameters

name (required)

Input parameter. Name of an object representing a searchstate.

bucket (optional)

Input parameter. Bucket name. If not specified, the attribute name is used. Use this parameter to indicate which constraint you need when the same attribute is used in multiple constraints. A bucket is an alias and functions similar to an inner join in SQL; when you have multiple tables that have columns of the same name, you must specify the table name to indicate which column you are requesting.

typename (optional)

Input parameter. Internal asset name for the attribute (either CAttributes for content attribute, or PAttributes for product attribute). If you do not specify TYPENAME, a value is supplied from a property in the gator.ini property file: mwb.defaultattributes=PAtributes. The default is PAttributes and the value may be changed.

attribute (required)

Input parameter. Name of the attribute to constrain.

lower | lowerequal (required)

Input parameter. Bottom end of the range.

upper | upperequal (required)

Input parameter. Top end of the range.

caseinsensitive (optional)

Input parameter. true indicates that the comparison is case-insensitive; false (the default) considers case in the comparison.

Example

```
<!-- if there is a rich text search parameter,
    use that instead of any parametric search parameters -->
<%
    String aid = ics.GetVar("gdSearch");
    if (aid == true)
    {
%>
        <searchstate:addrichtextconstraint name="mysearch"
        attribute="name" value="gdsearch" confidence=".1"/>
```

```

<%
  }
    else
    {
%>
    <searchstate:addsimplestandardconstraint name="mysearch"
      attribute="p1" VALUE="p2"/>
    <searchstate:addrangeconstraint name="mysearch"
      attribute="price"
      lower="low" upper="high"/>
<%
  }
%>

```

searchstate:addrichtextconstraint

Adds an index name and rich-text expression to the list of rich-text criteria for items.

Syntax

```

<searchstate:addrichtextconstraint
  name="ssname"
  [bucket="bucketname"]
  [typename="assettype"]
  attribute="attribname"
  value="criteria"
  [parser="parsername"]
  confidence="minlevel"
  [maxcount="number"]/>

```

Note

This tag requires that the attribute was created as indexed.

Parameters

name (required)

Input parameter. Name of an object representing a searchstate.

bucket (optional)

Input parameter. Bucket name. If not specified, the attribute name is used. Use this parameter to indicate which constraint you need when the same attribute is used in multiple constraints. A bucket is an alias and functions similar to an inner join in SQL; when you have multiple tables that have columns of the same name, you must specify the table name to indicate which column you are requesting.

typename (optional)

Input parameter. Internal asset name for the attribute (either CAttributes for content attribute, or PAttributes for product attribute). If you do not specify TYPENAME, a value is supplied from a property in the gator.ini property file: mwb.defaultattributes=PAtributes. The default is PAttributes and the value may be changed.

attribute (required)

Input parameter. Name of the attribute to constrain.

value (required)

Input parameter. Rich-text search criteria, which should apply to the attribute.

parser (optional)

Input parameter. Search-engine-dependent rich text parser to use.

confidence (required)

Input parameter. Minimum confidence level for the match.

maxcount (optional)

Input parameter. Maximum number of answers desired for the match. If this parameter is not specified, the number of results is limited only by the confidence and the number of products.

Description

The object described by the name parameter is updated to reflect the additional search criteria. If the attribute name is already mentioned as part of a rich-text constraint in the searchstate, then the existing constraint is removed first.

Example

```
<!-- if there is a rich text search parameter,
      use that instead of any parametric search parameters -->
<%
    String aid = ics.GetVar("gdSearch");
    if (aid == true)
    {
%>
        <searchstate:addrichtextconstraint name="mysearch"
attribute="name" value="gdsearch" confidence=".1"/>
<%
    }
        else
        {
%>
        <searchstate:addsimplestandardconstraint name="mysearch"
            attribute="p1" VALUE="p2"/>
        <searchstate:addrangeconstraint name="mysearch"
attribute="price"
            lower="low" upper="high"/>
<%
    }
%>
```

searchstate:addsimplelikeconstraint

Adds an attribute name/single value constraint to an existing searchstate.

Syntax

```
<searchstate:addsimplelikeconstraint
    name="ssname"
    [bucket="bucketname"]
    [typename="assettype"]
    attribute="attribname"
    value="value"
    [immediateonly="true|false"]/>
```

Parameters

name (required)

Input parameter. Name of an object representing a searchstate.

bucket (optional)

Input parameter. Bucket name. If not specified, the attribute name is used. Use this parameter to indicate which constraint you need when the same attribute is used in multiple constraints. A bucket is an alias and functions similar to an inner join in SQL; when you have multiple tables that have columns of the same name, you must specify the table name to indicate which column you are requesting.

typename (optional)

Input parameter. Internal asset name for the attribute (either CAttributes for content attribute, or PAttributes for product attribute). If you do not specify TYPENAME, a value is supplied from a property in the gator.ini property file: mwb.defaultattributes=PAtributes. The default is PAttributes and the value may be changed.

attribute (required)

Input parameter. Name of the attribute to constrain.

value (required)

Input parameter. Single value that constrains the attribute.

immediateonly (optional)

Input parameter. true indicates that the search is limited to values directly associated with the specified attribute; false (the default) extends the search to include values inherited from a parent.

Description

This tag is the simple version of addlikeconstraint. This tag is similar to a database LIKE operation and accepts wild cards (%). The object referred to by name is updated to reflect the new constraint. If the attribute name is already in the searchstate, then the new constraint replaces the old constraint.

searchstate:addsimplestandardconstraint

Adds an attribute name/single value constraint to an existing searchstate.

Syntax

```
<searchstate:addsimplestandardconstraint
    name="ssname"
    [bucket="bucketname"]
    [typename="assettype"]
    attribute="attribname"
    value="value"
    [immediateonly="value"]/>
```

Parameters

name (required)

Input parameter. Name of an object representing a searchstate.

bucket (optional)

Input parameter. Bucket name. If not specified, the attribute name is used. Use this parameter to indicate which constraint you need when the same attribute is used in multiple constraints. A bucket is an alias and functions similar to an inner join in SQL; when you have multiple tables that have columns of the same name, you must specify the table name to indicate which column you are requesting.

typename (optional)

Input parameter. Internal asset name for the attribute (either CAttributes for content attribute, or PAttributes for product attribute). If you do not specify TYPENAME, a value is supplied from a property in the gator.ini property file: mwb.defaultattributes=PAtributes. The default is PAttributes and the value may be changed.

attribute (required)

Input parameter. Name of the attribute to constrain.

value (required)

Input parameter. Single value that constrains the attribute.

immediateonly (optional)

Input parameter. true indicates that the search is limited to values directly associated with the specified attribute; false (the default) extends the search to include values inherited from a parent.

Description

This tag is the simple version of [searchstate:addstandardconstraint](#). The object referred to by name is updated to reflect the new constraint. If the attribute name is already in the searchstate, then the new constraint replaces the old constraint.

Example

```
<!-- if there is a rich text search parameter,
use that instead of any parametric search parameters -->
<ics:if condition="IsVariable.gdSearch=true">
<ics:then>
<searchstate:addrichtextconstraint name="mysearch"
attribute="name"
```

```

        value="variables.gdsearch" confidence=".1"/>
</ics:then>
<ics:else>
<searchstate:addsimplestandardconstraint name="mysearch"
    attribute="variables.p1" value="Variables.p2"/>
<searchstate:addrangeconstraint name="mysearch" attribute="price"
    lower="variables.low" upper="Variables.high"/>
</ics:else>
</ics:if>

```

searchstate:addstandardconstraint

Adds an attribute name/value constraint into a new or existing searchstate object.

Syntax

```

<searchstate:addstandardconstraint
    name="ssname"
    [bucket="bucketname"]
    [typename="assettype"]
    attribute="attribname"
    [list="listname"]
    [immediateonly="true|false"]
    [caseinsensitive="true|false"]/>

```

Description

The object referred to by name is updated to include the new constraint. If the attribute name is already in the searchstate, then the new constraint replaces the old constraint.

Parameters

name (required)

Input parameter. Name of an object representing a searchstate.

bucket (optional)

Input parameter. Bucket name. If not specified, the attribute name is used. Use this parameter to indicate which constraint you need when the same attribute is used in multiple constraints. A bucket is an alias and functions similar to an inner join in SQL; when you have multiple tables that have columns of the same name, you must specify the table name to indicate which column you are requesting.

typename (optional)

Input parameter. Internal asset name for the attribute (either CAttributes for content attribute, or PAttributes for product attribute). If you do not specify TYPENAME, a value is supplied from a property in the gator.ini property file: mwb.defaultattributes=PAtributes. The default is PAttributes and the value may be changed.

attribute (required)

Input parameter. Name of the attribute to constrain.

list (optional)

Input parameter. A list of the constrained values for the attribute. If specified, one or more of the values must match the attribute for a product to meet the constraint. The default is that all assets that have any value for the attribute match the constraint.

`immediateonly` (optional)

Input parameter. `true` indicates that the search is limited to values directly associated with the specified attribute; `false` (the default) extends the search to include values inherited from a parent.

`caseinsensitive` (optional)

Input parameter. `true` indicates that the comparison is case-insensitive; `false` (the default) considers case in the comparison.

searchstate:create

Builds an empty searchstate object.

Syntax

```
<searchstate:create
  name="ssname"
  [op="and|or"]/>
```

Parameters

`name` (required)

Input and output parameter. As input, name of the searchstate object to create on output.

`op` (optional)

Input parameter. Searchstate type, either `and` or `or`. The default is `and`.

Example

```
<!-- Iterate through a list of attribute values. -->
<!-- First, create a blank searchstate. -->
<searchstate:create name="ss"/>
<!-- Associate the blank searchstate with a new asset set. -->
<assetset:setsearchedassets name="as" constraint="ss"/>
<!-- Get a set of all values for the given attribute. -->
<assetset:getattributevalues ordering="descending" name="as"
  attribute="price" listvarname="results"/>
```

searchstate:deleteconstraint

Deletes a specific attribute constraint from a searchstate.

Syntax

```
<searchstate:deleteconstraint
  name="ssname"
  [bucket="bucketname"]
  attribute="attribname"/>
```

Parameters

`name` (required)

Input parameter. Name of an object representing a searchstate.

bucket (optional)

Input parameter. Bucket name. If not specified, the attribute name is used. Use this parameter to indicate which constraint you need when the same attribute is used in multiple constraints. A bucket is an alias and functions similar to an inner join in SQL; when you have multiple tables that have columns of the same name, you must specify the table name to indicate which column you are requesting.

typename (optional)

Input parameter. Internal asset name for the attribute (either CAttributes for content attribute, or PAttributes for product attribute). If you do not specify TYPENAME, a value is supplied from a property in the gator.ini property file: mwb.defaultattributes=PAtributes. The default is PAttributes and the value may be changed.

attribute (optional)

Input parameter. Note this parameter is required if there is no bucket present. Name of the attribute whose constraint should be deleted.

searchstate:duplicate

Duplicates a searchstate object.

Syntax

```
<searchstate:duplicate
    name="ssname"
    varname="varname"/>
```

Parameters**name (required)**

Input parameter. Name of the searchstate object.

varname (required)

Input and output parameter. As input, name of the new searchstate object to hold the duplicate on output.

searchstate:findconstraint

Checks a searchstate for a constraint to a specific attribute.

Syntax

```
<searchstate:findconstraint
    name="ssname"
    [bucket="bucketname"]
    attribute="attribname"
    varname="varname"/>
```

Parameters**name (required)**

Input parameter. Name of an object representing a searchstate.

`bucket` (optional)

Input parameter. Bucket name. If not specified, the attribute name is used. Use this parameter to indicate which constraint you need when the same attribute is used in multiple constraints. A bucket is an alias and functions similar to an inner join in SQL; when you have multiple tables that have columns of the same name, you must specify the table name to indicate which column you are requesting.

`attribute` (required)

Input parameter. Name of the attribute to constrain.

`varname` (required)

Input and output parameter. As input, name of a variable into which to write the constraint type on output. Note the variants below.

The following field applies only to nested constraints:

`objvarname` (required)

Name of an object to create with the contents of the nested constraint.

The following fields apply to `standard`, `simpleStandard`, `like`, and `simpleLike` constraints:

`attrtypevarname` (optional)

Name of a variable into which to write the attribute asset type name for the constraint.

`attrvarname` (optional)

Name of a variable into which to write the attribute name for the constraint.

`listvarname` (required)

Name of a variable into which to write the list of values for the attribute, if found. If not found, the variable is set to null. The list has a single column named `value`.

`immediateonlyvarname` (optional)

Name of a variable into which to write the `immediateonly` constraint value.

`caseinsensitivevarname` (optional)

Name of a variable into which to write the `caseinsensitive` constraint value.

The following fields apply only to rich text constraints:

`attrtypevarname` (optional)

Name of a variable into which to write the attribute asset type name for the constraint.

`attrvarname` (optional)

Name of a variable into which to write the attribute name for the constraint.

`expressionvarname` (optional)

Name of the variable into which to write the rich text expression.

`parservarname` (optional)

Name of the variable into which to write the rich text parser name.

`confidencevarname` (optional)

Name of the variable into which to write the confidence.

`maxcountvarname` (optional)

Name of the variable into which to write the max answers.

The following fields apply only to range constraints:

`attrtypevarname` (optional)

Name of a variable into which to write the attribute asset type name for the constraint.

`attrvarname` (optional)

Name of a variable into which to write the attribute name for the constraint.

`lowervarname` | `lowerequalvarname` (required)
Name of the variable into which to write the lower constraint.

`uppervarname` | `upperequalvarname` (required)
Name of the variable into which to write the upper constraint.

`immediateonlyvarname` (optional)
Name of the variable into which to write the `immediateonly` value constraint.

`caseinsensitivevarname` (optional)
Name of a variable into which to write the `caseinsensitive` constraint value.

Description

If found, data values about the constraint are returned.

searchstate:fromstring

Provides the ability for a `searchstate` object to be initialized from its string representation.

Syntax

```
<searchstate:fromstring
  name="objname"
  value="stringval" />
```

Parameters

`name` (required)
Input and output parameter. As input, name of the output object to be reinitialized with the string passed in the `value` argument.

`value` (required)
Input parameter. String value that initializes the `searchstate` object.

searchstate:tostring

Converts a `searchstate` object into its string representation that is suitable for various uses, such as saving in a session variable or packing into a URL.

Syntax

```
<searchstate:tostring
  name="objname"
  varname="varname" />
```

Parameters

`name` (required)
Input parameter. Name of the object.

`varname` (required)
Input and output parameter. As input, name of the string that is assigned the string representation on output.

Session Tags

The following section describes the [sessionobjects:flush](#) tag.

Note

Each JSP file containing JSP session tags must have the following directive at the top:

```
<%@ taglib prefix="sesssion" uri="futuretense_cs/  
session.tld" %>
```

sessionobjects:flush

Flushes object storage for all expired sessions.

Syntax

```
<sessionobjects:flush  
    [timestamp="cutofftime"]/>
```

Parameters

timestamp (optional)

Input parameter. Cutoff timestamp for objects. All objects older than the timestamp are removed.

String Tags

These tags perform various string manipulations.

Note

Each JSP file containing JSP string tags must have the following directive at the top:

```
<%@ taglib prefix="string" uri="futuretense_cs/string.tld"
%>
```

The following section describes each string tag, beginning with [string:fromobject](#).

string:fromobject

Converts a named Content Server object back to a string.

Syntax

```
<string:fromobject
    value="inputstring"
    varname="varname" />
```

Parameters

value (required)

Input parameter. Name of the Content Server object.

varname (required)

Input and output parameter. As input, name of the variable to hold the string on output.

string:length

Returns the length of an input string in the specified variable.

Syntax

```
<string:length
    value="inputstring"
    varname="varname" />
```

Parameters

value (required)

Input parameter. String to size.

varname (required)

Input and output parameter. As input, the name of variable to hold the length of the string on output.

string:stream

Streams text in an HTML context.

Syntax

```
<string:stream
    value="input string"
    [variable="variable name"]
    [ssvariable="session variable name"]
    [property="property name"]
    [list="list name"]
    [column="column name"] />
```

Parameters

value (required)

Input parameter. String to stream.

variable (optional)

Name of the Content Server variable.

ssvariable (optional)

Name of the Content Server session variable.

property (optional)

Name of the property from the `futuretense.ini` property file.

list (optional)

Name of the list.

column (optional)

Name of the field in the list. You must use `column` when you use `list`.

Description

Use this tag as an alternative to [CSVAR](#) to display text literally as stored in dCS database, that is, with no browser interpretation or Content Server evaluation of variable names that are present in the text. This feature is useful for Edit or Inspect forms. This tag streams character text after HTML-escaping special characters, as well as all characters greater than 127. Any strings that are special to Content Server evaluation engine (for example, `Variable.id`) are not evaluated.

string:toobject

Converts string to a named Content Server object.

Syntax

```
<string:toobject  
    value="inputstring"  
    varname="varname" />
```

Parameters

value (required)

Input parameter. String to convert.

varname (required)

Input and output parameter. As input, name of the object to create on output.

Text Format Tags

The following section describes each of the text format tags, beginning with [textformat:length](#).

Note

Each JSP file containing JSP text format tags must have the following directive at the top:

```
<%@ taglib prefix="textformat" uri="futuretense_cs/
textformat.tld" %>
```

textformat:length

Returns the length of an input string in the specified variable.

Syntax

```
<textformat:length
    value="inputstring"
    varname="varname" />
```

Parameters

value (required)

Input parameter. The string to size.

varname (required)

Input and output parameter. As input, the name of variable to hold the length of the string on output.

textformat:tostring

Converts a list to a displayable string.

Syntax

```
<textformat:tostring
    list="listname"
    [field="fieldname"]
    varname="varname"
    [maxcount="number"]
    [ellipsis="true|false"] />
```

Parameters

list (required)

Input parameter. Name of the list to convert.

field (optional)

Input parameter. A column name in the list.

`varname` (required)

Input and output parameter. As input, the name of variable to hold the string on output.

`maxcount` (optional)

Input parameter. Maximum number of entries to display.

`ellipsis` (optional)

Input parameter. Determines if an ellipsis string (...) is appended if more than the maximum number of entries exist in this list. Valid options are `true` (the default value) and `false`.

URL Tags (Deprecated)

As of Content Server version 4.0, replaced by [Render Tags](#).

URL tags provide basic packing and unpacking for URLs.

Note

Each JSP file containing JSP URL tags must have the following directive at the top:

```
<%@ taglib prefix="url" uri="futuretense_cs/url.tld" %>
```

The following section describes each URL tag, beginning with [url:pack \(Deprecated\)](#).

url:pack (Deprecated)

As of Content Server version 4.0, replaced by [Render Tags](#).

Syntax

```
<url:pack  
    value="inputstring"  
    varname="varname"/>
```

Parameters

value (required)

Input parameter. Input string.

varname (required)

Input and output parameter. As input, name of the variable to hold the URL-encoded string on output.

Description

The string held by the output variable is a packed string that is compatible with both iAS URL and WebLogic URL conventions. No unpacking is necessary if the packed string is used as part of a URL because the application server does it automatically.

url:unpack (Deprecated)

As of Content Server version 4.0, replaced by [Render Tags](#).

Syntax

```
<url:unpack  
    value="inputstring"  
    varname="varname"/>
```

Parameters

value (required)

Input parameter. Packed input string.

varname (required)

Input and output parameter. As input, name of the variable to hold the unpacked string on output.

Description

Use this tag for form elements that have been packed because they contain questionable JSP characters. You do not need to unpack URL values because the application server does it.

Visitor Data Manager Tags

The visitor data manager is the data access method that CS-Engage uses when processing rules. The visitor data manager enables you to gather, store, and retrieve visitor information from both the local, Content Server database as well as various external databases — Transact, for example.

When you create visitor attributes, history attributes, and history types for defining segments, the visitor data manager stores them in the local CS-Engage database. If you are using Transact, you can select Transact visitor attributes when defining segments as well. Then, when a method that determines which segments a visitor qualifies for is invoked, the visitor data manager checks the appropriate database for the needed visitor data. It provides a single access method to all of your visitor data.

There are additional visitor data manager object methods available with CS-Direct Advantage that you can use to link visitor sessions.

To maintain visitor session information after visitors end their sessions, you use persistent cookies. These cookies serve two purposes: to store the data and to mark the session with an identifier.

A visitor's Transact ID is also an identifier that can mark a visitor session. If a visitor performs an action on the site that requires him to log into Transact, his Transact ID is passed to his hard drive and identifies him as a valid user.

The object methods in the preceding list enable you to link visitor sessions marked by different identifiers by assigning the same alias to those sessions.

There are additional visitor data manager object tags available with CS-Engage.

Note

Each JSP file containing JSP visitor data manager tags must have the following directive at the top:

```
<%@ taglib prefix="vdm" uri="futuretense_cs/vdm.tld" %>
```

Use the [vdm:recordhistory](#) and [vdm:setscalar](#) tags to write visitor data to the CS-Engage database. Use the [vdm:recordhistory](#) tag to write a timestamped record of the data defined in a history type to the database. This data could include information from several history attributes, depending on the history type. Use the [vdm:setscalar](#) tag to write data for individual visitor attributes to the database.

Use the `vdm:get` tags ([vdm:getaccessid](#), [vdm:getalias](#), [vdm:getcommerceid](#)) to retrieve visitor data from either the local or external databases. Whenever one of the [Commerce Context Tags](#) invokes a segment tag, they retrieve this data -- you do not have to use the `vdm:get` tags in those cases.)

Note

Because the visitor data manager does not track revisions, you cannot create segments based on changes in visitor data. For example, two sessions ago a visitor changed his job description from “technical” to “financial.” When that change was written to the database, the value “financial” replaced the value “technical” and there is no record of his previous job description. Now, when segments based on job description are calculated, he qualifies only for the ones that include “financial” as an option.

The following section describes each vdm tag, beginning with [vdm:flushinactive](#).

vdm:flushinactive

Removes unused data recorded for visitors that have not returned to the site within a specified period of time.

Syntax

```
<vdm:flushinactive  
    startdate="timestamp"/>
```

Parameters

startdate (required)

Input parameter. Cutoff timestamp in milliseconds since Jan 1, 1970.

Description

This tag removes all data accumulated for visitors who have not been to the site since the specified cutoff date. All visitors who have been to the site since the cutoff date are considered current, and their data is not removed.

Note

This tag is for maintenance purposes only, and should not be used as part of site delivery. It can take a significant time to run.

vdm:getaccessid

Retrieves a copy of the visitor's access identifier, if there is one.

Syntax

```
<vdm:getaccessid  
    key="pluginname"  
    varname="varname"/>
```

Parameters

key (required)

Input parameter. Name of the IConnection plug-in. There is currently only one valid key name: tx for Transact.

varname (required)

Input and output parameter. As input, name of a variable to create on output to hold the access identifier.

Description

Use this tag when you have an IConnection plug-in for Commerce Connector that is not for Transact.

The visitor data manager passes this access identifier to Commerce Connector in order to obtain visitor data from an external database. Having an access identifier is equivalent to being logged into that external database. If the visitor does not have an access ID, there is no external visitor data available for this person.

Note

Setting this tag to `<vdm:getaccessid key="tx"/>` is the equivalent of using `<vdm:getcommerce id/>`

vdm:getalias

Retrieves the alias string for the current visitor if there is one.

Syntax

```
<vdm:getalias  
    key="keyvalue"  
    varname="varname"/>
```

Parameters

key (required)

Input parameter. Identifying key value for the current visitor (for example, a site cookie).

varname (required)

Input and output parameter. As input, name of the variable to assign the value of the alias on output.

Description

Use this tag for debugging during the development phase.

vdm:getcommerceid

Retrieves a copy of the visitor's commerce identifier, that is, the visitor's Transact ID.

Note

Using the `<vdm:getcommerce id/>` tag is the equivalent of setting `<vdm:getaccessid id= "tx"/>`.

Syntax

```
<vdm:getcommerceid
  varname="varname"/>
```

Parameters

varname (required)

Input and output parameter. As input, name of a variable to which the commerce identifier is assigned on output.

Description

The visitor data manager passes retrieved identifier to Commerce Connector in order to obtain Transact-based visitor data. Therefore, having a commerce identifier is equivalent to being logged into Transact. If the visitor does not have a commerce identifier, there is no Transact-based visitor data available for this person.

vdm:setaccessid

Sets (or clears) the visitor's access identifier.

Syntax

```
<vdm:setaccessid
  key="pluginname"
  [id="userid"]/>
```

Parameters

key (required)

Input parameter. Name of the IConnection plug-in. There is currently only one valid key name: tx for Transact.

id (optional)

Input parameter. The visitor's commerce identifier, user ID. Omit the ID to clear the commerce identifier for the current visitor.

Description

Having an access ID set for a visitor is the equivalent of logging into an external database (other than Transact's). Use this tag when you need to check whether or not the visitor is logged in to an external database.

Additionally, the visitor data manager passes this ID to Commerce Connector in order to obtain visitor data from an external database. If the visitor does not have an access identifier, there is no external visitor data available for this person.

Note

Setting this tag to `<vdm:setaccessid key="tx"/>` is the equivalent of using `<vdm:setcommerceid/>`

vdm:setalias

Sets the alias string for the current visitor using the specified alias key.

Syntax

```
<vdm:setalias
    key="keyvalue"
    value="aliasvalue"/>
```

Parameters

key (required)

Input parameter. Kind of alias, that is, an arbitrary string that describes the kind of link being set up (for example, a cookie).

value (required)

Input parameter. Value of the alias.

Description

`vdm:setalias` associates an alias with the current session. Use this tag to link sessions together so that session data from multiple sessions can be linked to the same (current) visitor.

For more information about working with aliases, see the *dCS Developer's Guide*.

vdm:setcommerceid

Sets or clears the visitor's commerce identifier; that is, the visitor's Transact ID.

Syntax

```
<vdm:setcommerceid  
  [id="userid"]/>
```

Parameters

id (optional)

Input parameter. Visitor's commerce identifier, user ID. Omit the ID to clear the commerce identifier for the current visitor.

Description

Having a commerce ID set for a visitor is the equivalent of logging into Transact. Use this tag when you need to check whether or not the visitor is logged in to Transact.

Additionally, the visitor data manager passes this ID to Commerce Connector in order to obtain Transact-based visitor data. If the visitor does not have a commerce identifier, there is no Transact-based visitor data available for this person.

For information about additional tags and parameters that are used with this one, see the descriptions of the [csuser:authenticate](#) and the user ID buyer parameter.

Note

Using the `<vdm:setcommerce id/>` tag is the equivalent of setting `<vdm:setaccessid id= "tx"/>`.

Chapter 12

CS-Engage JSP Tags

This chapter describes the functionality and JSP syntax of the CS-Engage object methods. These methods are implemented as Content Server custom application tags, which invoke application business logic for applications built with Content Server.

This chapter serves as your reference for the tags that you use to design application pages that gather visitor data and display recommendations or promotions (merchandising information) based on that data. Use the Content Server Direct Advantage object methods to design the catalog pages. Then use the CS-Engage object methods to display merchandising information on those pages, information that changes based on both the identity of the visitor, and, in some cases, the product or content that is currently displayed.

This chapter contains the following sections:

- [Assetset Tags](#)
- [Commerce Context Tags](#)
- [Visitor Data Manager Tags](#)

Assetset Tags

Assetset tags represent a group of assets that can be described either by a search constraint or by a list. An assetset can include different types of assets; it is not limited to assets of one type.

There are also assetset tags available with CS-Direct Advantage. See [Chapter 11, “CS-Direct Advantage JSP Tags”](#) for information.

Note

Each JSP file containing JSP assetset tags must have the following directive at the top:

```
<%@ taglib prefix="assetset" uri="futuretense_cs/
assetset.tld" %>
```

The following section describes the CS-Engage assetset tag, [assetset:establishratings](#).

assetset:establishratings

Enumerates the assets in the named set and recalculates the rating for each one according to the rules that apply to it.

Syntax

```
<assetset:establishratings
    name="assetsetname"
    [defaultrating="rating"]/>
```

Parameters

name (required)

Input parameter. Name of the assetset object.

defaultrating (optional)

Input parameter. Default rating for assets that do not have one. If you do not declare a value, unrated assets are assigned a default rating of 50 on a scale of 0-100.

Description

Use this tag is to refresh the ratings of the assets in the set when visitor data that could effect their ratings has changed since the first time the assets' ratings were requested.

If the assetset on which it is invoked is not sufficiently constrained, this tag might take considerable time and memory to return a result. Web designers should take this into account when designing the site, and try to limit the size of assetsets on which this operation is performed.

Ratings are established (refreshed) and stored for all the assets in the set. Assets that return with a value of 0 are deleted from the set.

Note

If all the assets return with a 0 rating, the assetset is empty. Be sure to check the assetset after you use this tag to determine whether the set has assets. For example, define a String variable and compare to 0, and then code an appropriate action if the assetset is empty.

This tag invokes the [commercecontext:getratings](#) tag. For information about ratings, see the *CSEE Developer's Guide*.

Commerce Context Tags

During a visitor's session, a visitor context is created for that visitor. Five types of objects are placed there:

- Current cart (see [Chapter 11, "CS-Direct Advantage JSP Tags"](#) for descriptions of the cart JSP tags)
- List of segments for which the visitor qualifies
- List of promotions for which the visitor qualifies
- Time object that is used for calculating time-based rules
- Utility object that gives you access to product attributes

All of these objects, when present, are persistent throughout the visitor session (that is, they are stored in the Content Server database and then flushed when the session ends). The visitor context is the rules engine plus any objects that have been placed in this context. There is one visitor context per session.

The rules engine is a CS-Engage component that uses a chaining algorithm to calculate the rules that define segments, recommendations, and promotions. It imposes the conditions, rules, and actions set up for segments, recommendations, and promotions on the objects placed in the visitor context.

The commerce context encompasses the visitor context and gives you access to it. The commerce context object tags are how you place objects in the visitor context. In other words, these tags are your interface to the visitor context.

The commerce context tags available with CS-Direct Advantage tags are described in [Chapter 11, "CS-Direct Advantage JSP Tags."](#)

Use these object tags to display recommendations and promotions on your application pages.

Note

Each JSP file containing JSP commerce context tags must have the following directive at the top:

```
<%@ taglib prefix="commercecontext" uri="futuretense_cs/
commercecontext.tld" %>
```

The following section describes each commerce context tag, beginning with [commercecontext:calculatepromotions](#).

commercecontext:calculatepromotions

Creates a list of the promotions that the current visitor is eligible for.

Syntax

```
<commercecontext:calculatepromotions/>
```

Parameters

None.

Description

This tag examines the available visitor data for the visitor, compares it to the visitor data that defines the segments associated with all the possible promotions (in other words, it invokes [commercecontext:calculatesegments](#)), and then lists only the promotions that are a match. It lets you recalculate promotion membership.

This tag is invoked whenever you use any of these tags:

```
commercecontext:discountcart  
commercecontext:discounttempcart  
commercecontext:getpromotions  
commercecontext:getrecommendations  
commercecontext:getsinglerecommendation
```

Use this tag to recalculate or refresh the promotion list when visitor data that could effect the promotion list for this visitor has changed since the promotion list was last requested during this session. This action is done the first time promotion information is required in a session.

commercecontext:calculatesegments

Lists the segments to which the visitor belongs.

Syntax

```
<commercecontext:calculatesegments/>
```

Parameters

None.

Description

This tag examines the available visitor data, compares it to the data types that define the segments, and then lists the segments that are a match. It lets you recalculate segment membership based on current information.

This tag is invoked when you use any of these tags:

```
commercecontext:calculatepromotions  
commercecontext:discountcart  
commercecontext:discounttempcart  
commercecontext:getpromotions
```

```
commercecontext:getratings  
commercecontext:getrecommendations  
commercecontext:getsegments  
commercecontext:getsinglerecommendation
```

Use this tag during the debugging phase of development. The segment list object is updated to contain only the segments that the visitor currently qualifies for. This action is done the first time segment information is required in a session.

commercecontext:discountcart

Examines the current promotion list to determine whether there are any appropriate discount rules.

Syntax

```
<commercecontext:discountcart/>
```

Parameters

None.

Description

If there are discount rules, this tag applies the discount to the current shopping cart. If the promotion list does not yet exist, this tag invokes [commercecontext:getpromotions](#) to obtain the list.

commercecontext:discounttempcart

Examines the current promotion list to determine whether there are any appropriate discount rules.

Syntax

```
<commercecontext:discounttempcart  
  cart="cartname"/>
```

Parameters

cart (required)

Input parameter. Name of the cart to discount (not the current cart).

Description

If there are discount rules, this tag applies the discount to the shopping cart that is passed into the argument.

If the promotion list does not yet exist, this tag invokes [commercecontext:getpromotions](#) to obtain one.

The current cart is not discounted.

Use this tag to display a discounted price on a page. For example, one way to show that an item is currently reduced in price (that is, on sale) is to create a temporary cart, put that

item in that cart, use this tag to apply a special discount to the temporary cart, and then display the resulting price on the page next to text that announces the discount.

commercecontext:getpromotions

Creates the list of promotions for which the current visitor is eligible.

Syntax

```
<commercecontext:getpromotions
    listvarname="promotionlist"/>
```

Parameters

listvarname (required)

Input and output parameter. As input, name of the list to create on output of the promotions that the current visitor is eligible for.

Description

This tag is invoked whenever a method that needs to consult the promotion list is invoked and then finds that the promotion list does not yet exist. It is invoked by the following methods if they do not find the promotion list:

```
commercecontext:getrecommendations
commercecontext:getsinglerecommendation
commercecontext:discountcart
commercecontext:discounttempcart
```

To create the list of promotions, commercecontext:getpromotions invokes the [commercecontext:calculatepromotions](#) method.

The list is created unsorted; it can be used to display appropriate promotions for the current visitor. Use this tag during the debugging phase of your application design.

commercecontext:getratings

Calculates the ratings of the assets in a named list.

Syntax

```
<commercecontext:getratings
    assets="assetlist"
    listvarname="ratinglist"
    defaultrating="defaultrating"/>
```

Parameters

assets (required)

Input parameter. Name of the list of assets whose ratings you want to calculate. Its columns are assettype and assetid.

listvarname (required)

Input and output parameter. As input, name of the list to create on output that stores the assets with their ratings. Its columns are assettype, assetid, and rating. The

list of assets is created and ordered according to the segment ratings associated with the assets.

defaultrating (required)

Input parameter. Default rating to use for assets that do not have a rating assigned to them yet.

Description

This tag calculates ratings according to how important the asset is to this visitor, based on the segments to which the visitor belongs. This tag is invoked whenever you use any of these tags:

```
commercecontext:getrecommendations
commercecontext:getsinglerecommendation
assetset:establishratings
```

Use this tag during a debugging phase of your application development.

commercecontext:getrecommendations

Retrieves and lists the assets that match the recommendation constraints passed to the tag.

Syntax

```
<commercecontext:getrecommendation
  collection="recommendationname"
  [list="inputlist"]
  [value="rating"]
  [maxcount="assetcount"]
  listvarname="assetlist"
  [filter="true|false"]/>
```

Parameters

collection (required)

Input parameter. Name of the recommendation. The sort and selection criteria defined in the recommendation are used to create the list of possible assets. You can constrain this list by using the **maxcount** argument, defined below. If there are any promotions in place that override this recommendation, it substitutes the name of the promotion, instead.

list (optional)

Input parameter. name of the list of assets you want to be used as the input for the calculation. This argument is applicable only if the recommendation named by **collection** is an context-based recommendation. Columns are **assettype** and **assetid**.

value (optional)

Input parameter. Default rating for assets that do not have one. If you do not declare a value, unrated assets are assigned a default rating of 50 on a scale of 1-100.

maxcount (optional)

Input parameter. Maximum number of assets to return. Use this value to constrain the list of recommended assets.

listvarname (required)

Input and output parameter. As input, name you want to assign to the list of assets returned on output. Its columns are: `assettype` and `assetid`.

filter (optional)

Input parameter. `true` specifies that no assets in the input list can be returned as output; `false` (default) allows input assets to be returned as output.

Description

A call to this tag invokes the [commercecontext:getsegments](#) and [commercecontext:getpromotions](#) tags. For information about creating recommendations, including context-based recommendations, see the *CSEE Developer's Guide*.

commercecontext:getsegments

Retrieves the list of segments that the current visitor belongs to.

Syntax

```
<commercecontext:getsegments
    listvarname="segmentlist"/>
```

Parameters

listvarname (required)

Input and output parameter. As input, name of the list to create on output of segments that the visitor belongs to. It has one column named `assetid`.

Description

This tag invokes [commercecontext:calculatesegments](#) in order to generate the list. This tag is invoked whenever you use any of these tags:

```
commercecontext:getrecommendation
commercecontext:getrecommendations
commercecontext:calculatepromotions
commercecontext:getpromotion
commercecontext:getpromotions
```

Use this tag during the debugging phase of development.

commercecontext:getsinglerecommendation

Returns one recommended asset based on the recommendation criteria passed to the tag.

Syntax

```
<commercecontext:getsinglerecommendation
  collection="recommendationname"
  [list="inputlist"]
  [value="rating"]
  varname="assetid"
  typevarname="assettype"
  [filter="true|false"]/>
```

Parameters

collection (required)

Input parameter. Name of the recommendation. The sort and selection criteria defined in the recommendation are used to determine which asset to return. If there are any promotions in place that override this recommendation, it substitutes the name of the promotion, instead.

list (optional)

Input parameter. Name of the list of assets you want to be used as input for the calculation. This argument is only applicable if the recommendation named by **collection** is an context-based recommendation. Columns are **assettype** and **assetid**.

value (optional)

Input parameter. Default rating for assets that do not have one. If you do not declare a value, unrated assets are assigned a default rating of 50 on a scale of 1-100.

varname (required)

Input and output parameter. Name you want to assign to the variable that holds the ID of the asset that is returned on output.

typevarname (required)

Input and output parameter. Name you want to assign to the variable that holds the type of asset that is returned on output.

filter (optional)

Input parameter. A boolean value: **true** specifies that no assets in the input list can be returned as output; **false** (default) allows input assets to be returned as output.

Description

Use this tag to feature one product or to put one product on sale. This tag invokes the [commercecontext:getsegments](#) and [commercecontext:getpromotions](#) tags. For information about creating recommendations, including context-based recommendations, see the *CSEE Developer's Guide*.

commercecontext:setpromotioncutoff

Sets a value to compare against to determine promotion eligibility.

Syntax

```
<commercecontext:setpromotioncutoff  
    value="cutoff"/>
```

Parameters

value (required)

Input parameter. This number is compared against each promotion's rating to determine if the visitor is eligible for the promotion.

Description

This tag updates the promotion rating threshold for promotion eligibility. Prior to using the tag, the rating is 50 by default.

commercecontext:setsegmentcutoff

Sets a value to compare against to determine segment membership.

Syntax

```
<commercecontext:setsegmentcutoff  
    value="cutoff"/>
```

Parameters

value (required)

Input parameter. This number is compared against each segment's rating to determine whether the visitor is a member of the segment.

Description

This tag updates the segment rating threshold for segment membership. Prior to using the tag, the rating is 50 by default.

Visitor Data Manager Tags

The visitor data manager is the data access method that CS-Engage uses when processing rules. The visitor data manager enables you to gather, store, and retrieve visitor information from both the local, Content Server database as well as various external databases — Transact, for example.

When you create visitor attributes, history attributes, and history types for defining segments, the visitor data manager stores them in the local CS-Engage database. If you are using Transact, you can select Transact visitor attributes when defining segments as well. Then, when a method that determines which segments a visitor qualifies for is invoked, the visitor data manager checks the appropriate database for the needed visitor data. It provides a single access method to all of your visitor data. For information about creating visitor attributes, history attributes, and history types, see the *CSEE Developer's Guide*.

To maintain visitor session information after visitors end their sessions, you use persistent cookies. These cookies serve two purposes: to store the data and to mark the session with an identifier.

A visitor's Transact ID is also an identifier that can mark a visitor session. If a visitor performs an action on the site that requires him to log into Transact, his Transact ID is passed to his hard drive and identifies him as a valid user.

There are additional visitor data manager object tags available with CS-Engage.

Note

Each JSP file containing JSP visitor data manager tags must have the following directive at the top:

```
<%@ taglib prefix="vdm" uri="futuretense_cs/vdm.tld" %>
```

Use the [vdm:recordhistory](#) and [vdm:setscalar](#) tags to write visitor data to the CS-Engage database. Use the [vdm:recordhistory](#) tag to write a timestamped record of the data defined in a history type to the database. This data could include information from several history attributes, depending on the history type. Use the [vdm:setscalar](#) tag to write data for individual visitor attributes to the database.

Use the [get](#) tags to retrieve visitor data from either the local or external databases. Whenever one of the [commercecontext](#) tags invokes a [segment](#) tag, they retrieve this data — you do not have to use the [vdm:get](#) tags in those cases.)

Note

Because the visitor data manager does not track revisions you cannot create segments based on changes in visitor data. For example, two sessions ago a visitor changed his job description from “technical” to “financial.” When that change was written to the database, the value “financial” replaced the value “technical” and there is no record of his previous job description. Now, when segments based on job description are calculated, he qualifies only for the ones that include “financial” as an option.

vdm:gethistorycount

Retrieves the number of history records recorded for this visitor that match the specified criteria

Syntax

```
<vdm:gethistorycount
    attribute="attribute"
    varname="varname"
    [startdate="date1"]
    [enddate="date2"]
    [list="constraints"]/>
```

Parameters

attribute (required)

Input parameter. Name of a history type or name of an external history vector (that is, a vector of history data stored in the Transact database).

varname (required)

Input and output parameter. As input, name of a variable to create on output to hold the value of the count.

startdate (optional)

Input parameter. Start date in milliseconds from epoch.

enddate (optional)

Input parameter. End date in milliseconds from epoch.

list (optional)

Input parameter. Name of a list of values to use to constrain the count. The list has two columns:

- **field** – Name of an individual history attribute.
- **value** – Corresponding value for the history attribute.

Use it if you want to count only the instances of this record that match specific criteria. There might be more than one row for the same field.

Description

You can use this tag to count the number of times the visitor has viewed a specific advertisement and then display something on the current page based on that number.

vdm:gethistoryearliest

Retrieves the timestamp of the first time the specified history type was recorded for this visitor.

Syntax

```
<vdm:gethistoryearliest
  attribute="attribute"
  varname="varname"
  [startdate="date1"]
  [enddate="date2"]
  [list="constraints"] />
```

Parameters

attribute (required)

Input parameter. Name of a history type or name of an external history record (that is, a record of history data stored in the Transact database).

varname (required)

Input and output parameter. As input, name of a variable to create on output to hold the timestamp.

startdate (optional)

Input parameter. Start date in milliseconds from epoch.

enddate (optional)

Input parameter. End date in milliseconds from epoch.

list (optional)

Input parameter. Name of a list of values to use to constrain the date. The list has two columns:

- **field** – Name of an individual history attribute.
- **value** – Corresponding value for the history attribute.

Use this argument if you want to consider only specific instances of this record. There might be more than one row for the same field.

vdm:gethistorylatest

Retrieves the timestamp of the last time (that is, the most recent time) the specified history type was recorded for this visitor.

Syntax

```
<vdm:gethistorylatest
  attribute="attribute"
  varname="varname"
  [startdate="date1"]
  [enddate="date2"]
  [list="constraints"] />
```

Parameters

attribute (required)

Input parameter. Name of a history type or name of an external history record (that is, a record of history data stored in the Transact database).

varname (required)

Input and output parameter. As input, name of a variable to create on output to hold the timestamp.

startdate (optional)

Input parameter. Start date (in milliseconds from epoch).

enddate (optional)

Input parameter. End date (in milliseconds from epoch)

list (optional)

Input parameter. Name of a list of values to use to constrain the date. The list has two columns:

- **field** – Name of an individual history attribute.
- **value** – Corresponding value for the history attribute.

Use this argument if you want to consider only specific instances of this record. There might be more than one row for the same field.

vdm:gethistorysum

Sums the entries in a specific field for the specified history type.

Syntax

```
<vdm:gethistorysum
  attribute="attribute"
  field="fieldname"
  varname="varname"
  [startdate="date1"]
  [enddate="date2"]
  [list="constraints"]/>
```

Parameters

attribute (required)

Input parameter. Name of a history type or name of an external history vector (that is, a vector of history data stored in the Transact database).

field (required)

Input and output parameter. Name of the history attribute in the history type, or the vector attribute in the external history vector to sum.

varname (required)

Input and output parameter. As input, name of a variable to create on output to hold the sum.

startdate (optional)

Input parameter. Start date in milliseconds from epoch.

enddate (optional)

Input parameter. End date in milliseconds from epoch.

list (optional)

Input parameter. Name of a list of values to use to constrain the sum. The list has two columns:

- **field** – Name of an individual history attribute.
- **value** – Corresponding value for the history attribute.

Use this argument if you want to sum only the instances of this record that match specific criteria. There might be more than one row for the same field.

Description

You can use this tag to sum the amount of money a visitor spent during a period of time.

vdm:getscalar

Retrieves a specific single (scalar) visitor attribute for the current visitor.

Syntax

```
<vdm:getscalar  
    attribute="attribute"  
    varname="varname" />
```

Parameters

attribute (required)

Input parameter. Name of a visitor attribute or the name of an external scalar visitor data type; that is, a scalar value for visitor data stored in the Transact database.

varname (required)

Input and output parameter. As input, name of a variable to create on output for this attribute with the value for this visitor attribute.

Description

You can use this tag to display a personal message to the visitor based on this information.

vdm:loadscalarobject

Retrieves an object stored as a single (scalar) visitor attribute.

Syntax

```
<vdm:loadscalarobject  
    attribute="attribute"  
    varname="varname" />
```

Parameters

attribute (required)

Input parameter. Name of a visitor attribute.

varname (required)

Input and output parameter. As input, name of a variable to create on output to hold the object retrieved from the attribute.

See Also

[vdm:savescalarobject](#)

vdm:recordhistory

Saves a time-stamped copy of the specified history type with its current values for the visitor to the database.

Syntax

```
<vdm:recordhistory
    attribute="attribute"
    list="valuelist"/>
```

Parameters

attribute (required)

Input parameter. Name of a history type.

list (required)

Input parameter. Name of the list of values to record in the database. The list has two columns:

- **field** – Name of an individual history attribute.
- **value** – Corresponding value for the history attribute.

There is only one row for the same field.

Description

Use this tag to save the history data you have gathered about this visitor so that the segments can take it into consideration.

vdm:savescalarobject

Saves an object to the Content Server database as a visitor attribute.

Syntax

```
<vdm:savescalarobject
    attribute="attribute"
    object="objectname"/>
```

Parameters

attribute (required)

Input parameter. Name of the visitor attribute to save the object as.

object (required)

Input parameter. Name of the object to save as the visitor attribute. This object must implement either the standard Java interface, `Serializable`, or the Content Server implementation of the standard Java interface, `IStorableObject`.

Description

Use this tag to store a shopping cart across sessions.

Example

For example, using the CS-Engage visitor attribute form, you create a visitor attribute of type binary and name it “storedcart.”

You code your application pages to create a cart to be stored when the session ends:

```
<cart:create name="localcart"/>
```

Then you use this tag to store that cart object as the visitor attribute named “storedcart”:

```
<vdm:savescalarobject attribute="storedcart" object="localcart"/>
```

The next time the visitor arrives at the site, you can use `<vdm:loadscalarobject attribute="storedcart"/>` to retrieve the stored cart.

vdm:setscalar

Writes the data for a visitor attribute to the CS-Engage database.

Syntax

```
<vdm:setscalar
    attribute="attribute"
    value="value"/>
```

Parameters

attribute (required)

Input parameter. Name of the visitor attribute.

value (required)

Input and output parameter. As input, value to record for this visitor on output.

Description

Use this tag to save the visitor data you’ve gathered about the visitor so that the segments can take it into consideration.

Example

```
<!-- Write the registration information to the database.-->
<vdm:setscalar attribute="name" value= "vname"/>
<vdm:setscalar attribute="age" value= "vage"/>
<vdm:setscalar attribute="jobdesc" value= "vjobdesc"/>
<!-- where vname, vage, and vjobdesc are available either through
ics.GetVar, or by specifically declaring a String using JAVA
Scriptlets.-->
```

Chapter 13

CS-Satellite JSP Tags

As a page developer, you can control the following:

- What objects are cached on CS-Satellite
- How long each object is cached on CS-Satellite

You control CS-Satellite through JSP tags running on Content Server. That is, you do not place JSP tags into CS-Satellite itself; rather, you control CS-Satellite indirectly through Content Server. Before reading this chapter, you must be familiar with the Content Server JSP concepts described in the *dCS Developer's Guide*.

Note

Each JSP page containing CS-Satellite tags must have the following directive at the top:

```
<%@ taglib prefix="satellite" uri="futuretense_cs/satellite.tld" %>
```

The following section describes each CS-Satellite tag, beginning with [satellite:blob](#).

Alphabetical List of CS-Satellite JSP Tags

The following is an alphabetical list of CS-Satellite JSP tags:

satellite:blob

Gets a blob.

Syntax

```
<satellite:blob>
  [<satellite:parameter name="service" value="service_name"/>]
  [<satellite:parameter name="blobtable"
value="blob_table_name"/>]
  [<satellite:parameter name="blobkey" value="primary_key_name"/
>]
  [<satellite:parameter name="blobwhere"
value="primary_key_value"/>]
  [<satellite:parameter name="blobcol" value="column_name"/>]
  [<satellite:parameter name="blobheader" value="MIME_type"/>]
  [<satellite:parameter name="csblobid" value="session
variable"/>]
  <satellite:parameter name="cachecontrol"
                        value="expiration_date_time"/>
  [<satellite:parameter name="blobnocache"
                        value="true or false"/>]
  <satellite:parameter name="ALT" value="alttext"/>
  <satellite:parameter name="BORDER" value="border_size"/>
</satellite:blob>
```

Parameters

service (required)

Specify the start of the HTML tag to hold this blob. For example, if the blob is an image, set *service_name* to "img src".

blobtable (required)

Name of the content table containing the binary data.

blobkey (required)

Name of the column used as the table's primary key.

blobwhere (required)

Value of the primary key for the row containing the binary data.

blobcol (required)

Name of the column containing the binary data.

blobheader (required)

MIME type for returned data. It takes the form *description/extension*. For example, the MIME type for a GIF image is *image/gif*.

csblobid (optional)

The value of this parameter must correspond to a session variable of the same name when BlobServer security is on. When a request is made to the BlobServer, the *img src* service parameter sends a new request to the application server after the browser

reads the containing page. BlobServer then verifies if the `csblobid` parameter was set. When BlobServer security is set to true and the `csblobid` value matches the session variable value, the blob is served. If the values do not match a security violation is reported and no blob is served.

cachecontrol (optional)

Specifies when this blob expires from the cache. If you omit this tag, CS-Satellite uses the value of the `expiration` property to determine how long to cache this blob. You can override the default expiration value by specifying one of the following:

- `never` – This blob should never expire for time reasons. Such objects are not guaranteed to stay in the cache forever. For example, if the cache is full, CS-Satellite removes blobs on a LRU (Least Recently Used) basis.
- `immediate` – Expire this blob immediately; in other words, to not cache this page.
- `expiration_date_and_time` – Specifies the precise date(s) and time(s) this blob should expire.
- `session:expiration_date_and_time` – Specifies the precise date(s) and time(s) this blob should expire. Placing the word `session` in front of `expiration_date_and_time` tells CS-Satellite to serve this blob only to the person who originally requested it.

blobnocache (optional)

Whether or not to disable blob caching. A value of `true` prevents the blob from being cached.

ALT (optional)

Alternate text associated with the blob.

BORDER (optional)

Sets the border size for the blob.

Description

This tag retrieves a blob. If the specified blob is already cached on this CS-Satellite, `satellite:blob` loads it into the page. If the specified blob is not already cached on this CS-Satellite, `satellite:blob` extracts the blob from a table, caches it, and loads it into the page. Keep in mind that this tag works regardless of the cookie setting on the browser.

This tag overrides the default caching expiration time defined by the CS-Satellite `expiration` property.

The `satellite:blob` tag is the preferred method to render images, downloadable documents, and other large objects that are managed by Content Server.

Note

The `satellite:blob` tag functions correctly when cookies are disabled.

Many Web sites contain images, such as banners or corporate logos, that never expire or seldom expire. Such images might appear on every page of content and be served millions of times per day. If a blob never expires (or seldom expires), your first impulse might be to use the `satellite:blob` tag and set the `cachecontrol` attribute to `never`. This is an acceptable method of serving blobs, however, it does not provide the best performance. Whenever the cached blob is requested, Satellite must perform a lookup in the cache table.

Each individual lookup does not require many resources, but CS-Satellite might have to perform that same lookup millions of times per day.

An alternative way to serve never-expiring images is to do the following:

- Copy the never-expiring images to all your CS-Satellite hosts. Place them under the doc root for the Web server.
- Access them through `` HTML tags rather than through `satellite:blob` Satellite JSP tags.

For example, consider a never-expiring corporate logo file named `CorporateLogo.gif`. To use the alternate scheme, you first copy the file to the Web server's doc root on all your CS-Satellite hosts. Then, instead of serving this logo through a `satellite:blob` tag, your element could simply issue a tag like the following:

```

```

Note

Be careful when using this alternative mechanism for serving never-expiring images. For example, CS-Satellite cannot warn you that one of the CS-Satellite hosts does not contain the same image file as the other hosts.

You can use the `blobheadername` and `blobheadervalue` parameters to specify http headers in the BlobServer response. For example, you can specify the content-type as follows:

```
blobheadername1="Content-Type" blobheadervalue1="image/gif"
```

If you want to use http headers to prevent browser caching of BlobServer requests for all browsers, use both HTTP 1.0 and HTTP 1.1 cache control directives. In the following example, the first header (`Cache-Control:no-cache`) is the HTTP 1.1 directive and the second header (`Pragma:no-cache`) is the HTTP 1.0 directive.

```
blobheadername1="Cache-Control" blobheadervalue1="no-cache"
blobheadername2="Pragma" blobheadervalue2="no-cache"
```

To determine how long to cache a page or pagelet, these criteria are evaluated in this order:

1. A value passed in using the `cachecontrol` parameter.
2. If there is no value in `cachecontrol`, the value in the `satellite.page.cachecontrol.default` property, found in the `futuretense.ini` property file, is used.
3. If there is no value in `satellite.page.cachecontrol.default`, the value of the `expiration` property, located in the `satellite.ini` property file, is used.

divine recommends that you set values for the `satellite.page.cachecontrol.default` properties that make sense for the majority of your pages and pagelets. If you want to override the default expiration time, use the `cachecontrol` parameter. Use the `expiration` property to determine the timeout for pages that are not generated from this tag; for example, pages that are generated directly from a URL.

Example

```

<%
StringBuffer bf = new StringBuffer();

ics.SetVar("category", "logo");

IList list = ics.SelectTo("MyImageTable",
    "id,urlpicture,alttext",
    "category",
    null,
    1,
    "imagelist",
    true,
    bf);
%>

<satellite:blob>
    <satellite:parameter name='service' value='img src' />
    <satellite:parameter name='blobtable' value='MyImageTable' />
    <satellite:parameter name='blobkey' value='id' />
    <satellite:parameter name='blobwhere' value='imagelist.id' />
    <satellite:parameter name='blobcol' value='urlpicture' />
    <satellite:parameter name='blobheader' value='image/gif' />
    <satellite:parameter name='cachecontrol' value='*:30:0 */*/*' />
>
    <satellite:parameter name='ALT' value='imagelist.alttext' />
    <satellite:parameter name='BORDER' value='0' />
</satellite:blob>

```

satellite:cookie

Sets a cookie on the user's browser.

Syntax

```

<satellite:cookie>
    <satellite:parameter name='name' value='cookie_name' />
    <satellite:parameter name='value' value='cookie_value' />
    <satellite:parameter name='timeout' value='cookie_timeout' />
    <satellite:parameter name='secure' value='true|false' />
    <satellite:parameter name='url' value='url' />
</satellite:cookie>

```

Parameters

name (required)

The name of the cookie you want to set.

value (required)

The value of the cookie you want to set.

timeout (required)

Timeout value of the cookie in seconds.

secure (required)

Set to true to indicate a secure cookie; set to false to indicate a non-secure cookie.

url (required)

A valid URL. This value restricts the sending of the cookie from the client to this URL only. The path must include the servlet that set the cookie, for example, /catalog, which makes the cookie visible to all directories on the server under /catalog.

Description

The `satellite:cookie` tag sets a cookie on the user's browser.

HTML requires that serverside cookies must be set in the head section of a Web page. However, you don't have to put the `satellite:cookie` tag inside the head section of a Content Server element; you can place it anywhere within the element, because CS-Satellite causes the cookie to be set in a second request to the page. The `satellite:cookie` tag invokes the `CookieServer` to set the cookie.

Example

```
<satellite:cookie>
  <satellite:parameter name='name' value='mycookie_2' />
  <satellite:parameter name='value' value='mycookievalue_2' />
  <satellite:parameter name='timeout' value='6' />
  <satellite:parameter name='secure' value='false' />
  <satellite:parameter name='url' value='ContentServer/mysite/
frankspage' />
</satellite:cookie>
```

satellite:flush

Flushes individual items from the cache.

Syntax

```
<satellite:flush>
  [<satellite:server name="servername" username="username"
    password="password" />]
</satellite:flush>
```

Parameters**name (required)**

The name of the CS-Satellite whose cache you want to flush.

username (required)

The user name to log in to the specified CS-Satellite.

password (required)

The password to log in to the specified CS-Satellite.

Note

To avoid a JSP compile error, be sure that this parameter value is on **one** line (no line breaks or carriage returns).

Description

The `satellite:flush` tag flushes all the contents of one or more CS-Satellite caches. To flush individual components from the cache, use the CacheManger API. The CacheManager API provides a flush mechanism to Content Server and CS-Satellite.

Examples

The following example flushes a page and two pagelets from the holbein, vaneyck, and vermeer servers :

```
<satellite:flush>
  <satellite:server name="http://holbein:80" username="ftuser"
password="ftuser"/>
  <satellite:server name="http://vaneyck:80" username="ftuser"
password="ftuser"/>
  <satellite:server name="http://vermeer:80" username="ftuser"
password="ftuser"/>

  <satellite:pages pagetags="%3cpage pagename=%22xml/main%22/
page%3e%3cpage pagename=%22xml/pagelet1%22/page%3e%3cpage
pagename=%22xml/pagelet2%22/page%3e"/>
</satellite:flush>
```

The following example flushes a containing page and a blob from the holbein server. The page was accessed using the following URL:

```
http://holbein:80/servlet/Satellite/?pagename=xml/
main&name=thomasmore
```

Because the name argument was passed to the page, the code to flush the page from the cache contains the value of the name argument:

```
<satellite:flush>
  <satellite:server name="http://holbein:80" username="ftuser"
password="ftuser"/>
  <satellite:pages pagetags="%3cpage pagename=%22xml/main%22
name=%22thomasmore/page%3e%3cblob service=%22
image src%22blobtable=%22SmokeImage%22
blobwhere=%228765%22blobheader=%22image/gif%22
blobcol=%22urlpicture%22blobkey=%22id%22/blob%3e"/>
</satellite:flush>
```

satellite:link

Generates links.

Syntax

```
<satellite:link>
  [<satellite:parameter name="name1" value="value 1"/>]
  [<satellite:parameter name="nameN" value="value N"/>]
  [<satellite:parameter name="outstring"
    value="outstring info"/>]
  [<satellite:parameter name="classname"
    value="classname info"/>]
  [<satellite:parameter name="overridetype"
    value="override info"/>]
  [<satellite:parameter name="urlbase" value="some page name"/>]
  [<satellite:parameter name="pagename" value="some page name"/>]
</satellite:link>
```

Parameters

name1 (optional)

Name/value pair that modifies the query string.

nameN (optional)

Name/value pair that modifies the query string.

outstring (optional)

If this parameter is specified, no tag is returned and the primary attribute value is placed in the specified output variable. The default variable name is `referURL`.

classname (optional)

This parameter is used by the export mechanism to allow the class implementing the `IRreference` interface to be overridden. The default value is `COM.FutureTense.Export.DiskPageRef`.

overridetype (optional)

This parameter is used by the export mechanism to all the links to be hard coded as dynamic or static after the code is exported. The default is found in context; see the export code for more information.

urlbase (optional)

Specifies the entire URL up to the query string excluding the question mark (?). This parameter can be used to hard code another server or servlet. The default value is `ft.cgipath/ContentServer`.

pagename (optional)

Name of the page the link is for.

Description

This tag generates links for CS-Satellite. The links created by `satellite:link` are URL encoded. When the export code is used, this tag allows you to specify the pages to be exported. By default all the that are pages referenced are exported unless the `overridetype` parameter is used.

You can use this tag to link pages not rendered by Content Server by specifying the `urlbase` parameter and leaving the `pagename` blank.

satellite:page

Gets a pagelet and specifies its cache expiration.

Syntax

```
<satellite:page>
  [<satellite:parameter name='pagename' value='pagelet_name' />]
  <satellite:parameter name='cachecontrol' value='never' />
</satellite:page>
```

Parameters

pagename (required)

The name of a pagelet.

cachecontrol (optional)

Controls when this pagelet expires from the cache. If you omit this tag, CS-Satellite uses the value of the `expiration` property to determine how long to cache this pagelet. You can override the default expiration value by specifying one of the following:

- **never** — tells CS-Satellite that this pagelet should never expire for time reasons. Such objects are not guaranteed to stay in the cache forever. For example, if the cache is full, CS-Satellite still removes pagelets on a LRU (Least Recently Used) basis.
- **immediate** — tells CS-Satellite to expire this pagelet immediately; in other words, to not cache this page.
- **expiration_date_and_time** — tells CS-Satellite the precise date(s) and time(s) this pagelet should expire.
- **session:expiration_date_and_time** — tells CS-Satellite the precise date(s) and time(s) this pagelet should expire. Placing the word **session** in front of the expiration date and time tells CS-Satellite to serve this pagelet only to the current session. Typically, you specify **session:expiration_date_and_time** when the pagelet contains personalized data.

Description

The `satellite:page` tag is the CS-Satellite equivalent of the `CONTENTSERVER` tag or `EVALSERVER` tag. `satellite:page` gets a pagelet and specifies its cache expiration. If the specified pagelet is already cached on this CS-Satellite, `satellite:page` requests the page from Content Server, then loads it. If the specified pagelet is not already cached on this CS-Satellite, `satellite:page` caches it and loads it. This tag overrides the default caching expiration time defined by the CS-Satellite `expiration` property. Scoping for the `satellite:page` tag is set to `STACKED`. Cookies must be enabled for the `satellite:page` tag to function properly.

To determine how long to cache a page or pagelet, CS-Satellite evaluates these criteria in this order:

1. A value passed in using the `cachecontrol` parameter.

2. If there is no value in `cachecontrol`, the value in the `satellite.page.cachecontrol.default` property, found in the `futuretense.ini` property file, is used.
3. If there is no value in `satellite.page.cachecontrol.default`, the value of the expiration property, located in the `satellite.ini` property file, is used.

divine recommends that you set values for the `satellite.page.cachecontrol.default` properties that make sense for the majority of your pages and pagelets. If you want to override the default expiration time, use the `cachecontrol` parameter. Use the expiration property to determine the timeout for pages that are not generated from this tag; for example, pages that are generated directly from a URL.

Example

```
<satellite:page>
  <satellite:parameter name='pagename' value='example/jsp/
pagelet1' />
  <satellite:parameter name='cachecontrol' value='never' />
</satellite:page>
```

satellite:tag

Marks a region for use by CS-Satellite.

Syntax

```
<satellite:tag>
  <satellite:parameter [name= 'name'] value = 'open|closed' />
</satellite:tag>
```

Parameters

name (optional)

Specify zero or more valid name modifiers.

value (required)

Valid values are `open` and `closed`. A value of `open` marks the start of a pagelet CS-Satellite must parse. A value of `closed` marks the end of a pagelet CS-Satellite must parse.

Description

Tells the Satellite servlet that this pagelet contains a `satellite:page`, `satellite:link`, `satellite:cookie`, or `satellite:blob` tag. If this pagelet contains `satellite:page` or `satellite:blob` directives or any form or other form, then you must provide the `satellite:tag` pair. If you omit the pair, CS-Satellite streams the page without passing it.

If this pagelet does *not* contain `satellite:page` or `satellite:blob` directives, then you should omit the `satellite:tag` pair. The exception to this is if you are using `render:getpageurl`. If you mistakenly provide the `satellite:tag` pair, CS-Satellite still serves the page but performance may suffer.

If you do need the `satellite:tag` pair, we strongly recommend setting the “open” tag as the first JSP tag in your element and the “close” tag as the last JSP tag in your element, in between the fcs tags.

Example

```
<satellite:tag>  
  <satellite:parameter name='type' value='open' />  
</satellite:tag>
```


Chapter 14

Content Server SOAP Web Services

Content Server provides enhancements that permit the construction and streaming of SOAP service responses via the normal `pagename=evaluation` path. The following set of JSP tags enable you to construct soap messages using Content Server JSP elements.

This chapter contains the following sections:

- [SOAP Tags](#)
- [Web Services Tags](#)

SOAP Tags

These JSP tags construct SOAP messages using Content Server JSP elements. The following section describes each SOAP tag, beginning with [SOAP:ARGUMENT](#).

soap:argument

Defines arguments for the `soap:header` tag, which defines a simple soap header.

Syntax

```
<soap:header>
    [<soap:argument name="argName" value="argValue"/>]
</soap:header>
```

Parameter

`argument` (optional)

Arguments passed to SOAP as the page is called.

Description

The `soap:header` tag defines a simple SOAP header, which should contain at least one name/value pair or the tags should be omitted. The SOAP header and contained arguments are the first set of non-logic tags to be contained in the SOAP message. An error occurs if the SOAP header is in the wrong order.

errno

The possible values of `errno` include:

Value	Description

Example

The following example defines a header for a SOAP message.

```
<soap:header>
    <soap:argument name="DUMMY" value="VALUE"/>
</soap:header>
```

See Also

[soap:body](#)

[soap:fault](#)

[soap:header](#)

[soap:message](#)

soap:body

Indicates the main part of a SOAP message body

Syntax

```
<soap:body  
    [tagname="local_name"] >  
</soap:body>
```

Parameters

tagname (optional)

The first child element after the body has this local name. The default value is CS.

Description

The soap:body tag indicates the main part of a soap message body where user can stream data. The body tag must follow the soap header tag or an error results. The body can contain any legal JSP both in the form of literal data contained in the template and/or emitted by logic coded into the template.

Example

This example calls the element SoapTestOut.

```
<soap:body tagname="soapTestOut">  
    <CS xsi:type="soapresponse:CS">  
        <ics:listloop listname="RS" maxrows="5">  
            <headlines xsi:type="soapresponse:headlineStruct">  
                <id xsi:type="xsd:string">  
                    <ics:listget listname="RS" fieldname="id"/>  
                </id>  
                <headline xsi:type="xsd:string">  
                    <ics:listget listname="RS"  
                        fieldname="headline"/>  
                </headline>  
            </headlines>  
        </ics:listloop>  
    </CS>  
</soap:body>
```

See Also

[soap:header](#)

[soap:fault](#)

[soap:message](#)

soap:fault

Creates a status or error return message.

Syntax

```
<soap:fault
  code="error_code"
  string="error_as_string"
  [actor="cause_of_error"]
  [detail="error_details"] />
</soap:fault>
```

Parameters

code (required)

Code identifying the error. Default codes include:

- Version Mismatch
- MustUnderstand
- Client
- Server

string (required)

Error printed out as a string.

actor (optional)

Cause of the error.

detail (optional)

Detailed error information.

Description

SOAP messages can contain, at most, one status or error return message. This is called a soap fault element. Content Server does not check that more than one fault element is used. A fault element can be used for both error and status return.

The following table lists the default codes (example table taken from http://www.w3schools.com/soap/soap_errors.asp):

Error	Description
Version Mismatch	Invalid namespace for the SOAP Envelope element.
MustUnderstand	A child element of the Header element, with the mustUnderstand attribute set to "1", was not understood.
Client	The message was incorrectly formed or contained incorrect information.
Server	There was a problem with the server so the message could not proceed.

Example

This example code checks for an error, then displays the soap fault as the error message.

```
<IF COND="IsError.Variables.errno=true">
  <THEN>
    <soap:fault
      code="Server"
      string="Nothing to do!"
      actor="database"
      detail="Dummy"/>
  </THEN>
</IF>
```

See Also

[soap:body](#)

[soap:header](#)

[soap:message](#)

soap:header

Defines a simple soap header.

Syntax

```
<soap:header>
  [<soap:argument name="argName" value="argValue"/>]
</soap:header>
```

Parameter

argument (optional)

Arguments passed to SOAP as the page is called.

Description

The `soap:header` tag defines a simple SOAP header, which should contain at least one name/value pair, otherwise the tags should be omitted. The SOAP header and contained arguments are the first set of non-logic tags to be contained in the SOAP message. An error occurs if the SOAP header is in the wrong order.

Example

The following example defines a header for a SOAP message.

```
<soap:header>
  <soap:argument name="DUMMY" value="VALUE"/>
</soap:header>
```

See Also

[SOAP:ARGUMENT](#)

[soap:body](#)

[soap:fault](#)[soap:message](#)

soap:message

Processes the contents of an element. The element must exist in the ElementCatalog.

Syntax

```
<soap:message
  [ns="namespace prefix"]
  [uri="universal resource indicator"] >
</soap:message>
```

Parameters

ns (optional)
Namespace prefix.

uri (optional)
Universal resource indicator.

Description

The `soap:message` tag processes the contents of an element, which must exist in the ElementCatalog. This tag is used to define and surround the SOAP message and control the streaming of content contained within. Do not stream data, comments, or whitespace before or after this tag.

Example

```
<soap:message ns="soapresponse">
  <soap:header>
    <soap:argument name="DUMMY" value="VALUE"/>
  </soap:header>
  <soap:body tagname="soaptestOut">
    <CS xsi:type="soapresponse:CS">
      <ics:listloop listname="RS" maxrows="5">
        <headlines xsi:type="soapresponse:headlineStruct">
          <id xsi:type="xsd:string">
            <ics:listget listname="RS" fieldname="id"/>
          </id>
          <headline xsi:type="xsd:string">
            <ics:listget listname="RS"
              fieldname="headline"/>
          </headline>
        </headlines>
      </ics:listloop>
    </CS>
  </soap:body>
</soap:message>
```


See Also

[soap:body](#)

[soap:fault](#)

[soap:header](#)

Web Services Tags

These JSP tags construct SOAP messages using 'generic' Content Server JSP elements. The following section describes each web service tag, beginning with [webservices:invoke](#).

webservices:invoke

Processes the content of an element.

Syntax

```
<webservices:invoke
  wsdl="http://soapinterop.java.sun.com/round2/base?WSDL"
  target="http://soapinterop.org/"
  service="Round2Base"
  port="RIBaseIFPort"
  operation="echoString"
  object="echostring">
  <webservices:parameter
    type="string"
    value="hello world" />
</webservices:invoke>
```

Parameters

wsdl (optional)

Web Services Description Language (WSDL) defining the webservice

target (optional)

Target name space, as defined by the WSDL

service (optional)

A service, as defined by the WSDL, groups together a set of related ports. Together with port parameter, service specifies SOAP address

port (optional)

A port, as defined by the WSDL, is the combination of a binding and a network address. Together with service parameter, port specifies SOAP address.

operation (optional)

Function being invoked, which is supported by the web service.

object (optional)

Content Server object name. The returned object from the web service is stored here.

Description

Target, service, port, operation are all specified in the WSDL. See the W3 web site (<http://www.w3.org/TR/wsdl>) for more information.

Example

```
<webservices:invoke
  wsdl="http://soapinterop.java.sun.com/round2/base?WSDL"
  target="http://soapinterop.org/" service="Round2Base"
  port="RIBaseIFPort" operation="echoString"
```

```

        object="echostring"
        <web services:parameter
            type="string" value="hello world"/>
    </web services:invoke>
    echostring = <%=ics.GetObj("echostring")%>

```

See Also

[web services:parameter](#)

web services:parameter

Passes parameters to the [web services:invoke](#) tag. Must be used together with the [web services:invoke](#) tag.

Syntax

```

<web services:invoke ... />
    <web services:parameter
        type="string"
        value="hello world"/>
</web services:invoke>

```

Parameters

type (optional)
Name of parameter.

value (optional)
Value of parameter.

Description

The [web services:invoke](#) tag, along with the `web services:parameter` tag, processes the contents of an element, which must exist in the Element Catalog. These tags in conjunction are used to define and surround the SOAP message. They control the streaming of content contained within. Do not stream data, comments, or whitespace before or after these tags.

Example

```

<web services:invoke
    wsdl="http://soapinterop.java.sun.com/round2/base?WSDL"
    target="http://soapinterop.org/" service="Round2Base"
    port="RIBaseIFPort" operation="echoString"
    object="echostring" >
        <web services:parameter
            type="string" value="hello world"/>
    </web services:invoke>
    echostring = <%=ics.GetObj("echostring")%>

```

See Also

[web services:invoke](#)

Section 3

AssetMaker Utility: Tags and Overview

The following chapter in this section describes the AssetMaker tags. It also includes a brief overview of the AssetMaker utility and asset descriptor files:

- [Chapter 15, “AssetMaker XML Tags”](#) describes AssetMaker tags.

Chapter 15

AssetMaker XML Tags

AssetMaker is the Content Server Direct utility that creates new asset types using the basic asset model of one primary object table per asset type (as opposed to the flex asset model available with Content Server Direct Advantage).

To create a new basic asset type, you code an XML file called an asset descriptor file using the AssetMaker XML tags, and then you upload the file with the AssetMaker utility. This chapter defines the AssetMaker tags:

[ASSET](#)
[PROPERTIES](#)
[PROPERTY](#)
[STORAGE](#)
[INPUTFORM](#)
[SEARCHFORM](#)
[SEARCHRESULTS](#)

Note that there are no JSP versions of these tags.

This chapter also contains the following sections:

- [Overview: AssetMaker and Asset Descriptor Files](#)
- [Restrictions and Dependencies with TYPE](#)

Overview: AssetMaker and Asset Descriptor Files

The AssetMaker utility is located on the Admin tab in the CS-Direct product. AssetMaker creates new basic-style asset types. It does not create new flex-style asset types.

AssetMaker creates new asset types by uploading XML files called asset descriptor files. An asset descriptor file defines the asset type—the columns in the asset table and the fields in the forms that you use to create or edit a new asset of that type.

This is the general format of an asset descriptor file:

```
<ASSET NAME="assetTypeName" ...>
  <PROPERTIES>
    <PROPERTY...>
      <STORAGE.../>
      <INPUTFORM.../>
      <SEARCHFORM...>
      <SEARCHRESULTS.../>
    </PROPERTY>
    <PROPERTY...>
      <STORAGE.../>
      <INPUTFORM.../>
      <SEARCHFORM...>
      <SEARCHRESULTS.../>
    </PROPERTY>
  </PROPERTIES>
</ASSET>
```

The term “property” refers to both a column in a database table and the field that displays values from that column in the CS-Direct forms. An asset descriptor file must define several things about each of the properties for the asset type:

- The name and data type of the column that holds the values for the property.
- The name and format of the field when it is displayed in the CS-Direct **New** and **Edit** forms.
- Whether the field appears in the CS-Direct **Advanced Search** form and simple search for assets of this type.
- Whether the field appears in the CS-Direct **Search Results** form for assets of this type.

When AssetMaker creates an asset type, it does the following:

- Creates an object table in the Content Server database for assets of this type. The **STORAGE** clauses in the **PROPERTY** statements tell AssetMaker how to define the columns in the table. Additionally, there are several columns that AssetMaker creates by default.
- Creates elements that render the forms that you use in the CS-Direct product to create and maintain assets of this type. The **INPUTFORM**, **SEARCHFORM**, and **SEARCHRESULTS** clauses in the **PROPERTY** statements tell AssetMaker how to format the CS-Direct forms for those asset types.

If the Burlington Financial sample site is installed on your system, there are two example asset descriptor files in the AssetType table that you can examine: `ImageFile.xml` and `StyleSheet.xml`.

For more information about asset types and AssetMaker, see the *Developer's Guide*.

ASSET

The first tag in an asset descriptor file.

It names the new asset type and its database table and sets certain default values. The opening tag, `<ASSET>`, is always the first line of code in an asset descriptor file and the closing tag, `</ASSET>`, is always the last line of code.

There can be only one ASSET tag pair per asset descriptor file because you can create only one asset type per asset descriptor file.

Syntax

```
<ASSET
  NAME="assetTypeName"
  DESCRIPTION="assetTypeName"
  PROCESSOR= "4.0"
  [MARKERTEXT="*"]
  [MARKERIMAGE="/Xcelerate/data/something.gif"]
  DEFDIR= "x:\FutureTense\Storage\AssetTypeName">
  ...
</ASSET>
```

Parameters

NAME (required)

The internal name of the asset type. AssetMaker uses this string for the following items:

- As the name of the database table for assets of this type.
- As the name of the folder in the ElementCatalog table that holds the elements that AssetMaker creates to format the CS-Direct forms for working with assets of this type.

Because the string provided by this parameter becomes the name of the database table, you must use only the characters that your DBMS recognizes.

DESCRIPTION (required)

The external name of this asset type, that is, the name of this asset type when it is displayed in the CS-Direct forms and menus.

PROCESSOR (required)

The version number of AssetMaker at the time that you upload this asset descriptor file. The current version number is 4.

Earlier versions are used for asset types created in previous versions. The version number must be the same as the Version of Content Server when the asset type was created.

MARKERTEXT (optional)

A string of up to 32 characters that marks required fields for this asset type in the CS-Direct forms. (Fields are defined as required with the INPUTFORM attributes. See

[INPUTFORM](#) for more information.) You can use a text string only; you cannot use HTML or any other markup code in the string for MARKERTEXT.

This parameter is used in conjunction with the MARKERIMAGE argument, described next. If both MARKERTEXT and MARKERIMAGE are set, only the image specified by MARKERIMAGE is used in the CS-Direct forms.

If a field is defined as required but there is no value set for either MARKERTEXT or MARKERIMAGE, AssetMaker uses an asterisk (*) to mark the field as required.

MARKERIMAGE (optional)

An image file to use to mark required fields in the CS-Direct forms for assets of this type. The string defined for MARKERIMAGE must be a URL that refers to a .gif or a .jpg image file. The URL must point to a valid Web root directory that is configured on the Web server or to a directory relative to that root.

DEFDIR (optional)

The default storage directory for column data that is to be stored as an external file. In other words, the storage directory for any upload fields (URL column, URL field) in the asset type. All asset types must have DEFDIR defined.

For information about upload fields, see [INPUTFORM](#).

Example

This code fragment shows the ASSET statement for an asset type named ImageFile:

```
<ASSET NAME="ImageFile" DESCRIPTION="ImageFile"
MARKERIMAGE="/Xcelerate/data/required.gif" PROCESSOR="4.0"
DEFDIR="c:\FutureTense\Storage\ImageFile">
```

PROPERTIES

A pair of PROPERTIES tags marks the section in an asset descriptor file that holds all the PROPERTY statements. Nested within the PROPERTIES tags are the PROPERTY statements for each individual property.

There can be only one PROPERTIES tag pair per asset descriptor file because you can create only one asset type per asset descriptor file.

Syntax

```
<PROPERTIES>
    ...
</PROPERTIES>
```

Parameters

None.

PROPERTY

Instructs AssetMaker to create a new property and provides the name for the column in the table and the name for the field in the CS-Direct forms.

Syntax

```
<PROPERTY
  NAME="column name"
  DESCRIPTION="field name">
```

Parameters

NAME (required)

The internal name of the property, that is, the name of the column in the database table for this asset type.

If you use “URL” as the first three letters of the column name, Content Server recognizes it as a URL column (or upload field).

If you specify that this column is a URL column, Content Server stores the data entered for this field as a file in the external storage directory identified by the DEFDIR parameter for the ASSET tag.

DESCRIPTION (required)

The external name of the property, that is, the name of the field in the CS-Direct forms.

Description

Nested within each pair of PROPERTY tags are the [STORAGE](#), [INPUTFORM](#), [SEARCHFORM](#), and [SEARCHRESULTS](#) tags that define the data type that the column stores, the format of the field, and on which forms the field appears.

Example

This code fragment shows one PROPERTY statement, including its STORAGE, INPUTFORM, and SEARCHFORM statements:

```

<ASSET>
<PROPERTIES>
  <PROPERTY NAME="height" DESCRIPTION="Height">
    <STORAGE TYPE="INTEGER" LENGTH="4"/>
    <INPUTFORM TYPE="TEXT" WIDTH="4" MAXLENGTH="4" REQUIRED="NO"
      DEFAULT=""/>
    <SEARCHFORM DESCRIPTION="Height" TYPE="TEXT" WIDTH="4"
      MAXLENGTH="4" VERB=""/>
  </PROPERTY>
  ...
  ...
</PROPERTIES>
</ASSET>

```

STORAGE

This tag defines the data type of the column being defined for the property. A STORAGE statement is nested within a set of PROPERTY tags.

Syntax

```

<STORAGE
  TYPE="datatype"
  [LENGTH="length"]/>

```

Parameters

TYPE (required)

The data type of the column that this PROPERTY statement defines. You can specify any of the following generic ODBC/JDBC data types: CHAR, VARCHAR, SMALLINT, INTEGER, BIGINT, DOUBLE, TIMESTAMP.

These data types are defined by the following Content Server database properties located in the `futuretense.ini` file:

Type (generic ODBC/JDBC data type)	Property
CHAR	cc.char
VARCHAR	cc.varchar
SMALLINT	cc.smallint
INTEGER	cc.integer
BIGINT	cc.bigint
DOUBLE	cc.double
TIMESTAMP	cc.datetime

LENGTH (required if TYPE="VARCHAR")

The length of the field. Only applicable if the type is either VARCHAR or CHAR and required if the type is VARCHAR.

Example

This code fragment defines a column that can hold up to 255 characters:

```
<STORAGE TYPE="VARCHAR" LENGTH="255"/>
```

INPUTFORM

Specifies how data is placed in the edit and inspect forms for the property. An INPUTFORM statement is nested within a pair of PROPERTY tags and typically follows the STORAGE statement for the property. The input type for the field must be compatible with the data type of the database column, as specified by the STORAGE statement.

AssetMaker uses the INPUTFORM statements in the asset descriptor file when creating the following elements for the asset type:

- ContentForm.xml, which formats the CS-Direct **New** and **Edit** forms
- ContentDetails.xml, which formats the CS-Direct **Inspect** form

For more information about these elements, see the AssetMaker section in the *CSEE Developer's Guide*.

Syntax

The syntax for this tag varies based on the setting for TYPE.

TYPE="TEXT"

```
<INPUTFORM
  TYPE="TEXT"
  [REQUIRED="YES or NO"]
  [DEFAULT="default value"]
  [MAXLENGTH="length"]
  [INSTRUCTION="help for using the field"]
  [HIDDEN="YES or NO"]
  WIDTH="width"/>
```

TYPE="TEXTAREA"

```
<INPUTFORM
  TYPE="TEXTAREA"
  [REQUIRED="YES or NO"]
  [DEFAULT="default value"]
  [MAXLENGTH="length"]
  [INSTRUCTION="help for using the field"]
  [HIDDEN="YES or NO"]
  COLS="text box width"
  ROWS="text box height"/>
```

TYPE="UPLOAD"

```
<INPUTFORM
  TYPE="UPLOAD"
  [REQUIRED="YES or NO"]
  [DEFAULT="default value"]
  [MAXLENGTH="length"]
  [INSTRUCTION="help for using the field"]
  [HIDDEN="YES or NO"]
  WIDTH="width"
  LINKTEXT="text string"
  [MIMETYPE="file type"]/>
```

TYPE="SELECT"

```
<INPUTFORM
  TYPE="SELECT"
  [REQUIRED="YES or NO"]
  [DEFAULT="default value"]
  [INSTRUCTION="help for using the field"]
  [HIDDEN="YES or NO"]
  [MULTIPLE="YES or NO"]
  SOURCETYPE="STRING or TABLE"
```

The following must be specified for source type STRING:

```
OPTIONDESCRIPTIONS="xxx, yyy, zzz"
OPTIONVALUES="x, y, z"
```

The following must be specified for source type TABLE:

```
TABLERNAME="table name"
OPTIONDESCKEY="column name"
OPTIONVALUEKEY="column name"
[SQL="SQL statement"]/>
```

TYPE="CHECKBOX"

```
<INPUTFORM
  TYPE="CHECKBOX"
  [REQUIRED="YES or NO"]
  [DEFAULT="default value"]
  [INSTRUCTION="help for using the field"]
  [HIDDEN="YES or NO"]
  SOURCETYPE="STRING or TABLE"
```

The following must be specified for source type STRING:

```
CBDESCRIPTIONS="xxx, yyy, zzz"
CBVALUES="x, y, z"
```

The following must be specified for source type TABLE:

```
TABLERNAME="table name"
CBDESCKEY="column name"
CBVALUEKEY="column name"
[SQL="SQL statement"]/>
```

TYPE="RADIO"

```
<INPUTFORM
  TYPE="RADIO"
  [REQUIRED="YES or NO"]
  [DEFAULT="default value"]
  [INSTRUCTION="help for using the field"]
  [HIDDEN="YES or NO"]
  SOURCETYPE="STRING or TABLE"
```

The following must be specified for source type STRING:

```
RBDESCRIPTIONS="xxx, yyy, zzz"
RBVALUES="x, y, z"
```

The following must be specified for source type TABLE:

```
TABLENAME="table name"
RBDESCKEY="column name"
RBVALUEKEY="column name"
[SQL="SQL statement"]/>
```

TYPE="EWEBEDITPRO"

```
<INPUTFORM
  TYPE="EWEBEDITPRO"
  [REQUIRED="YES or NO"]
  [DEFAULT="default value"]
  [MAXLENGTH="length"]
  [INSTRUCTION="help for using the field"]
  [HIDDEN="YES or NO"]
  WIDTH="text box width"
  HEIGHT="text box height"/>
```

TYPE="ELEMENT"

```
<INPUTFORM
  TYPE="ELEMENT"
  [REQUIRED="YES|NO"]
  [DEFAULT="default value"]
  [INSTRUCTION="help for using the field"]
  WIDTH="text box width" />
```

Parameters**Parameters for All Types**

TYPE (required)

The input type of the field.

It must be set to one of the following values:

Input TYPE	Description
TEXT	A single line of text. Corresponds to the HTML input type named TEXT.
TEXTAREA	A text box, with scroll bars, that accepts multiple lines of text. Corresponds to the HTML input type named TEXTAREA. When you create a text box that displays the contents of a URL column (rather than using an upload field that displays the URL of a file), you must set specify a string for PROPERTY NAME that begins with the letters “ur l” and set the STORAGE TYPE to VARCHAR.
UPLOAD	A field that takes a file name and presents a Browse button so that you can either enter the name of a file or browse to it and select it. The content of an upload field is the URL of a file. The URL appears in the asset forms. When the user clicks Save, Content Server uploads the selected file and stores it in the file directory specified by the DEFDIR parameter in the ASSET tag. (See ASSET.) When you specify that a field is an upload field, set a string for PROPERTY NAME that begins with the letters “ur l” and set STORAGE TYPE (the property’s data type) to VARCHAR.
SELECT	A field that presents a drop-down list of options that can be selected. You can either specify the options that are presented in the list or you can specify a query so that the options are selected from the database and presented dynamically. Corresponds to the HTML input type SELECT.
CHECKBOX	A check box field. You can specify the names of the check box options or you can specify a query so that the names are selected from the database and presented dynamically. Corresponds to the HTML input type CHECKBOX.
RADIO	A radio button control. You can either specify the names of the radio options or you can specify a query so that the names are selected from the database and presented dynamically. Corresponds to the HTML input type RADIO.
EWEBEDITPRO	A field whose contents you edit by using the eWebEditPro HTML editor, a third-party tool from Ektron, Inc. Content Server Direct version 4.0 supports eWebEditPro version 2.1. When you specify that a field is an eWebEditPro field field, you must also set a string for PROPERTY NAME that begins with the letters “ur l” and set STORAGE TYPE (the property’s data type) to VARCHAR.

Input TYPE	Description
ELEMENT	A field to call a custom element during the display of the form. For ContentForm, the element is called from: OpenMarket/Xcelerate/AssetType/<i>assettype</i>/ ContentForm/<i>fieldname</i> For ContentDetails, the element is called from OpenMarket/Xcelerate/AssetType/<i>assettype</i>/ ContentDetails/<i>fieldname</i>

The value you set for INPUTFORM TYPE must be compatible with the STORAGE TYPE for this property. For more information, see [Restrictions and Dependencies with TYPE](#).

REQUIRED (optional)

Whether the field is a required field. To specify that the field is required, set REQUIRED to YES. When REQUIRED="YES", the field is marked in the CS-Direct forms by the value of either MARKERTEXT or MARKERIMAGE (see [ASSET](#)) and CS-Direct does not allow an asset of this type to be saved unless there is a legal value in this field.

If you do not provide a value for the REQUIRED parameter, the field is not required.

DEFAULT (optional)

The default value that appears in the field before a value has been entered or selected.

MAXLENGTH (optional)

This parameter is used only by TEXT, TEXTAREA, UPLOAD, and EWEBEDITPRO input types. The maximum number of characters that can be entered in this field. For upload or URL fields the value is the length of the file name. For TEXTAREA, Javascript is added to the form to ensure that the text entered does not exceed the designated MAXLENGTH. If this parameter is not specified, AssetMaker uses the LENGTH value that you specified with the STORAGE tag to determine how many values can be entered in the field.

Use this parameter when you want to limit users to entering fewer characters in the field than the column can actually hold. Not that you cannot set this value to be greater than the LENGTH specified by the STORAGE tag.

INSTRUCTION (optional)

A short help message of up to 80 alphanumeric characters that appears near the field in the CS-Direct **New** or **Edit** forms.

HIDDEN (optional)

Whether or not the field is to be kept hidden from the CS-Direct users. Set HIDDEN="YES" to specify that this field is a hidden field. Note that while a hidden field is not displayed on an **New** or **Edit** form, the value for it is displayed in the **Inspect** form.

TEXT Parameters

(TYPE="TEXT")

WIDTH (required)

The width of the field in characters when it is displayed in the form. Corresponds to the HTML attribute SIZE.

TEXTAREA Parameters

(TYPE= "TEXTAREA")

COLS (required)

The width of the text box in characters. For example, COLS="48". Corresponds to the HTML attribute COLS.

ROWS (required)

The height of the text box in lines. For example, ROWS="4".

Corresponds to the HTML attribute ROWS.

UPLOAD Parameters

(TYPE="UPLOAD")

WIDTH (required)

The width of the field in characters.

Corresponds to the HTML attribute SIZE.

MIMETYPE (optional)

The mimetype of the file when the field can hold files of one type only. For example, if the field holds PDF files only, set MIMETYPE="PDF".

If you do not specify the MIMETYPE for a field with an input type of UPLOAD, AssetMaker expects to find a separate mimetype field. You create a separate mimetype field when your UPLOAD field can hold files of more than one type.

For example, if the asset type is for image files and the UPLOAD field can hold either .jpg files or .gif files, code a separate PROPERTY statement in the asset descriptor file for a mimetype field that presents a drop-down list of file type options.

LINKTEXT (required)

A text string that AssetMaker uses to format the link to the file identified by this field on the **Inspect** form for the asset. For example, if you set LINKTEXT="Image", then, on the **Inspect** form for assets of this type, the link to file is named "Image."

SELECT Parameters

(TYPE="SELECT")

MULTIPLE (optional)

Whether or not the field accepts multiple values. When this parameter is not set, the field accepts one value only. If you set MULTIPLE="YES", the field can hold more than one value.

Note

When setting multiple YES values make sure that the storage field is long enough to handle all the possible selections.

SOURCETYPE (required)

The source of the options available in the drop-down list for this field.

- To specify the options, set SOURCETYPE to STRING and then use the next two parameters in this section (OPTIONDESCRIPTIONS and OPTIONVALUES).
- To specify a query that obtains the list dynamically from a database table, set SOURCETYPE to TABLE and then use the last four parameters in this section (TABLENAME, OPTIONDESCKEY, OPTIONVALUEKEY, and SQL).

OPTIONDESCRIPTIONS (required when SOURCETYPE="STRING")

A list of the options that appear in the drop-down list in the form, separated by commas. For example, `OPTIONDESCRIPTIONS="red,yellow,blue,green"`.

OPTIONVALUES (required when SOURCETYPE="STRING")

A list of the values that map to the options specified by the `OPTIONDESCRIPTIONS` parameter. In other words, you can specify that the options that appear in the drop-down list are "red,yellow,blue,green" but that the values that are written to the database are "r,y,b,g".

Note

If the data type of this field is numeric and you use the `OPTIONVALUES` parameter to create a string list for CS-Direct to use when populating the field, your list of values must be compatible with the way numeric values are stored in your DBMS.

For example, say that `STORAGE TYPE="DOUBLE"`, `INPUTFORM TYPE="SELECT"`, and `SOURCE="STRING"`. To specify that one of the values that can be entered in this field is 100,000, you specify "100000" as one of the `OPTIONVALUES` if your DBMS is SQLServer but you specify "100000.0000000000" if your DBMS is Oracle.

In other words, the format of the values in such a string list must match that of your DBMS. Generally speaking, using a static list of string values is not a good choice for numeric data types.

TABLERNAME (required when SOURCETYPE="TABLE")

The name of the Content Server table that the options and their corresponding values are to be found in. AssetMaker creates a SQL query for this field based on the values you set for `TABLERNAME`, `OPTIONDESCKEY`, and `OPTIONVALUEKEY`.

OPTIONDESCKEY (required when SOURCETYPE="TABLE")

The name of the column in the table identified by `TABLERNAME` that contains the options that appear in the drop-down list in the form.

OPTIONVALUEKEY (required when SOURCETYPE="TABLE")

The name of the column in the table identified by `TABLERNAME` that holds the values that are written to the database when the user selects an option from the drop-down list.

SQL (optional when SOURCETYPE="TABLE")

A SQL statement that overrides the SQL query that AssetMaker creates based on the values provided with the previous three parameters. Use the `SQL` parameter when you want to include additional clauses in the query.

The query is still based on the values provided for the previous three parameters which means that in addition to the additional clauses you include, the statement must specify the table specified by `TABLERNAME` as the source and it must select data from the two columns identified by `OPTIONDESCKEY` and `OPTIONVALUEKEY`.

CHECKBOX Parameters

(TYPE="CHECKBOX")

SOURCETYPE (required)

The source of the names listed next to the check boxes.

- To specify the names, set SOURCETYPE to STRING and then use the next two parameters (CBDESCRIPTIONS and CBVALUES).
- To specify a query that obtains the names dynamically from a database table, set SOURCETYPE to TABLE, and then use the last four parameters in this section (TABLENAME, CBDESCKEY, CBVALUEKEY, and SQL).

CBDESCRIPTIONS (required when SOURCETYPE="STRING")

A list of the names that appear next to the check boxes, in the order that you want them to appear, separated by commas. For example, CBDESCRIPTIONS="red,yellow,blue,green".

CBVALUES (required when SOURCETYPE="STRING")

A list of the values that map to the check box names specified by the CBDESCRIPTIONS parameter. In other words, you can specify that the names that appear next to the check boxes are "red,yellow,blue,green" but that the values that are written to the database are "r,y,b,g".

Note

If the data type of this field is numeric and you use the CBVALUES parameter to create a string list for CS-Direct to use as the check box names, your list of values must be compatible with the way numeric values are stored in your DBMS.

For example, say that STORAGE TYPE="DOUBLE", INPUTFORM TYPE="CHECKBOX", and SOURCE="STRING". To specify the value that can be entered in this field is 100,000, you specify "100000" as the value for OPTIONVALUES if your DBMS is SQLServer but you specify "100000.0000000000" if your DBMS is Oracle.

In other words, the format of the values in such a string list must match that of your DBMS. Generally speaking, using a static list of string values is not a good choice for numeric data types.

TABLENAME (required when SOURCETYPE="TABLE")

The name of the Content Server table that the check box names and their corresponding values are to be found in. AssetMaker creates a SQL query for this field based on the values you set for TABLENAME, CBDESCKEY, and CBVALUEKEY.

CBDESCKEY (required when SOURCETYPE="TABLE")

The name of the column in the table identified by TABLENAME that contains the names that appear next to the check boxes on the form.

CBVALUEKEY (required when SOURCETYPE="TABLE")

The name of the column in the table identified by TABLENAME that holds the values that are written to the database when the user selects the check box.

SQL (optional when SOURCETYPE="TABLE")

A SQL statement that overrides the SQL query that AssetMaker creates based on the values provided with the previous three parameters. Use the SQL parameter when you want to include additional clauses in the query.

The query is still based on the values provided for the previous three parameters which means that in addition to the additional clauses you include, the statement must specify the table specified by TABLENAME as the source and it must select data from the two columns identified by CBDESCKEY and CBVALUEKEY.

RADIO Parameters

(TYPE="RADIO")

SOURCETYPE (required)

The source of the names listed next to the radio buttons.

- To specify the names, set SOURCETYPE to STRING.
- To specify a query that obtains the names dynamically from a database table, set SOURCETYPE to TABLE. and then use the last four parameters in this section (TABLENAME, RBDESCKEY, RBVALUEKEY, and SQL).

RBDESCRIPTIONS (required when SOURCETYPE="STRING")

A list of the names that appear next to the radio options, in the order that you want them to appear, separated by commas. For example, RBDESCRIPTIONS="red, yellow, blue, green".

RBVALUES (required when SOURCETYPE="STRING")

A list of the values that map to the radio options specified by the RBDESCRIPTIONS parameter. In other words, you can specify that the options that appear next to the check boxes are "red, yellow, blue, green" but that the values that are written to the database are "r, y, b, g".

Note

If the data type of this field is numeric and you use the CBVALUES parameter to create a string list for CS-Direct to use as the check box names, your list of values must be compatible with the way numeric values are stored in your DBMS.

For example, say that STORAGE TYPE="DOUBLE", INPUTFORM TYPE="RADIO", and SOURCE="STRING". To specify the value that can be entered in this field is 100,000, you specify "100000" as the value for OPTIONVALUES if your DBMS is SQLServer but you specify "100000.0000000000" if your DBMS is Oracle.

In other words, the format of the values in such a string list must match that of your DBMS. Generally speaking, using a static list of string values is not a good choice for numeric data types.

TABLENAME (required when SOURCETYPE="TABLE")

The name of the Content Server table that the check box options and their corresponding values are to be found in. AssetMaker creates a SQL query for this field based on the values you set for TABLENAME, RBDESCKEY, and RBVALUEKEY.

RBDESCKEY (required when SOURCETYPE="TABLE")

The name of the column in the table identified by TABLENAME that contains the options that appear next to the radio buttons on the form.

RBVALUE (required when **SOURCETYPE**="TABLE")

The name of the column in the table identified by **TABLENAME** that holds the values that are written to the database when the user selects the radio button.

SQL (optional when **SOURCETYPE**="TABLE")

A SQL statement that overrides the SQL query that AssetMaker creates based on the values provided with the previous three parameters. Use the SQL parameter when you want to include additional clauses in the query.

The query is still based on the values provided for the previous three parameters which means that in addition to the additional clauses you include, the statement must specify the table specified by **TABLENAME** as the source and it must select data from the two columns identified by **RBDESCKEY** and **RBVALUEKEY**.

EWEBEDITPRO Parameters

(TYPE="EWEBEDITPRO")

WIDTH (required)

The width of the eWebEditPro field in characters. For example, **WIDTH**="300".

HEIGHT (required)

The height of the eWebEditPro field in lines. For example, **HEIGHT**="300".

Examples

This code fragment creates a text field named "Title":

```
<PROPERTY NAME="title" DESCRIPTION="Title">
  <STORAGE TYPE="VARCHAR" LENGTH="255"/>
  <INPUTFORM TYPE="TEXT" WIDTH="48" MAXLENGTH="44" REQUIRED="YES"
    DEFAULT="Enter the title here"/>
</PROPERTY>
```

This code fragment creates a text box field named "Author's Note":

```
<PROPERTY NAME="authorsnote" DESCRIPTION="Author's Note">
  <STORAGE TYPE="VARCHAR" LENGTH="125"/>
  <INPUTFORM TYPE="TEXTAREA" COLS="48" ROWS="2" MAXLENGTH="100"
    REQUIRED="YES" DEFAULT=""/>
</PROPERTY>
```

This code fragment creates an upload field named "Thumbnail File":

```
<PROPERTY NAME="urlthumbnail" DESCRIPTION="Thumbnail File">
  <STORAGE TYPE="VARCHAR" LENGTH="255"/>
  <INPUTFORM TYPE="UPLOAD" WIDTH="36" REQUIRED="NO"
    LINKTEXT="Image"/>
</PROPERTY>
```

This code fragment creates a select field named "Mimetype" with a drop-down list that uses a SQL query to obtain the values for the list:

```
<PROPERTY NAME="mimetype" DESCRIPTION="Mimetype">
  <STORAGE TYPE="VARCHAR" LENGTH="36"/>
  <INPUTFORM TYPE="SELECT" SOURCETYPE="TABLE"
    TABLENAME="mimetype" OPTIONDESCKEY="description"
    OPTIONVALUEKEY="mimetype" SQL="SELECT mimetype, description
    FROM mimetype WHERE keyword = 'stylesheet' AND isdefault =
    'y'"/>
</PROPERTY>
```

This code fragment creates a select field that accepts multiple values from a drop-down field named “Credit Card Type”:

```
<PROPERTY NAME="cardtype" DESCRIPTION="Credit Card Type">
  <STORAGE TYPE="VARCHAR" LENGTH="48"/>
  <INPUTFORM TYPE="SELECT" MULTIPLE="YES"
    SOURCETYPE="TABLE"TABLENAME="CreditCard"OPTIONVALUEKEY="ccvalue
    " OPTIONDESCKEY="ccdescription" />
</PROPERTY>
```

This code fragment creates a set of check boxes named “Ground,” “2nd Day,” and “Next Day”:

```
<PROPERTY NAME="shipping" DESCRIPTION="Shipping option">
  <STORAGE TYPE="VARCHAR" LENGTH= "36"/>
  <INPUTFORM TYPE="CHECKBOX" SOURCETYPE="STRING"
    CBDESCRIPTIONS="Ground, 2nd Day, Next Day" CBVALUES="0,1,2"/>
</PROPERTY>
```

This code fragment creates a set of a radio options “Ground,” “2nd Day,” and “Next Day”:

```
<PROPERTY NAME="shipping" DESCRIPTION="Shipping option">
  <STORAGE TYPE="VARCHAR" LENGTH= "36"/>
  <INPUTFORM TYPE="RADIO" SOURCETYPE="STRING"
    RBDESCRIPTIONS="Ground, 2nd Day, Next Day" RBVALUES="0,1,2"/>
</PROPERTY>
```

This code fragment creates an eWebEditPro field named “urlbody”:

```
<PROPERTY NAME="urlbody" DESCRIPTION="Body">
  <STORAGE TYPE="VARCHAR" LENGTH="500"/>
  <INPUTFORM TYPE="EWEBEDITPRO" WIDTH="300" HEIGHT="300"
    REQUIRED="YES" INSTRUCTION="Be concise! No more than 3
    paragraphs."/>
  <SEARCHFORM TYPE="TEXTAREA" DESCRIPTION="Body Contains:"/>
</PROPERTY>
```

SEARCHFORM

Specifies the format of the field (property) and how data is entered into it when it appears on the **Advanced Search** form. A SEARCHFORM statement is nested within a pair of PROPERTY tags and typically follows the INPUTFORM statement for the property.

If the PROPERTY section does not include a SEARCHFORM statement, the field does not appear on the **Advanced Search** section.

Typically the SEARCHFORM statement closely matches the INPUTFORM statement regarding the input type and where the options (if any) come from. However, the types do not have to match exactly (although they must be similar, compatible).

AssetMaker uses this SEARCHFORM statements in an asset descriptor file to create the SearchForm.xml, AppendSelectDetails.xml, and SelectSummary.sql files for this asset type. (For information about these elements, see the AssetMaker section in the CSEE Developer's Guide.)

You cannot include upload fields or check boxes on the search forms. You also cannot include select fields if they accept more than one value at a time.

Note

When the TYPE is "Table," the select does not appear on the SimpleSearch form, but it does appear in the Advanced search for the asset type.

Syntax

The syntax for this tag varies based on the setting for TYPE.

TYPE="TEXT"

```
<SEARCHFORM
  TYPE="TEXT"
  DESCRIPTION="NameOfProperty"
  MAXLENGTH="length"
  WIDTH="width"/>
```

TYPE="TEXTAREA"

```
<SEARCHFORM
  TYPE="TEXTAREA"
  DESCRIPTION="NameOfProperty"
  MAXLENGTH="length"
  COLS="width of text box"
  ROWS="height of text box"/>
```

TYPE="SELECT"

```
<SEARCHFORM
  TYPE="SELECT"
  DESCRIPTION="NameOfProperty"
  [VERB="SQL verb"]
  SOURCETYPE="STRING or TABLE"
  OPTIONDESCRIPTIONS="xxx, yyy, zzz"
  OPTIONVALUES="x, y, z"
```



```

    TABLENAME="table name"
    OPTIONDESCKEY="column name"
    OPTIONVALUEKEY="column name"
    [SQL="SQL statement"]/>

```

TYPE="RADIO"

```

<SEARCHFORM
  TYPE="RADIO"
  DESCRIPTION="NameOfProperty"
  [VERB="SQL verb"]
  SOURCETYPE="STRING or TABLE"
  RBDESCRIPTIONS="xxx,yyy,zzz"
  RBVALUES="x,y,z"
  TABLENAME="table name"
  RBDESCKEY="column name"
  RBVALUEKEY="column name"
  [SQL="SQL statement"]/>

```

TYPE="ELEMENT"

```

<SEARCHFORM
  TYPE="ELEMENT"
  WIDTH="width"
  REQUIRED="YES|NO"
  DEFAULT="default value"
  INSTRUCTION="help for the form" />

```

Parameters

Parameters For All Types

TYPE (required)

Input type of the field. It must be set to one of the following values: TEXT, TEXTAREA, SELECT, or RADIO. You cannot set it to UPLOAD or CHECKBOX. For definitions of these input types, see [INPUTFORM](#).

The TYPE for SEARCHFORM must be compatible with the TYPE for INPUTFORM. Typically, it is the same TYPE, but it does not have to be.

DESCRIPTION (required)

Name of the field. The DESCRIPTION for SEARCHFORM should either match or include the DESCRIPTION for PROPERTY. See [PROPERTY](#).

MAXLENGTH (optional)

Maximum number of characters that can be entered in this field. If this parameter is not specified, AssetMaker uses the LENGTH value that you specified with the STORAGE tag to determine how many values can be entered in the field.

Use this parameter when you want to limit users to entering fewer characters in the field than the column can actually hold. Not that you cannot set this value to be greater than the LENGTH specified by the STORAGE tag.

VERB (required when STORAGE TYPE is BIGINT, SMALLINT, INTEGER, DOUBLE, or TIMESTAMP)

Sets the SQL verb that evaluates the value entered in this search field. You can set VERB to any of the following values:

Setting for VERB	Description
VERB="="	is equal to
VERB="!="	is not equal to
VERB="<"	is < than
VERB=">"	is > than

For example, if you set VERB="<" and a user enters 4 in the field in the **Advanced Search** form, CS-Direct searches for assets whose value in that field is less than 4. If VERB is not specified, the SQL verb is "LIKE", by default.

TEXT Parameters

(TYPE="TEXT")

WIDTH (required)

The width of the field in characters when it is displayed in the form. Corresponds to the HTML attribute SIZE.

TEXTAREA Parameters

(TYPE="TEXTAREA")

COLS (required)

The width of the text box in characters. For example, COLS="48".

ROWS (required)

The height of the text box in lines. For example, ROWS="4".

Corresponds to the HTML attribute ROWS.

SELECT Parameters

(TYPE="SELECT")

SOURCETYPE (required)

The source of the options available in the drop-down list for this field.

- To specify the options, set SOURCETYPE to STRING and then use the next two parameters in this section (OPTIONDESCRIPTORS and OPTIONVALUES).
- To specify a query that obtains the list dynamically from a database table, set SOURCETYPE to TABLE and then use the last four parameters in this section (TABLENAME, OPTIONDESCKEY, OPTIONVALUEKEY, and SQL).

OPTIONDESCRIPTORS (required when SOURCETYPE="STRING")

A list of the options that appear in the drop-down list in the form, separated by commas. For example, OPTIONDESCRIPTORS="red,yellow,blue,green".

OPTIONVALUES (required when SOURCETYPE="STRING")

A list of the values that map to the options specified by the OPTIONDESCRIPTORS parameter. In other words, you can specify that the options that appear in the drop-down list are "red,yellow,blue,green" but that the values that are written to the database are "r,y,b,g".

Note

If the data type of this field is numeric and you use the `OPTIONVALUES` parameter to create a string list for CS-Direct to use when populating the field, your list of values must be compatible with the way numeric values are stored in your DBMS.

For example, say that `STORAGE TYPE="DOUBLE"`, `SEARCHFORM TYPE="SELECT"`, and `SOURCE="STRING"`. To specify that one of the values that can be entered in this field is 100,000, you specify "100000" as one of the `OPTIONVALUES` if your DBMS is SQLServer but you specify "100000.0000000000" if your DBMS is Oracle.

In other words, the format of the values in such a string list must match that of your DBMS. Generally speaking, using a static list of string values is not a good choice for a numeric fields.

TABLENAME (required when `SOURCETYPE="TABLE"`)

The name of the Content Server table that the options and their corresponding values are to be found in. AssetMaker creates a SQL query for this field based on the values you set for `TABLENAME`, `OPTIONDESCKEY`, and `OPTIONVALUEKEY`.

OPTIONDESCKEY (required when `SOURCETYPE="TABLE"`)

The name of the column in the table identified by `TABLENAME` that contains the options that appear in the drop-down list in the form.

OPTIONVALUEKEY (required when `SOURCETYPE="TABLE"`)

The name of the column in the table identified by `TABLENAME` that holds the values that are written to the database when the user selects an option from the drop-down list.

SQL (optional when `SOURCETYPE="TABLE"`)

A SQL statement that overrides the SQL query that AssetMaker creates based on the values provided with the previous three parameters. Use the `SQL` parameter when you want to include additional clauses in the query.

The query is still based on the values provided for the previous three parameters which means that in addition to the additional clauses you include, the statement must specify the table specified by `TABLENAME` as the source and it must select data from the two columns identified by `OPTIONDESCKEY` and `OPTIONVALUEKEY`.

RADIO Parameters

(TYPE="RADIO")

SOURCETYPE (required)

The source of the options available in the drop-down list for this field.

- To specify the options, set `SOURCETYPE` to `STRING` and then use the next two parameters in this section (`RBDESCRIPTIONS` and `RBVALUES`).
- To specify a query that obtains the list dynamically from a database table, set `SOURCETYPE` to `TABLE` and then use the last four parameters in this section (`TABLENAME`, `RBDESCKEY`, `RBVALUEKEY`, and `SQL`).

RBDESCRIPTIONS (required when `SOURCETYPE="STRING"`)

A list of the options that appear in next to the check boxes, in the order that you want them to appear, separated by commas. For example, `CBDESCRIPTIONS="red, yellow, blue, green"`.

RBVALUES (required when SOURCETYPE="STRING")

A list of the values that map to the check box options specified by the CBDESCRIPTIONS parameter. In other words, you can specify that the options that appear next to the check boxes are "red, yellow, blue, green" but that the values that are written to the database are "r, y, b, g".

Note

If the data type of this field is numeric and you use the CBVALUES parameter to create a string list for CS-Direct to use as the check box names, your list of values must be compatible with the way numeric values are stored in your DBMS.

For example, if STORAGE TYPE="DOUBLE" and you want one of the values that could be entered in this field to be 100,000, you specify "100000" if your DBMS is SQLServer but you specify "100000.0000000000" if your DBMS is Oracle.

In other words, the format of the values in such a string list must match that of your DBMS. Generally speaking, using a static list of string values is better suited for text fields than it is for numeric fields.

TABLERNAME (required when SOURCETYPE="TABLE")

The name of the Content Server table that the names of the radio and their corresponding values are to be found in. AssetMaker creates a SQL query for this field based on the values you set for TABLERNAME, RBDESCKEY, and RBVALUEKEY.

RBDESCKEY (required when SOURCETYPE="TABLE")

The name of the column in the table identified by TABLERNAME that contains the names that appear next to the radio buttons on the form.

RBVALUE (required when SOURCETYPE="TABLE")

The name of the column in the table identified by TABLERNAME that holds the values that are written to the database when the user selects the radio button.

SQL (optional when SOURCETYPE="TABLE")

A SQL statement that overrides the SQL query that AssetMaker creates based on the values provided with the previous three parameters. Use the SQL parameter when you want to include additional clauses in the query.

The query is still based on the values provided for the previous three parameters which means that in addition to the additional clauses you include, the statement must specify the table specified by TABLERNAME as the source and it must select data from the two columns identified by RBDESCKEY and RBVALUEKEY.

ELEMENT Parameters**(TYPE="ELEMENT")****INSTRUCTION** (required)

A short help message that appears near the field in the CS-Direct **New** or **Edit** forms. Format is yyyy-mm-dd hh:mm.

Example

This code fragment includes the text field named "Title" on the **Advanced Search** form for the asset type:

```
<PROPERTY NAME="title" DESCRIPTION="Title">
  <STORAGE TYPE="VARCHAR" LENGTH="255"/>
```

```

    <INPUTFORM TYPE="TEXT" WIDTH="48" MAXLENGTH="44" REQUIRED="YES"
    DEFAULT="Enter the title here"/>
    <SEARCHFORM TYPE="TEXT" DESCRIPTION="Title contains" WIDTH="48"
    MAXLENGTH="44"/>
</PROPERTY>

```

This code fragment includes the text box field named “Author’s Note” on the **Advanced Search** form for the asset type:

```

<PROPERTY NAME="authorsnote" DESCRIPTION="Author's Note">
    <STORAGE TYPE="VARCHAR" LENGTH="125"/>
    <INPUTFORM TYPE="TEXTAREA" COLS="48" ROWS="2" MAXLENGTH="100"
    REQUIRED="YES" DEFAULT=""/>
    <INPUTFORM TYPE="TEXTAREA" DESCRIPTION="Author's Note
    contains" COLS="48" ROWS="2" MAXLENGTH="100"/>
</PROPERTY>

```

This code fragment includes a select box named “Mimetype” on the **Advanced Search** form for the asset type. The field’s drop-down list is populated by a SQL query:

```

<PROPERTY NAME="mimetype" DESCRIPTION="Mimetype">
    <STORAGE TYPE="VARCHAR" LENGTH="36"/>
    <INPUTFORM TYPE="SELECT" SOURCETYPE="TABLE"
    TABLENAME="mimetype" OPTIONDESCKEY="description"
    OPTIONVALUEKEY="mimetype" SQL="SELECT mimetype, description
    FROM mimetype WHERE keyword = 'stylesheet' AND isdefault =
    'y'"/>
    <SEARCHFORM DESCRIPTION="Mimetype" TYPE="SELECT"
    SOURCETYPE="TABLE" TABLENAME="mimetype"
    OPTIONDESCKEY="description" OPTIONVALUEKEY="mimetype"
    SQL="SELECT mimetype, description FROM mimetype WHERE keyword =
    'stylesheet' AND isdefault = 'y'"/>
</PROPERTY>

```

This code fragment tells includes a set of check boxes named “Ground,” “Next Day,” and “2nd Day” on the **Advanced Search** form for the asset type:

```

<PROPERTY NAME="shipping" DESCRIPTION="Shipping option">
    <STORAGE TYPE="VARCHAR" LENGTH="36"/>
    <INPUTFORM TYPE="CHECKBOX" SOURCETYPE="STRING"
    CBDESCRIPTIONS="Ground, 2nd Day, Next Day" CBVALUES="0,1,2"/>
    <SEARCHFORM TYPE="CHECKBOX" DESCRIPTION="Shipping contains"
    SOURCETYPE="STRING" CBDESCRIPTIONS="Ground, 2nd Day, Next Day"
    CBVALUES="0,1,2"/>
</PROPERTY>

```

This code fragment includes a set of radio options named “Ground,” “2nd Day,” and “Next Day” on the **Advanced Search** form for the asset type:

```

<PROPERTY NAME="shipping" DESCRIPTION="Shipping option">
    <STORAGE TYPE="VARCHAR" LENGTH="36"/>
    <INPUTFORM TYPE="RADIO" SOURCETYPE="STRING"
    RBDESCRIPTIONS="Ground, 2nd Day, Next Day" RBVALUES="0,1,2"/>
    <INPUTFORM TYPE="RADIO" DESCRIPTION="Shipping contains"
    SOURCETYPE="STRING" RBDESCRIPTIONS="Ground, 2nd Day, Next Day"
    RBVALUES="0,1,2"/>
</PROPERTY>

```

This code fragment includes a numeric field named “Height” on the **Advanced Search** form for the asset type. Notice that the VERB argument is specified because the STORAGE TYPE is set to INTEGER:

```
<PROPERTY NAME="height" DESCRIPTION="Height">
  <STORAGE TYPE="INTEGER" LENGTH="4"/>
  <INPUTFORM TYPE="TEXT" WIDTH="4" MAXLENGTH="4" REQUIRED="NO"
    DEFAULT="" />
  <SEARCHFORM DESCRIPTION="Height" TYPE="TEXT" WIDTH="4"
    MAXLENGTH="4" VERB="="/>
</PROPERTY>
```

SEARCHRESULTS

Determines which fields (properties) are displayed on the Search Results form after a search is run. A SEARCHRESULTS statement is nested within a pair of PROPERTY tags and typically follows the SEARCHFORM statement for the property.

If the INCLUDE parameter is set to "FALSE" or there is no SEARCHRESULTS statement at all for a property, the value for the field is not displayed on the **Search Results** form.

AssetMaker uses the information from the SEARCHRESULTS statements in an asset descriptor file to create the Title.xml element for this asset type. For information about the elements that AssetMaker creates, see the AssetMaker section in the *Developer's Guide*.

Syntax

```
<SEARCHRESULT
  INCLUDE="TRUE"
  LINKTYPE="EDIT or DETAILS"/>
```

Parameters

INCLUDE (optional)

Whether to include a value for the field (property) on the Search Results form.

LINKTYPE (optional)

Whether to present the information returned for the field as a link to another CS-Direct form. Valid settings are either EDIT or DETAILS.

When you set LINKTYPE to EDIT, the value returned on the **Search Results** form is a link to the **Edit** form for the asset. When you set LINKTYPE to DETAILS, the value returned on the **Search Results** form is a link to the **Inspect** form for the asset. When you do not include LINKTYPE, the data is with no link.

Example

This code fragment creates a property whose field is displayed on the **Search Results** form as a link to the **Edit** form:

```
<PROPERTY NAME="title" DESCRIPTION="Title">
  <STORAGE TYPE="VARCHAR" LENGTH="255"/>
  <INPUTFORM TYPE="TEXT" WIDTH="48" MAXLENGTH="44" REQUIRED="YES"
    DEFAULT="Enter the title here"/>
  <SEARCHFORM TYPE="TEXT" DESCRIPTION="Title contains" WIDTH="48"
    MAXLENGTH="44"/>
  <SEARCHRESULTS INCLUDE="TRUE" LINKTYPE="EDIT"/>
</PROPERTY>
```

Restrictions and Dependencies with TYPE

The possible settings for your INPUTFORM TYPE and SEARCHFORM TYPE parameters are dependent on the data type you set for STORAGE TYPE. For example, you can set INPUTFORM TYPE to UPLOAD only when you set your STORAGE TYPE to VARCHAR.

This section describes the dependencies, restrictions, and issues associated with the two TYPES, based on the data type specified for STORAGE TYPE.

CHAR (STORAGE TYPE="CHAR")

INPUTFORM and SEARCHFORM TYPE	INPUTFORM issues	SEARCHFORM issues
TEXT	none	The search in the field uses the SQL verb "LIKE" when building its query.
TEXTAREA	If the field is a URL field (PROPERTY NAME begins with the letters "url"), the data type cannot be CHAR, it must be VARCHAR.	The search in the field uses the SQL verb "LIKE" when building its query.
UPLOAD	You cannot create an upload field if the data type is CHAR. You can create an upload field only if the data type is VARCHAR.	NA
SELECT	If multiple values are legal, they are concatenated into a delimited list when they are saved to the database. Be sure to set the field length large enough to store all of the values in the concatenated list, including the delimiters.	You cannot include a select field that accepts multiple values on the Advanced Search form. The search in the field uses the SQL verb "LIKE" by default when building its query, you may also specify "VERB".
CHECKBOXES	none	You cannot include check boxes in the Advanced Search form.
RADIO	none	The search in the field uses the SQL verb "LIKE" when building its query.
EWEBEDITPRO	You cannot create an eWebEditPro field if the data type is CHAR. You can create an eWebEditPro field only if the data type is VARCHAR.	not applicable

VARCHAR (STORAGE TYPE="VARCHAR")

INPUTFORM and SEARCHFORM TYPE	INPUTFORM issues	SEARCHFORM issues
TEXT	none	The search in the field uses the SQL verb "LIKE" as the default when building its query, you may also specify "VERB".
TEXTAREA	none	The search in the field uses the SQL verb "LIKE" as the default when building its query, you may also specify "VERB".
UPLOAD	UPLOAD is available only with the VARCHAR data type.	You cannot include upload fields in the Advanced Search form.
SELECT	If multiple values are legal, they are concatenated into a delimited list when they are saved to the database. Be sure to set the field length large enough to store all of the values in the concatenated list, including the delimiters.	You cannot include a select field that accepts multiple values on the Advanced Search form. The search in the field uses the SQL verb "LIKE" when building its query.
CHECKBOX	If multiple values are legal, they are concatenated into a delimited list when they are saved to the database. Be sure to set the field length large enough to store all of the values in the concatenated list, including the delimiters.	You cannot include check boxes in the Advanced Search form.
RADIO	none	The search in the field uses the SQL verb "LIKE" when building its query.
EWEBEDITPRO	You cannot create an eWebEditPro field if the data type is CHAR. You can create an eWebEditPro field only if the data type is VARCHAR.	not applicable

SMALLINT

INPUTFORM and SEARCHFORM TYPE	INPUTFORM issues	SEARCHFORM issues
TEXT	none	You must set the VERB attribute. See SEARCHFORM .
TEXTAREA	Not a good choice for a SMALLINT data type. Additionally, if the field is a URL field (PROPERTY NAME begins with the letters “url”), the data type cannot be SMALLINT, it must be VARCHAR.	You must set the VERB attribute. See SEARCHFORM .
UPLOAD	You cannot create an upload field if the data type is SMALLINT. You can only create an upload field if the data type is VARCHAR.	not applicable
SELECT	You cannot create a field that holds multiple values if the data type is SMALLINT.	You must set the VERB attribute. See SEARCHFORM .
CHECKBOX	You cannot create a field that holds multiple values if the data type is SMALLINT.	You must set the VERB attribute. See SEARCHFORM .
RADIO	none	You must set the VERB attribute. See SEARCHFORM .
EWEBEDITPRO	You cannot create an eWebEditPro field if the data type is SMALLINT. You can create an eWebEditPro field only if the data type is VARCHAR.	not applicable

INTEGER (STORAGE TYPE="INTEGER")

INPUTFORM and SEARCHFORM TYPE	INPUTFORM issues	SEARCHFORM issues
TEXT	none	You must set the VERB attribute. See SEARCHFORM .
TEXTAREA	Not a good choice for an INTEGER data type. Additionally, if the field is a URL field (PROPERTY NAME begins with the letters "url"), the data type cannot be INTEGER, it must be VARCHAR.	You must set the VERB attribute. See SEARCHFORM .
UPLOAD	You cannot create an upload field if the data type is INTEGER. You can only create an upload field if the data type is VARCHAR.	Not applicable.
SELECT	You cannot create a field that holds multiple values if the data type is INTEGER.	You must set the VERB attribute. See SEARCHFORM .
CHECKBOX	You cannot create a field that holds multiple values if the data type is INTEGER.	You must set the VERB attribute. See SEARCHFORM .
RADIO	none	You must set the VERB attribute. See SEARCHFORM .
EWEBEDITPRO	You cannot create an eWebEditPro field if the data type is INTEGER. You can create an eWebEditPro field only if the data type is VARCHAR.	not applicable

BIGINT (STORAGE TYPE="BIGINT")

INPUTFORM and SEARCHFORM TYPE	INPUTFORM issues	SEARCHFORM issues
TEXT	none	You must set the VERB attribute. See SEARCHFORM .
TEXTAREA	Not a good choice for a BIGINT data type.	You must set the VERB attribute. See SEARCHFORM .
UPLOAD	You cannot create an upload field if the data type is BIGINT. You can only create an upload field if the data type is VARCHAR.	not applicable
SELECT	You cannot create a field that holds multiple values if the data type is BIGINT.	You must set the VERB attribute. See SEARCHFORM .
CHECKBOX	You cannot create a field that holds multiple values if the data type is BIGINT.	You must set the VERB attribute. See SEARCHFORM .
RADIO	none	You must set the VERB attribute. See SEARCHFORM .
EWEBEDITPRO	You cannot create an eWebEditPro field if the data type is BIGINT. You can create an eWebEditPro field only if the data type is VARCHAR.	not applicable

DOUBLE (STORAGE TYPE="DOUBLE")

INPUTFORM and SEARCHFORM TYPE	INPUTFORM issues	SEARCHFORM issues
TEXT	none	You must set the VERB attribute. See SEARCHFORM .
TEXTAREA	Not a good choice for the DOUBLE data type. Additionally, if the field is a URL field (PROPERTY NAME begins with the letters "url"), the data type cannot be DOUBLE, it must be VARCHAR.	You must set the VERB attribute. See SEARCHFORM .
UPLOAD	You cannot create an upload field if the data type is DOUBLE. You can only create an upload field if the data type is VARCHAR.	You must set the VERB attribute. See SEARCHFORM .
SELECT	You cannot create a field that holds multiple values if the data type is DOUBLE.	You must set the VERB attribute. See SEARCHFORM .
CHECKBOX	You cannot create a field that holds multiple values if the data type is DOUBLE.	You must set the VERB attribute. See SEARCHFORM .
RADIO	none	You must set the VERB attribute. See SEARCHFORM .
EWEBEDITPRO	You cannot create an eWebEditPro field if the data type is DOUBLE. You can create an eWebEditPro field only if the data type is VARCHAR.	not applicable

TIMESTAMP (STORAGE TYPE="TIMESTAMP")

INPUTFORM and SEARCHFORM TYPE	INPUTFORM issues	SEARCHFORM issues
TEXT	none	You must set the VERB attribute. See SEARCHFORM .
TEXTAREA	Not a good choice for a TIMESTAMP data type. Additionally, if the field is a URL field (PROPERTY NAME begins with the letters "url"), the data type cannot be TIMESTAMP, it must be VARCHAR.	You must set the VERB attribute. See SEARCHFORM .
UPLOAD	You cannot create an upload field if the data type is TIMESTAMP. You can only create an upload field if the data type is VARCHAR.	not applicable
SELECT	Not a good choice for a TIMESTAMP data type.	You must set the VERB attribute. See SEARCHFORM .
CHECKBOX	Not a good choice for a TIMESTAMP data type.	You must set the VERB attribute. See SEARCHFORM .
RADIO	Not a good choice for a TIMESTAMP data type.	You must set the VERB attribute. See SEARCHFORM .
EWEBEDITPRO	You cannot create an eWebEditPro field if the data type is TIMESTAMP. You can create an eWebEditPro field only if the data type is VARCHAR.	not applicable

Section 4

CS-Direct Advantage Parameters

The following chapters in this section describe CS-Direct Advantage parameters:

- [Chapter 16, “CS-Direct Advantage Cart Parameters”](#)
- [Chapter 17, “CS-Direct Advantage Order Item Parameters”](#)
- [Chapter 18, “CS-Direct Advantage Buyer Parameters”](#)

Chapter 16

CS-Direct Advantage Cart Parameters

Your shopping cart interface includes two types of parameters:

- **Order parameters** describe the entire order. For example, the order parameter named `totalSalesAmount` holds the total cost of the entire order.
- **Order item parameters** describe a single order item. For example, the order item parameter `UniqueID` holds the SKU of the item being sold.

This chapter describes in detail all shopping cart order parameters. For information about order item parameters, see [Chapter 17, “CS-Direct Advantage Order Item Parameters”](#)

The terms registered buyer and walk-in buyer appear throughout this chapter. Registered buyers have already set up accounts that identify their names, billing addresses, and payment brands. Registered buyers identify themselves to CS-Direct Advantage, which retrieves their account information, and then passes it on via the Commerce Connector to Transact for processing. By contrast, walk-in buyers do not have accounts set up with Transact. Therefore, walk-in buyers must provide all name, billing address, and payment brand information.

Note

The Transact Order Entry API (OEAPI) documentation uses the term *attribute* when referring to a shopping cart argument. This document uses the term *parameter* instead to avoid confusion with an *attribute*, which is an asset that describes other assets in the catalog.

To set a parameter, call [CART.SETPARAMETERS](#).

To get the value of a single parameter, call [CART.GETPARAMETER](#). To get the values of a list of parameters, call [CART.GETPARAMETERS](#).

For more information about shopping cart tags, see [Chapter 3, “CS-Direct Advantage XML Tags.”](#)

Note

If you call [CART.SETPARAMETERS](#) to set the value of an *informational* parameter, the Transact system that processes cart transactions may erase that value the next time you call either [CART.GETPARAMETER](#) or [CART.GETPARAMETERS](#).

Order Parameters

Each order parameter describes a characteristic of an order. Most, but not all, of the order parameters contain information that ordinarily appears in your order form.

All order parameters fall into one of these categories:

- *Required*, which you must set.
- *Optional*, which you may or may not set, but might provide additional useful information to Transact.
- *Informational*, which are set by Commerce Connector or Transact.

Names are case-sensitive. Values, including numeric quantities, are strings for both input and output.

Some shopping cart parameters are composite names that consist of a field name concatenated with a hyphen and a secondary field. Those parameters with the same field name are stored within a common hierarchical object.

The following section describes each order parameter, beginning with

Required Cart Parameters

The following shopping cart parameters are required for successful commit or check operations.

Note

If you set any of the `billToAddress-*` parameter fields, then you must set all of the `billToAddress-*` fields.

billToAddress-name1

Required for walk-in buyers. Also required for registered buyers who want to use a payment brand other than one registered in their account. Informational (and read-only) for registered buyers who plan on using a registered payment account.

Holds the name of the buyer to be used in the billing address.

billToAddress-name2

Optional for walk-in buyers. Informational for registered buyers.

This can hold additional name information, such as the buyer's first name or title (for example, *Chief of Surgery*).

Note

The Transact configuration determines whether `billToAddress-name2` is required for extra name information. If not required, you should write the whole name into `billToAddress-name1`. (By default, Transact does not require the `billToAddress-name2` field.)

billToAddress-address1

Required for walk-in buyers. Also required for registered buyers who want to use a payment brand other than one registered in their account. Informational (and read-only) for registered buyers who plan on using a registered payment account.

Holds the street address to be used in the billing address.

billToAddress-address2

Optional for walk-in buyers. Informational for registered buyers. Informational (and read-only) for registered buyers who use a registered payment account.

Holds additional street address information (for example, an apartment number or office number) to be used in the billing address.

billToAddress-city

Required for walk-in buyers. Also required for registered buyers who want to use a payment brand other than one registered in their account. Informational (and read-only) for registered buyers who plan on using a registered payment account.

Holds the city name to be used in the billing address.

Note

In case of ambiguity, Transact returns legal values after check or commit (USA only).

billToAddress-state

Each Transact site determines whether this parameter is optional or required for walk-in buyers (and for registered users who want to use a payment brand other than the registered one). Informational (and read-only) for registered buyers who plan on using a registered payment account.

Holds the state or province to be used in the billing address.

billToAddress-postalCode

Required for walk-in buyers. Also required for registered buyers who want to use a payment brand other than one registered in their account. Informational (and read-only) for registered buyers who plan on using a registered payment account.

Holds the postal code to be used in the billing address.

Note

In case of ambiguity, Transact returns legal values after check or commit (USA only).

billToAddress-country

Required for walk-in buyers. Also required for registered buyers who want to use a payment brand other than one registered in their account. Informational for registered buyers. Informational (and read-only) for registered buyers who plan on using a registered payment account.

Holds the two-letter country code to be used in the billing address. For example, the two-letter country code for the United States is "US".

billToAddress-county

Optional for walk-in buyers. Informational (and read-only) for registered buyers.

Holds the county name to be used in the billing address. The county name is occasionally useful in resolving ambiguities in addresses.

Note

Not required outside of the USA, and not required within the USA, unless the combined `billToAddress-city`, `-state`, and `-postalCode` parameters do not uniquely identify the county.

In case of ambiguity, Transact returns legal values after check or commit.

billToAddress-emailAddress

Required for walk-in buyers. Also required for registered buyers who want to use a payment brand other than one registered in their account. Informational for registered buyers who plan on using a registered payment account.

Holds the e-mail address to be used in the billing address.

billToAddress-telephone

Optional for walk-in buyers. Informational for registered buyers.

Holds the telephone number to be used in the billing address.

paymentInfo-brandID

This is the Transact GUID of a specific payment brand such as Visa or MasterCard. When entering the order with the `CART.COMMIT` tag, your application must provide either of the following parameters:

- `paymentInfo-accountID`
- Both `paymentInfo-brandID` and `paymentInfo-instrument` (`paymentInfo-instrument` only if the payment brand is a purchase order).

`paymentInfo-brandID` is an identifier for this payment brand. When your application calls `CART.CHECK`, Transact returns a list of payment brand identifiers (and their associated names) for the specified store. To get this list of identifiers, call `CART.GETLEGALVALUES` with `FIELD="paymentInfo-brandID"`.

Note

Transact always returns legal values after check or commit.

paymentInfo-ccn

The collection of digits (or possibly a combination of digits and letters) that appears on the card. `paymentInfo-ccn` is typically 12-18 digits long.

paymentInfo-expirationDate

The expiration date of the card, formatted as MM/YY. For example, a July 1999 expiration date is represented as 07/99, and a November 2000 expiration date is represented as 11/00.

paymentInfo-accountID

This is the Transact GUID of a payment instrument previously stored for a registered buyer. If set, `paymentInfo-brandID`, `-ccn`, and `-expirationDate` are ignored. The buyer's ID or GUID must be contained within the buyer parameter.

To get this list of identifiers, call `CART.GETLEGALVALUES` with the `FIELD` argument set to `paymentInfo-accountID`.

Note

Transact always returns legal values after check or commit.

Buyer Parameters

A record that identifies a visitor previously registered with Transact as a buyer.

For registered buyers, one of the following conditions is required for identification:

- `buyer-userID`
- Both `buyer-accessName` and `buyer-password`

If neither condition is set, Transact assumes the order is for a walk-in buyer.

buyer-userID

The identifier of this user (also returned by `VDM.GETCOMMERCEID`).

buyer-accessName

The name that this registered buyer has established for his account.

buyer-password

The password that this registered buyer has established for his account.

Optional Cart Parameters

The following shopping cart parameters are required for successful commit or check operations.

Note

If you do not set any of the `shipToAddress-*` parameters, then all `shipToAddress-*` parameters default to the corresponding `billToAddress-*` parameters. If you set any of the `shipToAddress-*` parameters, then you must set all of the `shipToAddress-*` parameters.

shipToAddress-name1

Required for orders with physical goods. Optional otherwise.

Holds the recipient's name, used in the shipping address.

shipToAddress-name2

Optional for walk-in buyers. Informational for registered buyers.

Holds extra name information, such as the buyer's title.

Note

(By default, Transact does not require the shipToAddress-name2 field.

shipToAddress-address1

Required for orders with physical goods, optional otherwise.

This parameter holds the recipient's street address.

shipToAddress-address2

Optional. This parameter holds additional street address information, such as the recipient's apartment number or office number.

shipToAddress-city

Required for orders with physical goods. Optional for digital goods.

This parameter holds the city name to be used in the recipient's address.

shipToAddress-state

Optional. This depends on how the site is configured.

Holds the recipient's state or province.

shipToAddress-postalCode

Required for orders with physical goods. Optional otherwise.

Holds the recipient's postal code.

Note

In case of ambiguity, Transact returns legal values after check or commit.

shipToAddress-country

Required for orders with physical goods. Optional for digital goods.

This parameter holds the two-letter country code (for example, US, CA, JP) to be used in the recipient's address.

shipToAddress-county

Optional. Holds the recipient's county name.

shipToAddress-emailAddress

Optional. Holds the recipient's e-mail address.

shipToAddress-telephone

Required for orders with physical goods. Optional otherwise.

Holds the recipient's telephone number.

Parameters Returned by Checking Carts

Transact always returns the following values after [CART.CHECK](#) operations.

Note

The [CART.COMMIT](#) tag also performs a check operation, and returns the following values, on a cart that has not previously been checked.

subTotal

The numeric cost of items (no currency symbols). This is the same value calculated by the tag [CART.GETPRELIMINARYTOTAL](#).

shippingInfo-shippingCost

The numeric cost of shipping (no currency symbols). This is the same value returned by the tag [CART.GETSHIPPINGTOTAL](#).

shippingInfo-shipperID

Not required unless the cart contains `Type=tangible` items. This is the Transact GUID of a shipper specified by the store's seller account at store configuration time. Legal values can be obtained after check or commit.

When you call [CART.CHECK](#), Transact sets `shippingInfo-shipperID` to the ID of the default shipper. You can either keep this value or you can set a different ID.

To get a list of potential IDs (and the names of the shippers), To get this list of identifiers, call [CART.GETLEGALVALUES](#) with `FIELD="shippingInfo-shipperID"`.

shippingInfo-shipperName

Not required unless the cart contains `Type=tangible` items. This is the Transact name of a shipper specified by the store's seller account at store configuration time. Legal values can be obtained after check or commit.

shippingInfo-shippingTax

A separate break-out of the tax specifically on `shippingCost`. `tax-taxAmount` already includes `shippingTax-taxAmount`, including the following sub fields:

`shippingInfo-shippingTax-taxType` – Type of tax: `USSalesTax`, `Canadian`, `VAT`

`shippingInfo-shippingTax-taxAmount` – if `taxType` is not `Canadian`

`shippingInfo-shippingTax-GST` – `Canadian GST`

`shippingInfo-shippingTax-PST` – `Canadian PST`

tax

Total tax on the order. `tax-taxAmount` already includes `shippingTax-taxAmount`. Includes the following subFields:

- `tax-taxType` – Type of tax: `USSalesTax`, `Canadian`, `VAT`
- `tax-taxAmount` – This is the same value returned by the tag `CART.GETTAXTOTAL` (if `taxType` is not `Canadian`).
- `tax-GST` – `Canadian GST`
- `tax-PST` – `Canadian PST`

totalSalesAmount

The total cost of the order including shipping and tax (no currency symbols). This is the sum of `subTotal`, `shippingCost`, and `tax-taxAmount`. This is the same value returned by the tag `CART.GETFINALTOTAL`.

store

A set of parameters that usually come from the `transact.ini` property file. (Usually, the Web developer is only concerned with `storeID` and `currencyCode`.)

<code>store-storeID</code>	<code>store-catalogUrl</code>
<code>store-currencyCode</code>	<code>store-receiptKeyFile</code>
<code>store-credentials</code>	<code>store-fulfillmentScheme</code>
<code>store-offerKeyFile</code>	<code>store-fulfillmentServer</code>
<code>store-catalogScheme</code>	<code>store-fulfillmentPort</code>
<code>store-catalogServer</code>	<code>store-fulfillmentUrl</code>
<code>store-catalogPort</code>	

Other Returned Parameters

As described in the preceding sections, Transact also returns the following values after [CART.CHECK](#) operations.

Note

The [CART.COMMIT](#) tag also performs a check operation, and returns the following values, on a cart that has not previously been checked.

Returned in case of ambiguity:

- billToAddress-city
- billToAddress-postalCode
- billToAddress-county
- shipToAddress-postalCode

Returned always:

- paymentInfo-brandID
- paymentInfo-accountID

Parameters Returned by Committing Carts

In addition to the values in the preceding section returned by check operations, Transact returns the following values after successful [CART.COMMIT](#) operations.

orderDate

A Java timestamp indicating the time the order was entered, in number of milliseconds since the epoch.

Note

Use the [DATE.CLOCKLIST](#) tag to convert the returned epoch format long integer into one of several available date/time fields and formats.

orderID

GUID which uniquely identifies the order on Transact. Value returned by [CART.GETTRANSACTIONID](#).

orderNumber

Integer which uniquely identifies the order within the specific store.

smartReceipt

URL of the Transact Smart Receipt page.

Chapter 17

CS-Direct Advantage Order Item Parameters

This chapter describes the order item parameters in your shopping cart. An order item holds information about one item to be purchased. **Order item parameters** describe the characteristics of a single order item.

All order item parameters fall into one of these categories:

- Required, which you must set.
- Optional, which you may or may not set, but might provide additional useful information to Transact.

Names are case-sensitive. Values, including numeric quantities, are strings for both input and output.

Note

The Transact Order Entry API (OEAPI) documentation uses the term *attribute* when referring to a shopping cart argument. This document uses the term *parameter* instead to avoid confusion with an *attribute*, which is an asset that describes other assets in the catalog.

To set a parameter, call [CART.SETITEMPARAMETERS](#).

To get the value of a single parameter, call [CART.GETITEMPARAMETER](#). To get the values of a list of parameters, call [CART.GETITEMPARAMETERS](#).

For more information about shopping cart XML tags, see [Chapter 3, “CS-Direct Advantage XML Tags.”](#) For more information about shopping cart parameters, see [Chapter 16, “CS-Direct Advantage Cart Parameters.”](#)

Alphabetical List of Order Item Parameters

The following table lists all the order item attributes in alphabetical order:

- [AccountRequired](#)
- [AllowPaymentBrands](#)
- [AutoRenew](#)
- [CF__fieldname](#)
- [CF_R_fieldname](#)
- [CodeSet](#)
- [DeliveryMechanism](#)
- [Detail](#)
- [ExcludePaymentBrands](#)
- [FulfillmentURL](#)
- [GracePeriod](#)
- [InstallmentCount](#)
- [InstallmentPeriod](#)
- [Name](#)
- [OfferURL](#)
- [ReceiptDuration](#)
- [RefundPolicy](#)
- [ShippingCharge](#)
- [ShippingWeight](#)
- [Subscription](#)
- [SubscriptionDuration](#)
- [TaxClass](#)
- [TrialPeriod](#)
- [Type](#)
- [UniqueID](#)
- [VatFlag](#)
- [VatRate](#)

Order Item Parameters by Function

The following section provides brief descriptions of each parameter, in tables that are organized by function:

- [General Parameters](#)
- [Product Description Parameters](#)
- [Delivery Parameters](#)
- [Cost and Payment Parameters](#)
- [Subscription Parameters](#)
- [Tax Parameters](#)
- [Custom Field Parameters](#)

Each table contains the following four columns:

Name – Name of the parameter.

Description – A brief description of the parameter. More detailed explanations of the parameters follow each table.

Value Range – Range of values for the parameter.

Constraints – One of the following:

- *Required*: You must specify a value for this parameter.
- *Required if*: You must specify a value if the stated condition is true.
- *OK if*: You may specify a value only if the stated condition is true.
- *Prohibited if*: You cannot specify a value if the other specified parameter has the specified value.
- *Mutually Exclusive*: You cannot specify a value if the other specified parameter has any value.
- *Default*: The default value if you do not specify a value.

General Parameters

The following table summarizes general order item parameters:

Name	Description	Value Range	Constraints
AccountRequired	Transact account required?	0 = no; 1 = yes	Default: 0
		1 = yes	Required if Subscription = 1
CodeSet	The character set used in the following fields: ClassID Detail Name UniqueID	String. The preferred MIME name specified in the IANA character sets registry.	Default: Latin1
OfferURL	URL of the Web page or CGI script containing the digital offer; used as a link from the order form. This is the value in the Continue Shopping or Return to Store button.	Absolute or partial URL	Required

Product Description Parameters

The following table summarizes the parameters that describe products for sale:

Name	Description	Value Range	Constraints
Detail	Descriptive information; reported on seller's Smart Statement	Alphanumeric	Mutually exclusive: Subscription = 1
Name	Product name or description, reported on the Transact order form page and the Transact smart receipt	Alphanumeric	Required
Type	Type of product being offered or function being performed	online, service, or tangible	Required
UniqueID	Unique product/coupon identification code (for example, model number, SKU, etc.)	Alphanumeric; up to 40 characters	Required

Delivery Parameters

The following table summarizes the parameters that determine how physical goods, digital goods and digital subscriptions are delivered to buyers:

Name	Description	Value Range	Constraints
DeliveryMechanism	The method used for delivery of a digital good or subscription. Controls the presentation on the receipt page.	online or download	OK if Type = online Prohibited if Type != online
FulfillmentURL	Location (URL) of the digital good or subscription content.	Absolute or partial URL	Required if Type = online, credit, or micro Prohibited if Type = tangible, coupon, or none
ReceiptDuration	Time in seconds that the customer has to access a subscription or digital good from the time the digital receipt is created.	Integer; 1 or greater	Default: 30*24*60*60 seconds (30 days) Prohibited if Type = tangible, coupon, or none

Cost and Payment Parameters

The following table summarizes the parameters that determine the cost of goods and methods of payment:

Name	Description	Value Range	Constraints
AllowPaymentBrands	List of payment brands that can be used with this offer or coupon	Transact and store dependent; set to Demo to test stores	Prohibited if Type = micro or none
ExcludePaymentBrands	List of payment brands that cannot be used with this offer or coupon.	Transact and store dependent	Prohibited if Type = micro or none
ShippingCharge	Fixed shipping charge for the product	Money amount: numerals with optional decimal point, no currency symbol, no commas; example: 6.95	OK if Type = tangible
ShippingWeight	Product weight for computing a shipping charge	Integer, 0 or higher; units specified during store registration	OK if Type = tangible

Subscription Parameters

The following table summarizes the parameters for digital subscriptions:

Name	Description	Value Range	Constraints
AutoRenew	Subscription is automatically renewed when it expires	0 = no 1 = yes	OK if Subscription = 1 Default: 0
GracePeriod	Days before billing is performed for each billing cycle	Integer; 1 or greater	OK if Subscription = 1 Default: 0
InstallmentCount	Number of payments	Integer; 1 or greater	OK if Subscription = 1; Mutually Exclusive: AutoRenew = 1 Default: 1
InstallmentPeriod	Number of months between payments	Integer; 1 or greater	OK if InstallmentCount >= 2
RefundPolicy	Subscription refund policy	none prorate full	OK if Subscription = 1 Default: none
Subscription	Digital good is a subscription	0 = no 1 = yes	OK if Type = online Prohibited if Type != online Default: 0
SubscriptionDuration	Length of a subscription in months	Integer; 1 or greater	Required if: Subscription = 1 Prohibited if: Subscription = 0
TrialPeriod	Period before billing begins	Positive integer followed by d w m (days, weeks, months); example: 2w (two weeks)	OK if Subscription=1 Default: 0d (zero days)

Tax Parameters

The following table summarizes the parameters that control taxation of goods:

Name	Description	Value Range	Constraints
TaxClass	Product classification code used for sales tax/use tax calculation	TAXWARE product code (five-digit integer)	Prohibited if Type = coupon or none
VatFlag	Does price include VAT (value added tax)?	0 is yes 1 is no 2 is VAT exempt	Required if seller specifies a tax mode of VAT during store registration Prohibited if Type = coupon or none
VatRate	VAT rate charged for the item	A decimal number >= 0.0	Required if VatFlag = 0 or VatFlag = 1 Prohibited if Type = coupon or none

Custom Field Parameters

The following table summarizes the custom parameters:

Name	Description	Value Range	Constraints
CF__fieldname	Optional custom field	string starting with CF_ _ (note double underscore)	Prohibited if Type = none or micro
CF_R_fieldname	Required custom field	string starting with CF_R_	Prohibited if Type = none or micro

Detailed Descriptions of Parameters

This section provides full details for each parameter. The parameters are listed in alphabetical order.

AccountRequired

Required for digital subscription.

Optional for all offers other than digital subscription.

Indicates whether the buyer must be registered with Transact (have a buyer profile) to purchase this item. Setting AccountRequired to 1 means the buyer must be registered with Transact. Setting AccountRequired to 0 means the buyer does not have to be registered to purchase this item.

For digital subscriptions, you must set AccountRequired to 1. In other words, subscribers must be registered with Transact.

AllowPaymentBrands

Optional for all offers.

A comma-separated list of payment types that are accepted for this offer. Allowed payment types are dependent on the configuration of your electronic store as determined by your Transact administrator.

You can find out the codes for the available payment brands (usually two letter abbreviations) from the Transact administration screen, under "Payment Agent Configuration."

For example, to restrict the payment brands that can be used to purchase a particular item to Mastercard and American Express, set AllowPaymentBrands to MA, AM.

AutoRenew

Optional for digital subscriptions.

Prohibited for all other digital offers.

Indicates whether a digital subscription automatically renews when it expires. Two possible values:

- 0 (default) indicates no renewal.
- 1 indicates renewal.

If AutoRenew is set to 1, the buyer is automatically billed for the renewal.

CF__fieldname

Optional for all offers.

This is a seller-defined custom parameter for adding additional optional information to offers. These custom fields are not passed back in digital receipts.

You may customize screens to display custom fields. The name of the custom field in the offer must match the field name used in any custom screens.

CF_R_fieldname

Required for all offers.

This is a seller-defined custom parameter for adding additional required information to offers. These custom fields are not passed back in digital receipts.

You may customize screens to display custom fields. The name of the custom field in the offer must match the field name used in any custom screens.

Setting Custom Field Parameters

You can create your own required and optional custom field parameters to store additional information about a product. You create a custom parameter using the XML tag `CART.SETITEMPARAMETERS`. To create a custom parameter, simply specify a parameter name with the appropriate prefix.

Note

Make sure you insert two underscore characters in the prefix for an optional custom attribute.

CodeSet

Optional for all offers.

Specifies the code set used to display the following fields: `ClassID`, `Detail`, `Name`, and `UniqueID`.

DeliveryMechanism

Optional for digital goods and digital subscriptions.

Prohibited for all other offers.

There are two possible values:

- `online`
- `download`

Sets the text of the link on the Transact Smart Receipt that indicates how a digital good or digital subscription is delivered to a buyer: the buyer can either download the product, or

access the product online. Specifying online sets the link text to Browse it now. Specifying download sets the link text to Download it now.

Detail

Optional for all offers.

Descriptive text that only appears on the seller's Transact Smart Statement. Because buyers do not see this field, this information is only for the seller. See also Name.

ExcludePaymentBrands

Optional for all offers.

A comma-separated list of payment types that are *not* accepted by this offer or coupon. Allowed payment types are dependent on the configuration of your online store as determined by your Transact administrator.

You can find out the codes for the available payment brands (usually two letter abbreviations) from the Transact administrator, or from the Transact administration screen, under "Payment Agent Configuration." For example, to restrict the payment brands that can be used to purchase a particular item to Mastercard and American Express, set ExcludePaymentBrands to MA, AM.

FulfillmentURL

Required for digital goods, microtransactions, and digital subscriptions. Prohibited for tangible goods.

This URL specifies the location of the digital item, microtransaction item, or digital subscription to be delivered. The URL might specify an HTML page, or more typically a CGI program that manages the content delivery. This URL must be protected so that only buyers who have paid for this product have access.

FulfillmentURL is prefixed by several properties in the `transact.ini` file that identify a particular store. By default, these come from the default catalog server properties, identified in the `transact.ini` file by the string: `tx.defaultStore.*`.

Commerce Connector constructs the FulfillmentURL string as follows:

```
<fulfillmentScheme>://<fulfillmentServer>:<fulfillmentPort>/  
<fulfillmentURL><itemURL>
```

Default Store Property Values

If the default fulfillment server properties in `transact.ini` are:

```
tx.defaultStore.fulfillmentScheme=http  
tx.defaultStore.fulfillmentServer=myhost.com  
tx.defaultStore.fulfillmentPort=80  
tx.defaultStore.fulfillmentURL=/cgi-bin/gx.cgi/  
AppLogic+FTContentServer?
```

and `ItemURL` is `pagename=itempage`, then the concatenated URL string is:

```
http://myhost.com:80/cgi-bin/gx.cgi/
AppLogic+FTContentServer?pagename=itempage
```

Registered Store Property Values

If you have one or more stores registered with Transact, then the values that make up the OfferURL are determined by properties in `transact.ini` that correspond to a specific store, identified by the string `tx.store_N.*`, where N is a non-negative integer that identifies the store. For example, `tx.store_0.*`, identifies your store number one.

For example, if the registered store properties in `transact.ini` are:

```
tx.store_0.fulfillmentScheme=http
tx.store_0.fulfillmentServer=myhost.com
tx.store_0.fulfillmentPort=80
tx.store_0.fulfillmentURL=/cgi-bin/gx.cgi/AppLogic+FTContentServer?
```

and *ItemURL* is `pagename=getitem`, then the concatenated URL string is:

```
http://myhost.com:80/cgi-bin/gx.cgi/
AppLogic+FTContentServer?pagename=getitem
```

When you register a store with Transact, you must use the Content Server Property Editor to manually enter the properties for that store. For instructions on using the Property Editor, see the *CSEE Administrator's Guide*.

GracePeriod

Optional for digital subscriptions.

Prohibited for all other offers.

Specifies the number of days before billing occurs for each billing cycle.

Default: 0 (no grace period).

For offers that set both `GracePeriod` and `TrialPeriod`, the first billing starts at:
`purchase date + GracePeriod + TrialPeriod`

For installment payments, subsequent billings are not influenced by `TrialPeriod`. For example, consider a subscription purchased on January 1 in which:

- `TrialPeriod` is set to 2m (two months)
- `GracePeriod` is set to 14 (days)
- `InstallmentPeriod` is set to 6 (months).

In this case, the buyer is first billed on March 15 (January 1 + 2 months + 14 days), and next billed on September 15 (first bill + 6 months).

InstallmentCount

Optional for digital subscriptions. Requires that `InstallmentPeriod` be set.

Prohibited for all other offers.

The number of installments to be billed for this digital subscription. The amount billed for each installment is given by `Price`.

If `InstallmentCount` is set, then `InstallmentPeriod` is required.

Note

`InstallmentCount` and `InstallmentPeriod` cannot be used if `AutoRenew` is set to 1.

InstallmentPeriod

Optional for digital subscriptions. Requires that `InstallmentCount` be set.

Prohibited for all other offers.

Specifies the number of months between installment payments. For example, setting `InstallmentPeriod` to 6 indicates that installment payments are due every six months. The amount billed at each installment is the value of `Price`.

If `InstallmentPeriod` is set, then `InstallmentCount` is required.

`InstallmentCount` and `InstallmentPeriod` cannot be used if `AutoRenew` is set to 1.

Name

Required for all offers.

A brief label for this product or coupon, such as `Digeridoo Australian Shiraz` or `Don Corleone Italian Chianti`. You can assign any string for `Name`.

Transact uses the value of `Name` in:

- The Product Description column of the Transact Order Form screen.
- The Product Description column of the Smart Receipt.

The value of `Name` does not have to be the same as the value of `Detail`.

OfferURL

Required for all goods.

This is the URL of the Web page containing the catalog entry for the item, `OfferURL` links Transact pages back to the store.

`OfferURL` is prefixed by several properties in the `transact.ini` file that identify a particular store. By default, these come from the default catalog server properties, identified in the `transact.ini` file by the string: `tx.defaultStore.*`.

Commerce Connector constructs the complete `OfferURL` string as follows:

```
<catalogScheme>://<catalogServer>:<catalogPort>/  
<catalogURL><itemURL>
```

Default Store Property Values

If the default catalog server properties in `transact.ini` are:


```
tx.defaultStore.catalogScheme=http
tx.defaultStore.catalogServer=myhost.com
tx.defaultStore.catalogPort=80
tx.defaultStore.catalogURL=/cgi-bin/gx.cgi/
AppLogic+FTContentServer?
```

and *ItemURL* is *pagename=itempage*, then the concatenated URL string is:

```
http://myhost.com:80/cgi-bin/gx.cgi/
AppLogic+FTContentServer?pagename=itempage
```

Registered Store Property Values

If you have one or more stores registered with Transact, then the values that make up the *OfferURL* are determined by properties in *transact.ini* that correspond to a specific store, identified by the string *tx.store_N.**, where *N* is a non-negative integer that identifies the store. For example, *tx.store_0.**, identifies your store number one.

For example, if the registered store properties in *transact.ini* are:

```
tx.store_0.catalogScheme=http
tx.store_0.catalogServer=myhost.com
tx.store_0.catalogPort=80
tx.store_0.catalogURL=/cgi-bin/gx.cgi/AppLogic+FTContentServer?
```

and *ItemURL* is *pagename=itempage*, then the concatenated URL string is:

```
http://myhost.com:80/cgi-bin/gx.cgi/
AppLogic+FTContentServer?pagename=itempage
```

When you register a store with Transact, you must use the Content Server Property Editor to manually enter the properties for that store. For instructions on using the Property Editor, see the *CSEE Administrator's Guide*.

ReceiptDuration

Optional for digital goods.

Prohibited for all other offers.

The length of time in seconds before the digital receipt generated for this transaction expires, measured from the time of creation. That is, the length of time a buyer has to access the digital good after he purchases that access. There is no default value.

RefundPolicy

Optional for digital subscriptions.

Prohibited for all other offers.

Specifies the refund policy for this digital subscription. Possible values:

- none (default) – subscribers do not receive a refund after cancellation.
- full – subscribers are refunded the entire amount of their current subscription.
- prorate – Subscribers are refunded an amount proportional to the unused part of their current subscription (for example, if the subscriber has paid for a twelve month

digital subscription, and cancels after six months, they are refunded half the cost of the subscription).

ShippingCharge

Optional for tangible goods.

Prohibited for all other offers.

The fixed cost of shipping this item.

For stores selling tangible goods, you must configure one or more transport methods when you register your electronic store. For each transport method and zone, you must indicate one of the following shipping models:

- flat-rate – Shipping charge for the entire order has a fixed rate.
- amount-based – Shipping charge is determined using the total cost of all the items in the order as an index into a transport rate table (defined during configuration).
- weight-based – Shipping charge is determined using the total weight of all the items in the order as an index into a transport rate table (defined during configuration).

If ShippingCharge is set, its value supersedes the Transact shipping model for an item. For example, if ShippingCharge is 5 and Currency is USD, then the shipping charge for this item is \$5.00, regardless of the product's total cost or weight.

If an order consists entirely of items with ShippingCharge set, then the shipping cost of the order is the sum of all the ShippingCharge values, even if the shipping model for the transport/zone combination is flat-rate.

An order can also consist of some items with ShippingCharge set and others with ShippingCharge not set. In this kind of order, items with ShippingCharge set do not figure into the order's total cost or weight. For example, given the weight-based shipping model, consider the following three-item order:

- A bowling ball weighs 6 kilograms. ShippingCharge is not set.
- A painting weighs 8 kilograms. ShippingCharge is not set.
- A toy weighs 10 kilograms. ShippingCharge is set to 2.

In this case, the total shipping charge is the sum of

- The charge for shipping 14 kilograms (weight of the bowling ball and the painting)
- The fixed \$2 charge for shipping the toy.

Note

The commerce utilities cannot diagnose inappropriate, missing, or extraneous ShippingCharge values. It is your responsibility to ensure that ShippingCharge is set properly.

ShippingWeight

Optional for tangible goods.

Prohibited for all other offers.

The weight of this product. The unit of weight is specified when you register your store. For example, if you specify “kilograms” as the weight unit, then setting ShippingWeight to 7 indicates the product’s shipping weight is 7 kilograms.

If ShippingWeight and ShippingCharge are specified in the same offer, ShippingWeight is ignored.

If the Quantity parameter has a value greater than 1, then you should set ShippingWeight to the total weight, not the weight of each individual item. For example, suppose a certain bowling ball weighs 6 kilograms. To represent three bowling balls, you should set Quantity to 3 and ShippingWeight to 18.

Note

Transact cannot diagnose inappropriate, missing, or extraneous ShippingCharge values. Be sure that ShippingCharge is set properly.

Subscription

Required for digital subscriptions.

Prohibited for all other offers.

Specifies whether this product is a digital subscription. Setting Subscription to 1 means this product *is* a subscription. Setting Subscription to 0 (the default) means the product *is not* a subscription.

SubscriptionDuration

Required for digital subscriptions.

Prohibited for all other offers.

The length of the digital subscription in months, ranging from 1 to 99. For example, setting SubscriptionDuration to 24 indicates a two-year subscription.

TaxClass

Optional for all offers.

Specifies a classification code that determines the sales tax for this product. Codes are determined by Taxware, which Transact uses to calculate taxes in the USA and Canada. Not used outside of the USA and Canada.

TrialPeriod

Optional for digital subscriptions. Prohibited for all other offers.

The length of time that a buyer receives a digital subscription before getting billed for the subscription. It is expressed as an integer and an optional time unit (d, w, or m for days, weeks, or months, respectively). Do not put any white space between the integer and the time unit. For example, setting `TrialPeriod` to 2m lets the buyer receive a digital subscription for two months without getting billed.

The default integer is 0 and the default time unit is d (days). Thus, the default trial period is 0 days.

See also `GracePeriod`.

Type

Required for all offers.

Specifies the type of offer.

Possible values:

- `tangible` – a tangible, or physical, good.
- `online` – a digital good. Includes digital subscriptions.
- `credit` – a microtransaction credit.

UniqueID

Required for all offers.

If `Type` is set to `tangible`, `UniqueID` specifies a product identification code such as an SKU or model number. The `UniqueID` can be any sequence of letters and numbers up to 40 characters long.

If `Type` is set to `online` or `micro`, `UniqueID` specifies the token under which the `FulfillmentURL` is protected. The `UniqueID` is returned by the transaction server as the Group ID in the digital receipt or digital ticket. That value must match the group ID set in the content protection scheme (ticketing).

You can set `UniqueID` to any alphanumeric value of up to 40 characters in length.

VatFlag

See below for required and prohibited offers.

Indicates whether the value of the `Price` parameter includes the VAT.

When you register your electronic store, Transact asks you to choose one of the following taxation settings: `No Tax`, `Sales Tax`, or `VAT`. You must set `VatFlag` if you selected `VAT` (value-added tax) as the taxation mode. `VatFlag` is prohibited if you selected `No Tax` or `Sales Tax` as the taxation mode.

- Setting `VatFlag` to 0 indicates the value of `Price` includes VAT.

- Setting `VatFlag` to 1 indicates the value of `Price` does not include VAT.
- Setting `VatFlag` to 2 indicates this item is exempt from VAT.

VatRate

The decimal rate for calculating the VAT (0.175 = 17.5%).

Required if `VatFlag` is 0 or 1. Ignored if `VatFlag` is 2.

Prohibited for microtransactions.

The value of `VatRate` is the VAT percentage expressed as a decimal value between 0.0 and 1.0. For example, current UK VAT rates are:

Name of Tax	Percentage	What You Assign to <code>VatRate</code>
standard-rate	17.5%	0.175
reduced-rate		0.08
zero-rate	0%	0

Setting `VatFlag` to 2 (VAT exempt) is not equivalent to setting `VatRate` to 0.0 (zero-rate). If a product is VAT exempt, the seller cannot reclaim VAT paid on raw materials used to create that product. By contrast, if a product is zero-rate, the seller can reclaim VAT paid on raw materials used to create that product.

Consult your local VAT office for the current rates and VAT classification of the products you are selling.

Chapter 18

CS-Direct Advantage Buyer Parameters

This chapter describes the buyer parameters that are used primarily in the CSUSER tags. You use CSUSER tags to produce, obtain, and maintain information about buyers who are registered with Transact. The CSUSER tags require Commerce Connector for communication with Transact. For descriptions of the CSUSER tags, see [Chapter 3, “CS-Direct Advantage XML Tags.”](#)

Buyers who are registered with Transact have already set up accounts that identify their names, billing addresses, and payment brands. When registered buyers log in and identify themselves, CS-Direct Advantage retrieves their account information and sends it using the Commerce Connector to Transact for processing.

In contrast, walk-in buyers do not have Transact accounts. Therefore, walk-in buyers must provide all name, billing address, and payment brand information.

Buyer parameter names are case-sensitive. Values, including numeric quantities, are strings for both input and output.

Some buyer parameters are composite names that consist of a field name concatenated with a hyphen and a secondary field. Those parameters with the same field name are stored within a common hierarchical object.

You set and retrieve individual buyer parameters in a manner similar to that used for shopping cart parameters.

accessName

The buyer's user name.

You can use either accessName or userID to reference buyer profile information.

password

The buyer's password; used for basic authorization.

Note

The password parameter is not preserved between calls to the Commerce Connector (using [CSUSER.REGISTER](#), [CSUSER.LOOKUP](#), [CSUSER.MODIFY](#), or [CSUSER.AUTHENTICATE](#)). If you want to save the password, you must do so independently.

locale

Optional. The locale within which this buyer operates.

This field contains a series of tokens separated by hyphens. The tokens may be any combination of letters and numbers and may be of any length, for example, en-US. (You can obtain valid token strings from the Transact administrator.)

address-name1

The buyer's first name, or given name.

address-name2

Optional. The buyer's last name, or surname.

Note

Whether both name fields are required is determined by Transact configuration. If both fields are not required, the whole name is placed in the address-name1 field.

address-address1

The buyer's street address.

address-address2

Optional. Usually an apartment number, suite number, or office number.

address-city

The buyer's city or town.

In US, legal values are obtainable after check/commit.

address-state

The two-letter abbreviation for the buyer's U.S. state or Canadian province.

address-postalCode

The buyer's postal code or U.S. ZIP code.

In US, legal values are obtainable after check/commit.

address-country

The buyer's country of residence.

address-county

The buyer's county of residence. Only required in US, if the city, state, and postalCode combination does not uniquely identify the county.

In US, legal values are obtainable after check/commit.

address-emailAddress

The buyer's e-mail address, exactly as the buyer entered.

address-telephone

The buyer's phone number, exactly as the buyer entered.

address-fax

The buyer's fax number, exactly as the buyer entered.

userID

The identifier of the buyer, userID is used in calls to [CSUSER.LOOKUP](#) and [CSUSER.MODIFY](#). This value is returned by the Commerce Connector upon successful registration or authentication.

The user ID parameter is used in calls to [VDM.SETCOMMERCEID](#).

Section 5

Appendixes

This section contains the following appendixes:

- [Appendix A, “Error Conditions”](#)
- [Appendix B, “CSEE Reserved Variables”](#)
- [Appendix C, “TimeZoneID Numbers”](#)

Appendix A

Error Conditions

Errors reported by Content Server tags are stored in the variable, `errno`. When the value of `errno` needs to be checked for an error condition, it should be set to 0 (zero) prior to using a tag and then checked after the tag to see whether the value has changed.

You can find information about revision tracking, BlobServer, and EvalServer errors by checking [errdetail Error Values](#).

Note

To support the standard J2EE JNDI interface, any of the directory tags may cause any directory error to be set. Directory tag client code should gracefully handle all of the error codes described in this document for any tag call.

errno Error Values

Value	Description
-1	Unknown user.
-2	Bad password.
-3	No privileges.
-4	Unknown error.
-5	No session.
-6	Not supported.
-7	No rt privileges.
-10	No such element.
-11	Can't find element definition.
-12	Error evaluating element.
-13	Limit out of bounds.
-14	Process exception error.
-15	Can't find SQL file.
-16	Empty list.
-20	Memory error.
-21	Memory error.
-22	Table already exists.
-30	Page not found.
-31	Page not live.
-32	No privileges to page.
-50	Bad start index.
-51	Bad end index.
-52	Empty output name.
-53	Empty search string.
-100	General exception.
-101	No rows returned.
-102	Empty column.
-103	No table.
-104	No table definition.
-105	Database error.

Value	Description
-106	Bad parameters.
-107	Duplicate.
-108	Tree selector error.
-109	Tree path not valid.
-110	Tree does not exist.
-111	No nodes.
-112	No parent.
-113	Invalid object information.
-114	Missing class type.
-115	Unknown object class.
-116	Unknown object ID.
-117	Unknown object style.
-118	No node to get.
-119	Child node error.
-200	Register event failed.
-201	Event exception.
-202	Email exception.
-203	Error managing event.
-204	Managed database event error.
-205	Event database exception.
-300	File not found.
-301	File not written.
-302	File not deleted.
-303	Path not found.
-304	File not read.
-305	Zip entry not read.
-306	Zip unpack error.
-307	File overwrite error.
-308	Zip exception.
-309	Zip validate error.
-310	File not folder.
-311	Folder not deleted.

Value	Description
-401	No XML input.
-402	No XML filter definition.
-403	XML parse failed.
-404	XML write failed.
-405	Bad XML file.
-406	Bad XML parameters.
-407	XML document failed.
-408	Bad XML syntax.
-500	Cannot go to row.
-501	Bad counter value.
-502	List has no columns.
-503	Non ContentServer list.
-504	List rename error.
-505	List copy error.
-506	Export error.
-507	Servlet I/O error (broken pipe).
-601	Mirror operation not running.
-602	Mirror operation in progress.
-603	Mirror variable not found.
-604	Mirror file not found.
-605	Mirror bad column type.
-606	Mirrored no rows.
-607	Mirror operation aborted.
-608	No mirror server.
-609	Network error occurred during mirror operation.
-610	Cannot mirror mixed columns.
-611	Mirror operation not completed.
-612	Bad mirror configuration information.
-613	Failed to create mirrored target.
-614	Not temp table.
-615	Node temp table.
-616	Attempt to mirror trees or tree nodes failed.

Value	Description
-617	Attempt to mirror tree table failed.
-700	Exception calling AppLogic.
-701	AppLogic error.
-800	Bad search type.
-801	Cannot load search engine.
-802	Unsupported search function.
-803	Search index create failed.
-804	Search index delete failed.
-805	Search add index failed.
-806	Search remove index failed.
-807	No search index found.
-808	Unknown search engine.
-809	Search failed.
-810	Bad character set.
-811	Bad native method call.
-812	Search no index.
-813	Bad search date.
-814	No field argument.
-815	Duplicate field values (Verity search).
-816	Pop file not created (Verity search).
-817	Data file not created (Verity search).
-818	Bad parser (Verity search).
-819	Unknown field type (Verity search).
-820	Unknown field (Verity search).
-821	No modified insert date (Verity search).
-822	Index already exists (Verity Search).
-823	Native method mismatch (Verity or AltaVista Search).
-850	No DRE connection.
-900	Class not found.
-901	No access.
-902	No instance.
-903	Execute error.

Value	Description
-904	Class cast.
-1000	Revision tracking error.
-2000	No command specified.
-2001	XML source descriptor not found. Valid assettype or assetdescriptor argument must be supplied to AssetMaker.
-2002	Descriptor parse failed.
-2003	Descriptor file is missing PROPERTIES tag.
-2004	No name field was specified for the asset in the descriptor file.
-2005	An error occurred creating the asset table. Please check the descriptor file for invalid STORAGE tags.
-2006	An error occurred creating the asset table. The table already exists.
-2007	An error occurred registering asset elements.
-3000	Bad certificate (SSL).
-3000	Request cache error.
-3100	BlobServer error.
-3200	EvalServer error.
-3300	Invalid JSP root.
-3301	Invalid JSP path.
-3302	Invalid object in hash.
-3303	No file created.
-3304	No dispatcher.
-3305	Error encountered when calling include file.
-3306	Problem with include file.
-3307	Bad parameters.
-3308	No JSP support.
-3400	Error executing database prepared statement.
-3401	Error executing database statement.
-3402	Error fetching from database.
-3403	Error on lookup of JNDI data source.
-3404	Installation error with JDBC driver.
-3405	Illegal access error with JDBC driver.
-3406	No class definition error with JDBC driver.
-3407	Error initializing database connection.

Value	Description
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-10006	The object ID is not valid.
-10007	The version of the object is not valid.
-10008	The node ID of the object is not valid.
-10009	More than one object met the specified criteria.
-10010	Invalid attribute value.
-10011	Table is not revision tracked.
-10012	Object is not checked out.
-10013	Object checked out by another user.
-10014	Object already checked out.
-10015	Cannot check out specified object.
-10016	Cannot check in specified object.
-10017	Cannot roll back specified object.
-10018	Cannot undo checkout of the specified object.
-10019	Table is not an object table.
-10020	Attempt to track a tracked table.
-10021	Cannot track specified object.
-10022	Cannot untrack specified object.
-10023	Invalid table name.
-10024	User not allowed access for request.
-12001	An expected table or Ilist field was not found.
-12002	The ID of an object table is not valid.
-12003	The specified project asset is empty, or there was a database error while retrieving its contents.
-12004	An asset was not found for a specified ID.
-12005	There is no list with the specified name.
-12006	There is no such asset type.
-12007	The specified value is not valid.

Value	Description
-12008	Could not create object or asset.
-12009	A required parameter (argument) is missing.
-12010	A class was not found.
-12011	Mirror failed.
-12012	FTCS parse error.
-12013	No such resource.
-12014	Cannot be child asset.
-12015	Missing site node.
-12016	End queue.
-12017	Illegal asset relation.
-12018	Missing publication.
-12019	Bad template page name.
-12020	No object by name.
-12021	Invalid version (an integer was expected).
-12022	Missing property.
-12023	Missing password.
-12024	Missing group.
-12025	Missing process.
-12026	Missing step.
-12027	Cannot cancel.
-12028	Not logged in.
-12029	Missing user.
-12030	Insufficient privilege.
-12031	Wrong group state.
-12032	Create reference failed.
-12033	Export failed.
-12034	Publish in progress.
-12035	No asset assignment.
-12036	Wrong asset state.
-12037	Required field missing.
-12038	Incorrect data type.
-12039	Field too long.

Value	Description
-12040	Already in group.
-12041	Already in workflow.
-12042	Unknown attribute in asset.
-12043	Data out of range.
-12044	Pub setup failure.
-12045	Pub setup remote failure.
-12046	Pub cleanup failure.
-12047	Pub cleanup remote failure.
-12048	Not in group.
-12049	Not in workflow.
-12050	Incompatible process.
-12051	Directory error.
-12052	Null object.
-12053	Cannot delegate.
-12054	XML parser error.
-12055	No folder.
-12056	Name of asset used is not unique (for example, asset by this name already exists).
-12057	Core field required for a save of the asset is not available (assettype, subtype, name, clientapp).
-12058	Core field Publist , which contains the site ID, is not available during a save of the asset.
-12059	A particular field that was required for a save of the asset was not found; marked required by either external client admin or by the asset description (on inspect).
-12060	A field sent to the save asset function (after XML parsing) was too long.
-12061	RAE function being accessed for a site that the user does not have access to.
-12062	Ask step requires assignees.
-12063	Unsupported asset search.
-12064	Invalid asset search.
-12065	Asset publication update error.
-12066	Subtype required.
-12067	Deadline not allowed.

Value	Description
-12068	Illegal process deadline.
-13000	General – <code>basic.jar</code> . Represents the starting errno, but otherwise does not identify an error condition.
-13001	Column, intended for <code>NoSuchFiledException</code> conditions. Currently unused.
-13002	Asset class. Currently unused.
-13003	No such field. Indicates that a tag uses incorrect parameters.
-13004	Currency format. The <code>CURRENCY.READCURRENCY</code> tag encountered an incorrect number or improper placing of commas, decimals, and so forth.
-13005	Decimal format. The <code>DECIMAL.READDECIMAL</code> tag cannot convert in input string to a decimal.
-13006	Class not found. The <code>misc.callmain</code> tag discovered that <code>.jar</code> files were missing from the classpath during installation. This may or may not be a problem. The names of the missing files are written to <code>stdout</code> and are logged to the installation log.
-13050	General – <code>assetframework.jar</code> . Usually indicates that fields or columns of a resultset or list are not as expected. This can happen with mixed CS-Direct versions, or within a dated schema.
-13051	Column, intended for <code>NoSuchFiledException</code> conditions. Currently unused.
-13052	Asset class. Usually indicates a problem with the Java class referenced in either the <code>ComplexAsset</code> or the <code>AssetType</code> table. Could mean that <code>assetmaker.jar</code> is missing.
-13053	No such field.
-13054	Bad publish target. Attempt to call <code>OpenMarket/Xcelerate/PrologActions/Publish/Mirror1/RemoteComplexAssetPublish</code> page on the target publish system failed.
-13055	Remote Publish Save Failure. Save of the published asset on the target system failed. Check the target system for errors.
-13100	Commerce initialization. There was a problem initializing <code>mwb.commercecontextclass</code> in <code>gator.ini</code> . Defaults to: <code>com.openmarket.gator.commercecontext.CommerceContext</code> .
-13101	Relationship error.

Value	Description
-13102	Attribute error. Possible causes include: • AssetType name (PAttributes, for example) is spelled incorrectly. • A tag references an invalid name of an attribute asset. • Attempted save of more than one value for a single-value attribute. • Attempted save of a value for an attribute that is defined as remote. • Multiple values specified for an attribute that is defined as remote.
-13103	XML parse error.
-13104	No index for attribute. A search engine indexed attribute is missing its indices.
-13105	Illegal external operation. The external ID of an external storage style attribute is null.
-13106	Invalid input list. Used by various cscommerce classes to indicate bad input data.
-13107	Cart merge failed.
-13108	Cart merge not allowed.
-13109	Negative price for line item.
-13110	No such cart in cartset.
-13111	Cartset data row does not match cartset index.
-13500	Content Server commerce error.
-13501	Impossible condition – probable bug.
-13502	Function not implemented, or Commerce Connector not installed.
-13510	Bad configuration data in <code>transact.ini</code> .
-13511	Invalid store ID.
-13512	Invalid input list.
-13513	Invalid cart or order parameter.
-13514	Invalid cart or order item parameter.
-13515	Invalid quantity for line item.
-13516	No such line item in cart or order.
-13517	Unknown currency.
-13518	Currency mismatch.
-13519	Invalid DO or DC.
-13520	Invalid user.
-13530	Cart or order has no items.
-13531	Cart or order already checked.

Value	Description
-13532	Cart or order must be checked first.
-13533	Error returned by remote commerce system.
-13700	Visitor Error. Possible causes are: • Rulesets are not of the correct type • Runtime class generation or compilation errors Check <code>vis.genclasspath</code> and <code>vis.compileclasspath</code> in <code>visitor.ini</code> .
-13701	No argument.
-13702	Bad argument.
-13703	Invalid object state. Possible causes include: • Invalid constraints on visitor or history attributes • Trying to retrieve data for an unknown user
-13704	Constraint violation. Violation of constraints on visitor or history attributes.
-13705	Type mismatch. Data or constraints do not match the data type on a visitor or history attribute.
-13706	Type unsupported. The attribute data type specified is not valid for a particular constraint.
-13707	Value not specified. A required attribute value was not specified.
-13708	Object not found. The name of a visitor or history attribute is misspelled or does not exist.
-13709	Object type unsupported
-13710	Function unavailable. An indication from Commerce Connector that an operation is unavailable, such as trying to write to the Transact database.
-13711	Commerce Connector status error. An indication that Commerce Connector issued an error.
-13712	Commerce Connector status error. An indication that Commerce Connector issued an error.
-13713	Commerce ID not set.
-13714	Connection closed. Currently unused.
-13715	Reserved name. Certain visitor and access key names are reserved for system use.
-13716	Rule parse error.
-15002	Name being created already exists.
-15003	Name being deleted does not exist.
-15004	Implementation doesn't support this operation

Value	Description
-15005	Unexpected naming exceptions from JNDI provider.
-15006	A required property missing, or class name specified for factory classes is in error.
-15007	Unexpected schema violation errors from JNDI provider.
-15008	General error in the syntax of an IName.
-15009	Unexpected invalid attribute exceptions from JNDI provider
-15010	Unexpected invalid attribute id exceptions from JNDI provider.
-15011	Unexpected invalid attribute value exceptions from JNDI provider.
-15012	User does not have permission to perform the operation.
-15013	Node cannot be deleted because it has children.
-15014	Unexpected JNDI read error.
-15015	Unexpected JNDI write error.
-15016	JNDI attribute already in use.
-15017	User is already a member.
-15018	User is not a member.
-15020	User is not valid.
-15023	Unexpected JNDI connection errors.
-15024	Required attribute missing.
-15025	Unknown error.
-15026	Missing required create attribute.
-15027	User create SQL failure.
-15028	User delete SQL failure.
-15029	Cannot delete user attributes.
-15030	SQL error.
-15031	Query failed.
-15032	Add attribute SQL failed.
-15033	Database error occurred while handling new attribute.
-15034	Database error occurred while attempting to delete an attribute.
-15036	Missing or invalid child argument tags for tag.
1	Success.
2	No change to current row.
3	Initialization complete.

Value	Description
4	Table created.
5	Replaced rows id.
6	Column case lower.
7	Column case mixed.
8	Validate completed.
9	Search index exists.
10	Tree successfully created.
10	Revision tracking system not initialized.
10	Revision tracking system not initialized.
11	Temporary table created.
11	User logged in.
12	Mirror get configuration.
13	Database ping acknowledgment.
20	Failed to get the tracked table.
21	Invalid version number for asset referenced in the tracker table.
22	Asset to be tracked does not exist in source.
23	Failed to get column data from source table.
24	Failed to read file referenced in a URL field.
25	Column in source table does not have corresponding column in tracker table.
26	Failed to insert row in to tracker table.
30	Failed to get tracked table.
31	Failed to find revision in tracker table.
32	Failed to get column data from tracker table.
33	Failed to read file for revision.
34	Column in tracker table does not have a corresponding column in source table.
35	Column in tracker table does not have a corresponding column in source table of the correct type.
36	Failed to replace row in source table.
40	Failed to find revision in tracker table.
41	Failed to delete row from tracker table.
50	User cannot lock revision.

Value	Description
51	Failed to update row in tracker table.
60	Failed to update row in tracker table.
70	User cannot commit revision.
71	Failed to find revision in tracker table.
72	Failed to verify maximum versions.
80	User cannot release revision.
81	Failed to find revision in tracker table.
90	Failed to get tracker table.
91	User cannot roll back revision.
92	Failed to find asset in tracker table.
93	Failed to find revision for asset in tracker table.
100	Table not currently tracked.
101	User cannot query for revisions on this table.
102	User can only query for revisions created by or locked by self.
110	Source table already tracked.
111	Failed to get source table to be tracked.
112	Failed to create tracker table.
113	User cannot track this table.
120	Source table not currently tracked.
121	Failed to get tracker table.
122	Failed to delete tracker table.
123	User cannot track this table.
130	User cannot unlock records in this table.
131	Failed to find asset in tracker table.
140	User cannot delete revisions for this table.
150	User cannot set number of revisions for this table.
151	Source table not currently tracked.
152	Failed to update number of revisions for this table.
160	User cannot retrieve revision.
161	Failed to find asset in tracker table.
162	Failed to find revision for asset in tracker table.
163	Failed to get tracked table.

Value	Description
164	Failed to get column data from tracker table.
165	Failed to read file for revision.
166	Column in tracker table does not have corresponding column in source table.
167	Column in tracker table does not have corresponding column in source table of correct type.

errdetail Error Values

For revision tracking operations, the reserved variable named `Variable.errdetails` provides additional information about the error.

BlobServer and EvalServer

The following table lists errdetails for BlobServer and EvalServer.

Value	Description
10	General init failure, probably due to missing or invalid property file or maybe missing a required <code>.jar</code> file in the classpath. Used by both BlobServer and EvalServer.
11	Invalid property file. <code>errdetail1</code> is the name of the missing or invalid property, <code>errdetail2</code> is the value of the property in <code>futuretense.ini</code> . Used by both BlobServer and EvalServer.
12	Missing command. BlobServer or EvalServer parameters did not specify either a <code>serve</code> or <code>flush</code> command.
13	Missing required parameters. Used by BlobServer if one of the required request parameters is missing (<code>blobcol</code> , <code>blobwhere</code> , or <code>blobkey</code>). <code>errdetail1</code> is the value provided for <code>blobcol</code> , <code>errdetail2</code> is the value provided for <code>blobwhere</code> , <code>errdetail3</code> is the value provided for <code>blobkey</code> .
20	Security exception flushing the cache. Used by BlobServer or EvalServer for a flush request that failed a security permissions check.
21	Security exception serving a request. Used by BlobServer or EvalServer for a blob or page request that failed a security permissions check.
30	Unknown error generating a response for a serve request. For EvalServer, <code>errdetail1</code> is the name of the page requested that has failed to evaluate. For BlobServer, <code>errdetail1</code> is the requested table, <code>errdetail2</code> is the requested column, <code>errdetail3</code> is the requested where criteria, and <code>errdetail4</code> is the requested column to match the where criteria.

Appendix B

CSEE Reserved Variables

The following table lists the reserved variables for CSEE.

Variable	Description
tablename	Required in multiple CATALOGMANAGER and <code>ics:catalogmananager</code> commands and some XML tags.
pagename	Required input to the ContentServer servlet.
ftcmd	Required input to CatalogManager.
username	Session variable for user names.
currentACL	Session variable for the ACL list of the current user.
currentUser	Session variable for the user id.
evalpage	Required input to evalserver.
empty	Sets an empty string, use the built in XML tag CS.EMPTY instead.
errno	Current or last error code.
null	Set to one blank, deprecated.
c	Asset type name.
cid	Asset ID.
cv	Reserved for future use.
p	Parent page asset's ID.
rendermode	Render mode of the asset.

ZPostedRawDataBytes	This variable holds the binary bytes that are part of an HTTP request.

Appendix C

TimeZoneID Numbers

This appendix lists the TimeZoneID numbers that you can use with the [Date Format Tags](#).

ID	Time Zone	Description
0	MIT	West Samoa Time
1	Pacific/Apia	West Samoa Time
2	Pacific/Niue	Niue Time
3	Pacific/Pago_Pago	Samoa Standard Time
4	America/Adak	Hawaii-Aleutian Standard Time
5	HST	Hawaii Standard Time
6	Pacific/Fakaofo	Tokelau Time
7	Pacific/Honolulu	Hawaii Standard Time
8	Pacific/Rarotonga	Cook Is. Time
9	Pacific/Tahiti	Tahiti Time
10	Pacific/Marquesas	Marquesas Time
11	AST	Alaska Standard Time
12	America/Anchorage	Alaska Standard Time
13	Pacific/Gambier	Gambier Time
14	America/Los_Angeles	Pacific Standard Time
15	America/Tijuana	Pacific Standard Time
16	America/Vancouver	Pacific Standard Time
17	PST	Pacific Standard Time

18	Pacific/Pitcairn	Pitcairn Standard Time
19	America/Dawson_Creek	Mountain Standard Time
20	America/Denver	Mountain Standard Time
21	America/Edmonton	Mountain Standard Time
22	America/Mazatlan	Mountain Standard Time
23	America/Phoenix	Mountain Standard Time
24	MST	Mountain Standard Time
25	PNT	Mountain Standard Time
26	America/Belize	Central Standard Time
27	America/Chicago	Central Standard Time
28	America/Costa_Rica	Central Standard Time
29	America/El_Salvador	Central Standard Time
30	America/Guatemala	Central Standard Time
31	America/Managua	Central Standard Time
32	America/Mexico_City	Central Standard Time
33	America/Regina	Central Standard Time
34	America/Tegucigalpa	Central Standard Time
35	America/Winnipeg	Central Standard Time
36	CST	Central Standard Time
37	Pacific/Easter	Easter Is. Time
38	Pacific/Galapagos	Galapagos Time
39	America/Bogota	Colombia Time
40	America/Cayman	Eastern Standard Time
41	America/Grand_Turk	Eastern Standard Time
42	America/Guayaquil	Ecuador Time
43	America/Havana	Central Standard Time
44	America/Indianapolis	Eastern Standard Time
45	America/Jamaica	Eastern Standard Time
46	America/Lima	Peru Time
47	America/Montreal	Eastern Standard Time
48	America/Nassau	Eastern Standard Time

49	America/New_York	Eastern Standard Time
50	America/Panama	Eastern Standard Time
51	America/Port-au-Prince	Eastern Standard Time
52	America/Porto_Acre	Acre Time
53	America/Rio_Branco	GMT-05:00
54	EST	Eastern Standard Time
55	IET	Eastern Standard Time
56	America/Anguilla	Atlantic Standard Time
57	America/Antigua	Atlantic Standard Time
58	America/Aruba	Atlantic Standard Time
59	America/Asuncion	Paraguay Time
60	America/Barbados	Atlantic Standard Time
61	America/Caracas	Venezuela Time
62	America/Cuiaba	Amazon Standard Time
63	America/Curacao	Atlantic Standard Time
64	America/Dominica	Atlantic Standard Time
65	America/Grenada	Atlantic Standard Time
66	America/Guadeloupe	Atlantic Standard Time
67	America/Guyana	Guyana Time
68	America/Halifax	Atlantic Standard Time
69	America/La_Paz	Bolivia Time
70	America/Manaus	Amazon Standard Time
71	America/Martinique	Atlantic Standard Time
72	America/Montserrat	Atlantic Standard Time
73	America/Port_of_Spain	Atlantic Standard Time
74	America/Puerto_Rico	Atlantic Standard Time
75	America/Santiago	Chile Time
76	America/Santo_Domingo	Atlantic Standard Time
77	America/St_Kitts	Atlantic Standard Time
78	America/St_Lucia	Atlantic Standard Time
79	America/St_Thomas	Atlantic Standard Time

80	America/St_Vincent	Atlantic Standard Time
81	America/Thule	Atlantic Standard Time
82	America/Tortola	Atlantic Standard Time
83	Antarctica/Palmer	Chile Time
84	Atlantic/Bermuda	Atlantic Standard Time
85	Atlantic/Stanley	Falkland Is. Time
86	PRT	Atlantic Standard Time
87	America/St_Johns	Newfoundland Standard Time
88	CNT	Newfoundland Standard Time
89	AGT	Argentine Time
90	America/Buenos_Aires	Argentine Time
91	America/Cayenne	French Guiana Time
92	America/Fortaleza	Brazil Time
93	America/Godthab	Western Greenland Time
94	America/Miquelon	Pierre & Miquelon Standard Time
95	America/Montevideo	Uruguay Time
96	America/Paramaribo	Suriname Time
97	America/Sao_Paulo	Brazil Time
98	BET	Brazil Time
99	America/Noronha	Fernando de Noronha Time
100	Atlantic/South_Georgia	South Georgia Standard Time
101	America/Scoresbysund	Eastern Greenland Time
102	Atlantic/Azores	Azores Time
103	Atlantic/Cape_Verde	Cape Verde Time
104	Atlantic/Jan_Mayen	Eastern Greenland Time
105	Africa/Abidjan	Greenwich Mean Time
106	Africa/Accra	Greenwich Mean Time
107	Africa/Banjul	Greenwich Mean Time
108	Africa/Bissau	Greenwich Mean Time
109	Africa/Casablanca	Western European Time
110	Africa/Conakry	Greenwich Mean Time

111	Africa/Dakar	Greenwich Mean Time
112	Africa/Freetown	Greenwich Mean Time
113	Africa/Lome	Greenwich Mean Time
114	Africa/Monrovia	Greenwich Mean Time
115	Africa/Nouakchott	Greenwich Mean Time
116	Africa/Ouagadougou	Greenwich Mean Time
117	Africa/Sao_Tome	Greenwich Mean Time
118	Africa/Timbuktu	Greenwich Mean Time
119	Atlantic/Canary	Western European Time
120	Atlantic/Faeroe	Western European Time
121	Atlantic/Reykjavik	Greenwich Mean Time
122	Atlantic/St_Helena	Greenwich Mean Time
123	Europe/Dublin	Greenwich Mean Time
124	Europe/Lisbon	Western European Time
125	Europe/London	Greenwich Mean Time
126	GMT	Greenwich Mean Time
127	UTC	Coordinated Universal Time
128	WET	Western European Time
129	Africa/Algiers	Central European Time
130	Africa/Bangui	Western African Time
131	Africa/Douala	Western African Time
132	Africa/Kinshasa	Western African Time
133	Africa/Lagos	Western African Time
134	Africa/Libreville	Western African Time
135	Africa/Luanda	Western African Time
136	Africa/Malabo	Western African Time
137	Africa/Ndjamena	Western African Time
138	Africa/Niamey	Western African Time
139	Africa/Porto-Novo	Western African Time
140	Africa/Tunis	Central European Time
141	Africa/Windhoek	Western African Time

142	ECT	Central European Time
143	Europe/Amsterdam	Central European Time
144	Europe/Andorra	Central European Time
145	Europe/Belgrade	Central European Time
146	Europe/Berlin	Central European Time
147	Europe/Brussels	Central European Time
148	Europe/Budapest	Central European Time
149	Europe/Copenhagen	Central European Time
150	Europe/Gibraltar	Central European Time
151	Europe/Luxembourg	Central European Time
152	Europe/Madrid	Central European Time
153	Europe/Malta	Central European Time
154	Europe/Monaco	Central European Time
155	Europe/Oslo	Central European Time
156	Europe/Paris	Central European Time
157	Europe/Prague	Central European Time
158	Europe/Rome	Central European Time
159	Europe/Stockholm	Central European Time
160	Europe/Tirane	Central European Time
161	Europe/Vaduz	Central European Time
162	Europe/Vienna	Central European Time
163	Europe/Warsaw	Central European Time
164	Europe/Zurich	Central European Time
165	ART	Eastern European Time
166	Africa/Blantyre	Central African Time
167	Africa/Bujumbura	Central African Time
168	Africa/Cairo	Eastern European Time
169	Africa/Gaborone	Central African Time
170	Africa/Harare	Central African Time
171	Africa/Johannesburg	South Africa Standard Time
172	Africa/Kigali	Central African Time

173	Africa/Lubumbashi	Central African Time
174	Africa/Lusaka	Central African Time
175	Africa/Maputo	Central African Time
176	Africa/Maseru	South Africa Standard Time
177	Africa/Mbabane	South Africa Standard Time
178	Africa/Tripoli	Eastern European Time
179	Asia/Amman	Eastern European Time
180	Asia/Beirut	Eastern European Time
181	Asia/Damascus	Eastern European Time
182	Asia/Jerusalem	Israel Standard Time
183	Asia/Nicosia	Eastern European Time
184	CAT	Central African Time
185	EET	Eastern European Time
186	Europe/Athens	Eastern European Time
187	Europe/Bucharest	Eastern European Time
188	Europe/Chisinau	Eastern European Time
189	Europe/Helsinki	Eastern European Time
190	Europe/Istanbul	Eastern European Time
191	Europe/Kaliningrad	Eastern European Time
192	Europe/Kiev	Eastern European Time
193	Europe/Minsk	Eastern European Time
194	Europe/Riga	Eastern European Time
195	Europe/Simferopol	Eastern European Time
196	Europe/Sofia	Eastern European Time
197	Europe/Tallinn	Eastern European Time
198	Europe/Vilnius	Eastern European Time
199	Africa/Addis_Ababa	Eastern African Time
200	Africa/Asmera	Eastern African Time
201	Africa/Dar_es_Salaam	Eastern African Time
202	Africa/Djibouti	Eastern African Time
203	Africa/Kampala	Eastern African Time

204	Africa/Khartoum	Eastern African Time
205	Africa/Mogadishu	Eastern African Time
206	Africa/Nairobi	Eastern African Time
207	Asia/Aden	Arabia Standard Time
208	Asia/Baghdad	Arabia Standard Time
209	Asia/Bahrain	Arabia Standard Time
210	Asia/Kuwait	Arabia Standard Time
211	Asia/Qatar	Arabia Standard Time
212	Asia/Riyadh	Arabia Standard Time
213	EAT	Eastern African Time
214	Europe/Moscow	Moscow Standard Time
215	Indian/Antananarivo	Eastern African Time
216	Indian/Comoro	Eastern African Time
217	Indian/Mayotte	Eastern African Time
218	Asia/Tehran	Iran Time
219	MET	Iran Time
220	Asia/Aqtau	Aqtau Time
221	Asia/Baku	Azerbaijan Time
222	Asia/Dubai	Gulf Standard Time
223	Asia/Muscat	Gulf Standard Time
224	Asia/Tbilisi	Georgia Time
225	Asia/Yerevan	Armenia Time
226	Europe/Samara	Samara Time
227	Indian/Mahe	Seychelles Time
228	Indian/Mauritius	Mauritius Time
229	Indian/Reunion	Reunion Time
230	NET	Armenia Time
231	Asia/Kabul	Afghanistan Time
232	Asia/Aqtobe	Aqtobe Time
233	Asia/Ashgabat	Turkmenistan Time
234	Asia/Ashkhabad	Turkmenistan Time

235	Asia/Bishkek	Kirgizstan Time
236	Asia/Dushanbe	Tajikistan Time
237	Asia/Karachi	Pakistan Time
238	Asia/Tashkent	Uzbekistan Time
239	Asia/Yekaterinburg	Yekaterinburg Time
240	Indian/Chagos	Indian Ocean Territory Time
241	Indian/Kerguelen	French Southern & Antarctic Lands Time
242	Indian/Maldives	Maldives Time
243	PLT	Pakistan Time
244	Asia/Calcutta	India Standard Time
245	IST	India Standard Time
246	Asia/Katmandu	Nepal Time
247	Antarctica/Mawson	Mawson Time
248	Asia/Almaty	Alma-Ata Time
249	Asia/Colombo	Sri Lanka Time
250	Asia/Dacca	Bangladesh Time
251	Asia/Dhaka	Bangladesh Time
252	Asia/Novosibirsk	Novosibirsk Time
253	Asia/Thimbu	Bhutan Time
254	Asia/Thimphu	Bhutan Time
255	BST	Bangladesh Time
256	Asia/Rangoon	Myanmar Time
257	Indian/Cocos	Cocos Islands Time
258	Asia/Bangkok	Indochina Time
259	Asia/Jakarta	Java Time
260	Asia/Krasnoyarsk	Krasnoyarsk Time
261	Asia/Phnom_Penh	Indochina Time
262	Asia/Saigon	Indochina Time
263	Asia/Vientiane	Indochina Time
264	Indian/Christmas	Christmas Island Time

265	VST	Indochina Time
266	Antarctica/Casey	Western Standard Time (Australia)
267	Asia/Brunei	Brunei Time
268	Asia/Hong_Kong	Hong Kong Time
269	Asia/Irkutsk	Irkutsk Time
270	Asia/Kuala_Lumpur	Malaysia Time
271	Asia/Macao	China Standard Time
272	Asia/Manila	Philippines Time
273	Asia/Shanghai	China Standard Time
274	Asia/Singapore	Singapore Time
275	Asia/Taipei	China Standard Time
276	Asia/Ujung_Pandang	Borneo Time
277	Asia/Ulaanbaatar	Ulaanbaatar Time
278	Asia/Ulan_Bator	Ulaanbaatar Time
279	Australia/Perth	Western Standard Time (Australia)
280	CTT	China Standard Time
281	Asia/Jayapura	Jayapura Time
282	Asia/Pyongyang	Korea Standard Time
283	Asia/Seoul	Korea Standard Time
284	Asia/Tokyo	Japan Standard Time
285	Asia/Yakutsk	Yakutsk Time
286	JST	Japan Standard Time
287	Pacific/Palau	Palau Time
288	ACT	Central Standard Time (Northern Territory)
289	Australia/Adelaide	Central Standard Time (South Australia)
290	Australia/Broken_Hill	Central Standard Time (South Australia/New South Wales)
291	Australia/Darwin	Central Standard Time (Northern Territory)

292	AET	Eastern Standard Time (New South Wales)
293	Antarctica/DumontD'Urville	Dumont-d'Urville Time
294	Asia/Vladivostok	Vladivostok Time
295	Australia/Brisbane	Eastern Standard Time (Queensland)
296	Australia/Hobart	Eastern Standard Time (Tasmania)
297	Australia/Sydney	Eastern Standard Time (New South Wales)
298	Pacific/Guam	Chamorro Standard Time
299	Pacific/Port_Moresby	Papua New Guinea Time
300	Pacific/Saipan	Chamorro Standard Time
301	Pacific/Truk	Truk Time
302	Australia/Lord_Howe	Load Howe Standard Time
303	Asia/Magadan	Magadan Time
304	Pacific/Efate	Vanuatu Time
305	Pacific/Guadalcanal	Solomon Is. Time
306	Pacific/Kosrae	Kosrae Time
307	Pacific/Noumea	New Caledonia Time
308	Pacific/Ponape	Ponape Time
309	SST	Solomon Is. Time
310	Pacific/Norfolk	Norfolk Time
311	Antarctica/McMurdo	New Zealand Standard Time
312	Asia/Anadyr	Anadyr Time
313	Asia/Kamchatka	Petropavlovsk-Kamchatski Time
314	NST	New Zealand Standard Time
315	Pacific/Auckland	New Zealand Standard Time
316	Pacific/Fiji	Fiji Time
317	Pacific/Funafuti	Tuvalu Time
318	Pacific/Majuro	Marshall Islands Time
319	Pacific/Nauru	Nauru Time
320	Pacific/Tarawa	Gilbert Is. Time

321	Pacific/Wake	Wake Time
322	Pacific/Wallis	Wallis & Futuna Time
323	Pacific/Chatham	Chatham Standard Time
324	Pacific/Enderbury	Phoenix Is. Time
325	Pacific/Tongatapu	Tonga Time
326	Pacific/Kiritimati	Line Is. Time

Index

A

access control list, *See* ACL

ACL (access control list)

- arguments for ACL tags, JSP 822
- arguments for ACL tags, XML 57
- copying fields from ACLs, JSP 826
- copying fields from ACLs, XML 61
- creating, JSP 823
- creating, XML 58
- deleting, JSP 824
- deleting, XML 59
- gathering fields, JSP 825
- gathering fields, XML 60
- loading, JSP 829
- loading, XML 63
- retrieving lists, JSP 827
- retrieving lists, XML 62
- saving, JSP 830
- saving, XML 65
- scattering fields, JSP 831
- setting fields, JSP 832
- setting fields, XML 67
- verifying users, XML 343

address logs

- exporting, JSP 882
- exporting, XML 106

Analysis Connector

- XML tags 779

application logic

- calling application logic, XML 145
- calling EvalServer applogic, XML 179

- calling TreeManager applogic, JSP 958
- calling treemanager applogic, XML 312
- events, XML 68
- invoking, XML 71

applications

- authorizing users, JSP 886
- authorizing users, XML 112

approot, JSP 1071

approot, XML 465, 467

arguments

- encoding arguments in URLs, JSP 1075
- encoding arguments in URLs, XML 469
- specifying arguments for ics indexadd tags, JSP 916, 954
- specifying arguments for LOCALE tags, JSP 979
- specifying arguments for LOCALE tags, XML 231
- specifying arguments for parent tags, JSP 858
- specifying arguments for parent tags, XML 191

asset types

- adding to lists, XML 503
- clearing lists, XML 503
- deleting from lists, XML 503
- gathering variables, XML 504
- listing, XML 504
- scattering variables, XML 505
- setting list descriptions, XML 506
- verifying in lists, XML 504
- verifying list descriptions, XML 505

AssetMaker

- creating asset new properties, XML 1247
- naming new asset types, XML 1245
- overview, XML 1244
- properties, XML 1247
- restrictions and dependencies for BIGINT datatypes, XML 1272
- restrictions and dependencies for CHAR datatypes, XML 1268
- restrictions and dependencies for datatypes, XML 1268
- restrictions and dependencies for DOUBLE datatypes, XML 1273
- restrictions and dependencies for INTEGER datatypes, XML 1271
- restrictions and dependencies for SMALLINT datatypes, XML 1270
- restrictions and dependencies for TIMESTAMP datatypes, XML 1274
- restrictions and dependencies for VARCHAR datatypes, XML 1269
- setting default values, XML 1245
- specifying asset storage properties, XML 1248
- specifying data entry for search forms, XML 1260
- specifying data entry properties, XML 1249
- specifying search result displays, XML 1267
- XML tags 1243

AssetPublication table

- asset methods, JSP 1022
- asset methods, XML 373

AssetRelationTree table, JSP 1022

AssetRelationTree table, XML 373

assets

- adding child assets, JSP 1023
- adding child assets, XML 374
- asset storage root directories, XML 368
- building assetsets from asset lists, JSP 1104
- building assetsets from asset lists, XML 668
- building assetsets, JSP 1102
- building assetsets, XML 667
- calculating asset ratings, JSP 1207
- calculating asset ratings, XML 766
- counting, JSP 1098

- creating entries for sites, XML 413
- creating object instances, JSP 1028
- creating object instances, XML 380
- creating objects, JSP 1045
- creating objects, XML 401
- exporting to XML, XML 382, 1030
- gathering variables, JSP 1024, 1031, 1040
- gathering variables, XML 376, 384, 394
- listing asset type attributes, XML 381, 1029
- listing asset types, XML 410, 1053
- listing attributes and specs, JSP 1041
- listing attributes and specs, XML 397
- listing child asset types, JSP 1028
- listing child asset types, XML 379
- listing legal asset subtypes, XML 387, 1034
- listing publish dependencies, XML 388, 1035
- listing referenced assets, XML 404, 1048
- listing sites an asset is shared with, XML 415, 1057
- listing, JSP 1042, 1099
- listing, XML 398, 663
- loading asset parent pages, JSP 1055
- loading asset parent pages, XML 414
- loading assets, JSP 1043, 1045
- loading assets, XML 399, 401
- loading previous revisions, XML 403, 1047
- parsing XML, JSP 1040
- parsing XML, XML 394
- recalculating asset ratings, JSP 1202
- recalculating asset ratings, XML 762
- relationships between page assets, JSP 1083
- relationships between page assets, XML 489
- removing child relations, XML 417, 1058
- removing entries from AssetRelationTree, XML 405, 1048
- removing from workflow groups 588
- retrieving asset IDs for workflow, XML 532
- retrieving asset subtypes, JSP 1037
- retrieving asset subtypes, XML 391
- retrieving asset types in workflow,

- XML 532
 - retrieving asset types, JSP 1033
 - retrieving asset types, XML 387
 - retrieving assets that match recommendations, JSP 1208
 - retrieving assets that match recommendations, XML 767
 - retrieving child assets, JSP 1025
 - retrieving child assets, XML 377
 - retrieving export file names, XML 1033
 - retrieving page asset names, JSP 1086
 - retrieving page asset names, XML 492
 - retrieving page asset node IDs, JSP 1036
 - retrieving page asset node IDs, XML 390
 - retrieving recommended assets, XML 768
 - retrieving template names, JSP 1039
 - retrieving template names, XML 392
 - retrieving template rootelements, XML 391, 1038
 - retrieving values from asset fields, JSP 1032
 - retrieving values from asset fields, XML 386
 - saving assets, XML 407, 1050
 - saving, XML 406, 1049
 - scattering, JSP 1051
 - scattering, XML 409
 - searching, JSP 1105
 - searching, XML 669
 - setting fields, XML 410, 412, 1053, 1054
 - setting status to void, XML 417, 1058
 - specifying child assets, JSP 1026
 - specifying child assets, XML 377
 - streaming field values, XML 416, 1057
 - verifying publish-in-progress, JSP 1024
 - verifying publish-in-progress, XML 375
- assetsets
- building from asset lists, JSP 1104
 - building from asset lists, XML 668
 - building, JSP 1102
 - building, XML 667
 - counting attribute values, JSP 1100
 - counting attribute values, XML 664
 - creating assetset objects using search criteria, JSP 1105
 - creating assetset objects using search criteria, XML 669
 - creating empty assetset objects, JSP 1103
 - creating empty assetset objects, XML 668
 - getting multiple attribute values, JSP 1101
 - getting multiple attribute values, XML 666
 - overview, JSP 1098
 - overview, XML 662
 - recalculating asset ratings, JSP 1202
 - recalculating asset ratings, XML 762
 - retrieving attribute values, JSP 1100
 - retrieving attribute values, XML 665
- associations
- named associations, JSP 1022
 - named associations, XML 373
- attributes
- adding to existing entries ,JSP 833
 - adding to existing entries, XML 154
 - counting attribute values, JSP 1100
 - counting attribute values, XML 664
 - deleting, JSP 849
 - deleting, XML 168
 - getting multiple attribute values, JSP 1101
 - getting multiple attribute values, XML 666
 - replacing attribute values, JSP 853
 - replacing attribute values, XML 172
 - retrieving attributes, JSP 842
 - retrieving attributes, XML 162
 - retrieving scalar visitor attributes, JSP 1216
 - retrieving scalar visitor attributes, XML 775
 - scattering attribute values, JSP 1101
 - scattering attribute values, XML 666
- B**
- billing 1279, 1280
 - blob (binary large object)
 - caching 792
 - caching, JSP 1222
 - caching, XML 792
 - creating URLs, JSP 1070
 - creating URLs, XML 463
 - loading, JSP 1107
 - loading, XML 672

- retrieving column names, JSP 1107
- retrieving column names, XML 672
- retrieving column values, JSP 1106
- retrieving column values, XML 671
- retrieving from CS-Satellite cache, JSP 1077
- retrieving from CS-Satellite cache, XML 471
- retrieving table names, JSP 1106
- retrieving table names, XML 671
- retrieving, JSP 1220
- retrieving, XML 790
- browsers
 - executing browser requests, XML 353
- built-ins
 - displaying values, XML 149
- buyers
 - buyer parameter names 1308
 - changing payment account information, JSP 1150
 - changing payment account information, XML 711
 - modifying buyer data, JSP 1149
 - modifying buyer data, XML 710
 - registered buyers, JSP 1132, 1143
 - registered buyers, XML 694
 - registered vs. walk-in 1277
 - registering new buyers, JSP 1151
 - registering new buyers, XML 712
 - retrieving buyers from Commerce Connector, JSP 1149
 - retrieving buyers from Commerce Connector, XML 710
 - setting single parameters, JSP 1152
 - setting single parameters, XML 713
 - setting specified parameters, JSP 1152
 - setting specified parameters, XML 713
 - walk-in buyers 1277
 - walk-in buyers, JSP 1132
 - walk-in buyers, XML 694

C

- cache
 - clearing resultsets, XML 186
 - deleting element cache, XML 459
 - deleting page cache, JSP 883
 - deleting page cache, XML 108
 - disabling, JSP 913, 1011

- disabling, XML 176
- flushing individual items from cache, JSP 1224
- flushing individual items from cache, XML 795
- flushing internal memory for tables, JSP 917
- flushing, JSP 883
- flushing, XML 107
- retrieving blobs and tables from CS-Satellite cache, JSP 1077
- retrieving blobs and tables from CS-Satellite cache, XML 471
- retrieving blobs, JSP 1220
- retrieving blobs, XML 790
- retrieving filenames, XML 354
- cart sets
 - See also* shopping carts
 - adding new carts, JSP 1132
 - adding new carts, XML 694
 - creating empty cart sets, JSP 1134
 - creating empty cart sets, XML 695
 - deleting carts, JSP 1133
 - deleting, XML 695
 - overview, JSP 1132
 - overview, XML 694
 - retrieving and creating, JSP 1134
 - retrieving and creating, XML 695
 - retrieving carts, JSP 1134
 - retrieving carts, XML 695
 - retrieving identifiers for carts, JSP 1133
 - retrieving identifiers for carts, XML 694
 - retrieving visitor cart sets, JSP 1134
 - retrieving visitor cart sets, XML 695

CGI

- paths, XML 465, 467
- characters
 - conversions, XML 150
 - encoding into URLs, XML 459
 - special characters, XML 150

child assets

- adding, JSP 1023
- adding, XML 374
- listing asset types, JSP 1028
- listing asset types, XML 379
- retrieving, JSP 1025
- retrieving, XML 377

CODE

- as association name with XML tags 378
- columns
 - constructing lists, XML 301
 - defining, JSP 863
 - defining, XML 76
 - retrieving column names, JSP 1174
 - retrieving column names, XML 734
 - verifying columns, XML 223
- Commerce Connector
 - checking carts against Transact, JSP 1112
 - checking carts against Transact, XML 676
 - creating new payment accounts for csuser objects, JSP 1144
 - creating new payment accounts for csuser objects, XML 705
 - obtaining transaction identifiers, JSP 1128
 - obtaining transaction identifiers, XML 690, 691
 - required cart parameters, JSP 1110
 - required cart parameters, XML 674
 - retrieving buyers, JSP 1149
 - retrieving buyers, XML 710
 - retrieving user information, JSP 1143
 - retrieving user information, XML 704
 - sending carts to Transact, JSP 1114
 - sending carts to Transact, XML 678
- commerce context
 - overview, JSP 1135
 - overview, XML 697
- conditionals
 - else clauses, JSP 913
 - if conditional, JSP 921
 - if, XML 206
 - then clause, JSP 955
 - then, XML 206
 - where clause, JSP 953
 - where clause, XML 204, 299
- constraints
 - and Searchstate methods, JSP 1175
 - and Searchstate methods, XML 735
 - definition 735
- constructing resultset lists, JSP 1170
- constructing resultsets, XML 730
- content
 - flushing body content, JSP 857
- Content Server
 - JSP tags 817
 - XML tags 51
- Content Server Direct
 - JSP tags 1021
 - XML tags 371
- Content Server Direct Advantage
 - parameters 1275
 - XML tags 661, 697, 1135
- Content Server Engage
 - JSP tags 1201
 - XML tags 761
- Content Server Satellite
 - JSP tags 1219
 - XML tags 789
- cookies
 - creating for Transact recognition, JSP 1139
 - creating for Transact recognition, XML 701
 - deleting, XML 273
 - inserting values, JSP 1139
 - inserting values, XML 701
 - setting, JSP 1223
 - setting, XML 291, 794
- counters
 - creating, XML 293
 - decrementing, XML 207
 - deleting, XML 274
 - displaying values, XML 149
 - incrementing, XML 207
 - verifying counters, XML 223
- currency
 - converting doubles to currency strings, JSP 1155
 - converting doubles to currency strings, XML 716
 - converting strings to decimals, JSP 1157
 - converting strings to decimals, XML 717
 - creating currency objects for ISO codes, JSP 1154
 - creating currency objects for ISO codes, XML 715
 - operations, JSP 1154
 - operations, XML 715
 - retrieving currency symbols, JSP 1156
 - retrieving currency symbols, XML 717

- retrieving ISO codes, JSP 1156
- retrieving ISO codes, XML 716
- rounding down, JSP 1158
- rounding down, XML 718
- rounding up, JSP 1158
- rounding up, XML 718
- rounding values, JSP 1157
- rounding values, XML 717

D

data

- flushing data streams, JSP 917
- retrieving from tables, XML 783

database tables

- adding multiple nodes to tree tables, JSP 960
- adding multiple nodes to tree tables, XML 316
- adding multiple rows, JSP 867
- adding multiple rows, XML 83
- adding nodes to tree tables, JSP 959
- adding nodes to tree tables, XML 314
- adding rows, JSP 865, 868
- adding rows, XML 80, 85
- adding, JSP 881
- adding, XML 105
- AssetPublication, JSP 1022
- AssetPublication, XML 373
- AssetRelationTree, JSP 1022
- AssetRelationTree, XML 373
- assets, JSP 1022
- assets, XML 373
- copying rows, JSP 889
- copying rows, XML 116
- creating Analysis Connector tables, XML 781
- creating new revision tracking records, XML 143
- creating object support tables, XML 250
- creating tree tables, JSP 963
- creating tree tables, XML 320
- creating, JSP 871
- creating, XML 78, 89
- deleting Analysis Connector tables, XML 781
- deleting multiple rows, JSP 876
- deleting multiple rows, XML 97
- deleting object support tables, JSP 988
- deleting object support tables, XML 246

- deleting rows, JSP 874
- deleting rows, XML 78, 95
- deleting tree tables, JSP 966
- deleting tree tables, XML 324
- deleting, JSP 877, 881
- deleting, XML 99, 105
- determining column definitions, JSP 863
- determining column definitions, XML 76
- disabling revision tracking, XML 132, 342
- enabling revision tracking for object tables, JSP 1008
- enabling revision tracking, JSP 899
- enabling revision tracking, XML 131, 265, 310
- executing queries, JSP 947
- executing queries, XML 286
- flushing internal memory, JSP 917
- getting upload folder values, XML 355
- listing pages in the SitePlanTree table, JSP 1088
- listing pages in the SitePlanTree table, XML 494
- loading site plan tree nodes, JSP 1089
- locking records in tracked tables, JSP 885
- locking records in tracked tables, XML 111
- modifying fields, JSP 878
- modifying fields, XML 101
- modifying multiple rows, JSP 880
- modifying multiple rows, XML 103
- modifying row values, XML 235
- modifying, JSP 881
- modifying, XML 78, 105
- purging rows, XML 782
- releasing locks on rows, JSP 892
- releasing locks on rows, XML 120
- releasing locks, XML 271
- replacing multiple rows, JSP 894
- replacing multiple rows, XML 124
- replacing rows, JSP 893
- replacing rows, XML 121
- retrieving data from specified tables, XML 783
- retrieving definitions, XML 784
- retrieving nodes from tables, JSP 970
- retrieving nodes from tables, XML 329

- retrieving root nodes of site plans, JSP 1091, 1093, 1094
- retrieving root nodes of site plans, XML 497, 499, 500
- retrieving tables from CS-Satellite cache, JSP 1077
- retrieving tables from CS-Satellite cache, XML 471
- reverting to previous revisions, JSP 896
- reverting to previous revisions, XML 127
- revision tracking, JSP 884
- revision tracking, XML 109
- selecting rows, JSP 881, 897
- selecting rows, XML 105, 128
- setting revisions, JSP 898
- setting revisions, XML 130
- SitePlanTree, JSP 1022
- SitePlanTree, XML 373
- stopping revision tracking, JSP 900
- unlocking, JSP 901
- unlocking, XML 133
- updating multiple, JSP 905
- updating multiple, XML 139
- updating rows, JSP 903, 907
- updating rows, XML 137, 141
- updating, JSP 902
- updating, XML 134
- verifying revision tracking, XML 228
- verifying tables, XML 222
- verifying tree tables, XML 229
- writing rows to specified tables, XML 785
- dates
 - clearing, XML 294
 - converting Java SQL date formats, JSP 945
 - converting Java SQL date formats, XML 285
 - converting to long format, XML 782, 783, 786
 - converting, JSP 1160, 1161
 - converting, XML 720, 721
 - getting date values, XML 355
 - getting the value of month, XML 362
 - getting the value of year, XML 369
 - getting value of SQL date, XML 368
 - getting values of days, XML 357
 - operations, JSP 1160
 - operations, XML 720
 - retrieving time zones, JSP 1162
 - retrieving time zones, XML 722
 - setting, XML 294
 - specifying date arguments, JSP 911
- debugging
 - enabling, JSP 912
 - writing strings to log files, JSP 936
 - writing strings to log files, XML 234
- decimals
 - converting points to strings, JSP 1163
 - converting points to strings, XML 723
 - converting strings to points, JSP 1165
 - converting strings to points, XML 725
 - creating decimal mapping objects, JSP 1163
 - creating decimal mapping objects, XML 723
 - getting decimal values, JSP 1165
 - getting maximum decimal places, JSP 1164
 - getting maximum decimal places, XML 724
 - getting minimum decimal places, JSP 1164
 - getting minimum decimal places, XML 724
 - setting maximum decimal places, JSP 1166
 - setting maximum decimal places, XML 725
 - setting minimum decimal places, JSP 1166
 - setting minimum decimal places, XML 726
 - setting thousandths separators, JSP 1166
 - setting thousandths separators, XML 726
- directives
 - for asset tags, JSP 1022
 - for render tags, JSP 1062
 - for site plan tags, JSP 1083
- directories
 - asset storage root directories, XML 368
 - retrieving child entries, JSP 837
 - retrieving child entries, XML 158
- discounts
 - applying to shopping carts, JSP 1117
 - applying to shopping carts, XML 680

- determining for promotions, JSP 1206
- determining for promotions, XML 765
- determining promotion discount rules, XML 765
- listing item discounts, JSP 1119
- listing item discounts, XML 682
- retrieving lists of discounts, JSP 1116
- retrieving lists of discounts, XML 682

E

elements

- calling, JSP 1063
- calling, XML 456
- halting element processing, XML 307
- processing, JSP 859
- processing, XML 71
- root tag, JSP 857
- verifying, XML 224

e-mail

- sending, JSP 949
- sending, XML 176, 289, 290

email

- gathering variables, XML 428
- retrieving body text, XML 429
- retrieving descriptions, XML 430
- retrieving IDs, XML 431
- retrieving object names, XML 432
- retrieving subject text, XML 433
- scattering variables, XML 434
- setting body text, XML 435
- setting description text, XML 436
- setting object names, XML 437
- setting subject text, XML 438
- translate subject text, XML 440
- translating body text, XML 439

email manager

- creating instance objects, XML 441
- deleting emails, XML 441
- listing available objects, XML 442
- loading objects, XML 443
- saving objects, XML 444

entries

- adding attributes to data types, JSP 833
- adding attributes to data types, XML 154

errdetail values 1329

errno values 1314

errors

- checking shopping cart parameter errors, JSP 1118
- checking shopping cart parameter errors, XML 681
- clearing error states, JSP 909
- clearing error states, XML 193
- enabling debugging, JSP 912
- errdetail values 1329
- errno values 1314
- error conditions 1313
- order item errors, JSP 1119
- order item errors, XML 683
- retrieving csuser errors, JSP 1147
- retrieving csuser errors, XML 708
- retrieving error codes, JSP 918
- retrieving error codes, XML 194
- testing errors, XML 224

events

- creating, XML 68
- deleting, XML 154
- disabling events, XML 175
- enabling, XML 178
- executing application logic, XML 68
- sending e-mail, XML 176

expiration

- specifying pagelet expiration, JSP 1227
- specifying pagelet expiration, XML 797

exporting

- status, XML 361

exporting to XML, XML 382, 1030

expressions

- calculating values, JSP 1108
- calculating values, XML 673

F

fields

- copying from ACLs, JSP 826
- copying from ACLs, XML 61
- copying from objects, JSP 991
- copying from objects, XML 249
- gathering into ACLs, XML 60, 825
- gathering into objects, JSP 990
- gathering into objects, XML 248
- scattering asset fields, JSP 1051
- scattering asset fields, XML 409
- scattering fields from objects, JSP 1003
- scattering fields from objects, XML 260

- setting into ACLs, JSP 832
- setting into ACLs, XML 67
- setting into objects, JSP 1004
- setting into objects, XML 261
- streaming field values from objects, JSP 1007
- streaming field values from objects, XML 264
- files
 - constructing lists, XML 184
 - deleting cache files, XML 459
 - deleting log files, JSP 872
 - deleting log files, XML 92
 - deleting, XML 151
 - inserting, JSP 932
 - inserting, XML 221
 - loading property files, XML 229
 - setting property files, XML 279
 - transact.ini 677, 1112
 - writing strings to any file, XML 350
 - writing strings to log files, JSP 936
 - writing strings to log files, XML 234
- flushing
 - individual items 1224
 - individual items from cache, XML 795
- ft.approot property, JSP 1071
- ft.approot property, XML 465, 467
- ft.cgipath property, XML 465, 467
- G**
 - groups
 - adding members, JSP 834
 - adding members, XML 156
 - listing members, JSP 844
 - listing members, XML 164
 - listing memberships, XML 165
 - listing memberships, XML 845
 - removing members from groups, JSP 851
 - removing members from groups, XML 170
- H**
 - hash tables
 - adding values, JSP 1167
 - adding values, XML 727
 - checking data, JSP 1168
 - checking data, XML 728
 - checking values, JSP 1167
 - checking values, XML 727
 - creating, JSP 1168
 - creating, XML 728
 - writing table values to strings, JSP 1169
 - writing table values to strings, XML 729
- hierarchy
 - site plan, JSP 1036
 - site plan, XML 390, 489
- history types
 - summing field entries, JSP 1215
 - summing field entries, XML 774
- HTTP syntax
 - submitting requests, XML 270
- I**
 - indexes
 - creating, XML 211
 - deleting, JSP 926
 - deleting, XML 213
 - finding, XML 216
 - removing entries, JSP 929
 - removing entries, XML 217
 - replacing entries, JSP 930
 - replacing entries, XML 219
 - searching, JSP 943
 - searching, XML 282
 - verifying, JSP 927
 - verifying, XML 214
 - IP addresses
 - exporting logs, JSP 882
 - exporting logs, XML 106
 - items
 - adding index names and rich-text expressions to search criteria, XML 738
 - setting prices and quantities, XML 692
- J**
 - Java classes
 - calling, XML 73
 - Java VM
 - generating unique IDs, XML 369
- L**
 - list items
 - displaying values, XML 149
 - list objects

- overview, JSP 1170
- overview, XML 730
- listing nodes, JSP 967
- listing nodes, XML 325
- lists
 - adding rows, JSP 1170
 - adding rows, XML 730
 - building assetsets from asset lists, JSP 1104
 - building assetsets from asset lists, XML 668
 - calculating asset ratings from a named list, XML 766
 - constructing file lists, XML 184
 - constructing, XML 301
 - converting list objects to resultsets, JSP 1172
 - converting list objects to resultsets, XML 731
 - converting lists to strings, JSP 1191
 - converting lists to strings, XML 751
 - copying from one list to another, JSP 910
 - copying from one list to another, XML 147
 - copying, JSP 910
 - copying, XML 147
 - creating lists of pages, JSP 1088
 - creating lists of pages, XML 494
 - creating lists of promotions for eligible visitors, XML 766
 - creating, JSP 1171
 - creating, XML 731
 - flushing, XML 186
 - iterating, JSP 934
 - listing abandoned shopping carts, JSP 1135
 - listing abandoned shopping carts, XML 697
 - listing ancestors to the CS.Property.dir.peopleParent property, JSP 847
 - listing ancestors to the CS.Property.dir.peopleParent property, XML 167
 - listing assets, JSP 1042, 1099
 - listing assets, XML 398, 663
 - listing child pages (nodes), JSP 1084
 - listing child pages (nodes), XML 489
 - listing item discount totals, JSP 1119
 - listing item discount totals, XML 682
 - listing members of groups, JSP 844
 - listing members of groups, XML 164
 - listing trees, JSP 972
 - listing trees, XML 332
 - listing visitor segments, XML 764
 - removing list objects, XML 276
 - retrieving and listing single values from site plan nodes, JSP 1085
 - retrieving and listing single values from site plan nodes, XML 491
 - retrieving child nodes, JSP 968
 - retrieving child nodes, XML 326
 - retrieving lists of ACLs, JSP 827
 - retrieving lists of ACLs, XML 62
 - retrieving lists of assets, JSP 1042
 - retrieving lists of assets, XML 398
 - retrieving lists of discounts, JSP 1116
 - retrieving lists of discounts, XML 682
 - retrieving names of columns, JSP 1174
 - retrieving names of columns, XML 734
 - retrieving object root node lists, JSP 1000
 - retrieving object root node lists, XML 258
 - retrieving, JSP 933
 - retrieving, XML 197
 - setting rows, XML 295
 - specifying list criteria, JSP 1026
 - specifying lists for mirroring, JSP 938
 - verifying lists, XML 227
 - verifying tables, XML 222
 - verifying, XML 223
- loading
 - assets, JSP 1043
 - assets, XML 399
 - parent pages, JSP 1055
 - parent pages, XML 414
 - site plan, JSP 1089
 - site plan, XML 495
- locales
 - creating descriptors, JSP 1173
 - creating descriptors, XML 733
- localization
 - returning localized values, JSP 979
 - returning localized values, XML 232
 - specifying arguments for LOCALE tags, JSP 979

- specifying arguments for LOCALE tags, XML 231
- TimeZoneIDs for use with DateFormat tags 1333
- logical conditions
 - if, JSP 921
 - if, XML 206
- logs
 - deleting log files, JSP 872
 - deleting log files, XML 92
- M**
- members
 - adding to groups, JSP 834
 - adding to groups, XML 156
 - listing members of groups, JSP 844
 - listing members of groups, XML 164
 - removing from groups, JSP 851
 - removing from groups, XML 170
- memory
 - caching to memory, XML 354
- Message Authentication Code (MAC), JSP 1139
- Message Authentication Code (MAC), XML 701
- mimetypes, JSP 1070, 1077
- mimetypes, XML 464, 471
- mirroring
 - aborting mirror operations, JSP 887
 - aborting mirror operations, XML 114
 - copying rows, JSP 889
 - copying rows, XML 116
 - mirrorlists, JSP 936
 - specifying subtrees, JSP 938
- N**
- names
 - named associations, JSP 1022
 - named associations, XML 373
- nodes
 - adding multiple nodes to tree tables, JSP 960
 - adding multiple nodes to tree tables, XML 316
 - adding to tree tables, JSP 959
 - adding to tree tables, XML 314
 - copying tree nodes, JSP 962
 - copying tree nodes, XML 319
 - creating lists of pages, JSP 1088
 - creating lists of pages, XML 494
 - deleting from trees, JSP 964
 - deleting from trees, XML 321
 - deleting multiple nodes, JSP 965
 - deleting multiple nodes, XML 322
 - listing ancestor nodes, JSP 974
 - listing ancestor nodes, XML 335
 - listing child nodes (pages), JSP 1084
 - listing child nodes, JSP 968
 - listing child nodes, XML 326, 489
 - listing, JSP 967
 - listing, XML 325
 - moving, JSP 973
 - moving, XML 334
 - parent, JSP 1086
 - parent, XML 491
 - removing from tree, JSP 1095
 - retrieving child nodes, JSP 985
 - retrieving node parent records, JSP 971
 - retrieving node parent records, XML 330
 - retrieving nodes from tables, JSP 970
 - retrieving nodes from tables, XML 329
 - retrieving object nodes, XML 243
 - retrieving page asset names, JSP 1086
 - retrieving page asset names, XML 492
 - retrieving page asset node IDs, JSP 1036
 - retrieving page asset node IDs, XML 390
 - retrieving root nodes of site plans, JSP 1091, 1094
 - retrieving root nodes of site plans, XML 497, 500
 - retrieving root nodes, JSP 1000
 - retrieving root nodes, XML 258
 - retrieving site plan root nodes, JSP 1093
 - retrieving site plan root nodes, XML 499
 - setting object node parents, JSP 1006
 - setting object node parents, XML 263
 - setting objects in nodes , JSP 975
 - setting objects in nodes , XML 262, 337
 - setting objects in nodes, JSP 1005
 - site nodes, JSP 1036, 1089
 - site nodes, XML 390, 495
 - validating nodes, JSP 977
 - validating nodes, XML 338
 - verifying node paths, JSP 978

verifying node paths, XML 340

O

object type, *See* descriptions of individual tags

objects

adding attribute constraints to searchstate objects, JSP 1175

adding rows to lists, JSP 1170

adding rows to lists, XML 730

adding values to hash table objects, JSP 1167

adding values to hash table objects, XML 727

checking data in hash tables, JSP 1168

checking data in hash tables, XML 728

checking in, JSP 983

checking in, XML 240

checking out, JSP 984

checking out, XML 241

checking values in hash tables, JSP 1167

checking values in hash tables, XML 727

clearing csuser object data and parameters, JSP 1144

clearing csuser object data and parameters, XML 705

converting list objects to resultsets, JSP 1172

converting list objects to resultsets, XML 731

converting strings, JSP 1190

converting strings, XML 750

converting to strings, JSP 1188

converting to strings, XML 748

copying fields, JSP 991

copying fields, XML 249

creating csuser objects from text strings, JSP 1146

creating csuser objects from text strings, XML 707

creating currency objects for ISO codes, JSP 1154

creating currency objects for ISO codes, XML 715

creating decimal mapping objects, JSP 1163

creating decimal mapping objects, XML 723

creating directory objects, JSP 838

creating directory objects, XML 159

creating empty assetset objects, JSP 1103

creating empty assetset objects, XML 668

creating empty csuser objects, JSP 1144

creating empty csuser objects, XML 705

creating empty shopping cart objects, JSP 1114

creating empty shopping cart objects, XML 679

creating hash tables, XML 728

creating lists, JSP 1171

creating lists, XML 731

creating locale descriptors, JSP 1173

creating locale descriptors, XML 733

creating new payment accounts for csuser objects, JSP 1144

creating new payment accounts for csuser objects, XML 705

creating objects, JSP 987

creating objects, XML 244

creating payment accounts, JSP 1144

creating payment accounts, XML 705

creating shopping cart objects from text strings, XML 682

creating, JSP 1045

creating, XML 401

deinstalling, JSP 988

deinstalling, XML 246

deleting directory objects, JSP 840

deleting directory objects, XML 161

deleting payment accounts, JSP 1145

deleting payment accounts, XML 706

deleting, JSP 989

deleting, XML 247

determining whether a field is empty, JSP 994

determining whether a field is empty, XML 251

duplicating searchstate objects, JSP 1184

duplicating searchstate objects, XML 744

enabling revision tracking, JSP 1008

enabling revision tracking, XML 265

flushing object storage, JSP 1187

flushing object storage, XML 747

gathering fields into objects, JSP 990

gathering fields into objects, XML 248

- installing objects, JSP 992
- installing objects, XML 250
- loading assets using object ID, JSP 1026
- loading assets using object ID, XML 377
- loading site plan tree nodes into object memory, JSP 1089
- loading site plan tree nodes into object memory, XML 495
- loading, JSP 996
- loading, XML 254
- obtaining parameter selections for csuer objects, XML 708
- obtaining parameter selections for csuser objects, XML 707
- removing list objects, XML 276
- removing revision tracking, JSP 1010
- removing revision tracking, XML 267
- retracting checkouts, JSP 1009
- retracting checkouts, XML 266
- retrieving child nodes, JSP 985
- retrieving child nodes, XML 243
- retrieving csuser errors, JSP 1147
- retrieving csuser errors, XML 708
- retrieving lists of objects, JSP 995
- retrieving lists of objects, XML 252
- retrieving parameter selections for csuser objects, JSP 1146, 1148
- retrieving parameter selections for csuser objects, XML 709
- retrieving parameters for csuser objects, JSP 1147
- retrieving root node lists, JSP 1000
- retrieving root node lists, XML 258
- retrieving values from asset fields, JSP 1032
- retrieving values from asset fields, XML 386
- rolling back revisions, JSP 999
- rolling back revisions, XML 256
- saving objects as visitor attributes, JSP 1217
- saving objects as visitor attributes, XML 776
- saving, JSP 1002
- saving, XML 259
- scattering fields, JSP 1003
- scattering fields, XML 260
- setting fields, JSP 1004
- setting fields, XML 261
- setting object node parents, JSP 1006
- setting object node parents, XML 263
- setting objects in nodes, JSP 975, 1005
- setting objects in nodes, XML 262, 337
- specifying arguments, JSP 982
- specifying arguments, XML 239
- storing commerce context objects in persistent storage, JSP 1137
- storing commerce context objects in persistent storage, XML 699
- streaming field values from objects, JSP 1007
- streaming field values from objects, XML 264
- verifying object names, XML 227
- writing csuser objects to text strings, JSP 1153
- writing csuser objects to text strings, XML 714
- order items
 - adding to shopping carts, JSP 1110
 - adding to shopping carts, XML 675
- output stream
 - inserting files, JSP 932
 - inserting files, XML 221
- P**
 - page assets
 - child 490
 - relationships between, XML 390
 - retrieving page asset names, JSP 1086
 - retrieving page asset names, XML 492
 - pagelets
 - calling and rendering, JSP 457, 1064
 - calling, JSP 1079
 - calling, XML 473
 - marking, JSP 1228
 - marking, XML 799
 - retrieving, JSP 1227
 - retrieving, XML 797
 - specifying cache expiration, JSP 1227
 - specifying cache expiration, XML 797
 - pages
 - caching, XML 354
 - calling and rendering, JSP 1064
 - calling and rendering, XML 457
 - calling, JSP 1079
 - calling, XML 473

- export status, XML 361
- listing child pages (nodes), JSP 1084
- listing child pages (nodes), XML 489
- listing pages in the SitePlanTree table, JSP 1088
- listing pages in the SitePlanTree table, XML 494
- parent pages, JSP 1055
- parent pages, XML 414
- retrieving page asset names, JSP 1086
- retrieving page asset names, XML 492
- parent tags
 - specifying arguments with child tags, JSP 858
 - specifying arguments with child tags, XML 191
- passwords
 - changing, JSP 1016
- payment brand identifier 1282
- promotions
 - creating lists, XML 766
 - creating, JSP 1205
 - creating, XML 763
 - determining discounts, JSP 1206
 - determining discounts, XML 765
 - determining eligibility, JSP 1211
 - determining eligibility, XML 769
 - determining promotion discount rules, XML 765
 - retrieving, JSP 1207
 - retrieving, XML 766
- properties
 - ft.approot, JSP 1071
 - ft.approot, XML 465, 467
 - listing ancestors to the CS.Property.dir.peopleParent property, JSP 847
 - listing ancestors to the CS.Property.dir.peopleParent property, XML 167
 - loading property files, XML 229
 - retrieving values, XML 363
- property files
 - setting, XML 279
- publishing
 - mirroring mirrorlists, JSP 936
 - specifying lists for mirroring, JSP 938
 - verifying assets, JSP 1024

- verifying assets, XML 375

Q

- queries
 - determining column definitions, JSP 863
 - determining column definitions, XML 76
 - displaying database table records, JSP 897
 - displaying database table records, XML 128
 - executing, JSP 947
 - executing, XML 286
 - retrieving and executing SQL queries, JSP 860
 - retrieving and executing SQL queries, XML 74

R

- ratings
 - calculating, JSP 1207
 - calculating, XML 766
- recommendations
 - retrieving assets matching recommendations, JSP 1208
 - retrieving assets matching recommendations, XML 767
 - retrieving recommended assets, XML 768
 - retrieving single assets, JSP 1210
 - retrieving single assets, XML 768
 - retrieving, JSP 1208
 - retrieving, XML 767
- records
 - committing, JSP 870
 - committing, XML 88
 - revision tracking, JSP 870
 - revision tracking, XML 88
- relationships
 - between page assets, XML 390, 489
 - unnamed, JSP 1022
 - unnamed, XML 373
- rendermode variable, JSP 1062
- rendermode variable, XML 455
- requests
 - executing browser requests, XML 353
- resultsets 730, 1170

- converting list objects to resultsets, JSP 1172
 - converting list objects to resultsets, XML 731
- returns 786
- revision tracking
 - checking in objects, JSP 983
 - checking in objects, XML 240
 - checking out objects, JSP 984
 - checking out objects, XML 241
 - committing records, JSP 870
 - committing records, XML 88
 - creating new records, XML 143
 - deleting revisions, JSP 873
 - deleting revisions, XML 93, 152
 - disabling tracking on tables, XML 342
 - enabling for object tables, JSP 1008
 - enabling for object tables, XML 265
 - enabling revision tracking on tables, XML 310
 - enabling, JSP 899
 - enabling, XML 131
 - history, XML 188
 - locking records, XML 233
 - releasing row locks, XML 271
 - removing from objects, JSP 1010
 - removing from objects, XML 267
 - retracting object checkouts, JSP 1009
 - retracting object checkouts, XML 266
 - retrieving tracked records, XML 280
 - reverting to a previous revision, XML 281
 - reverting to previous revisions, JSP 896
 - reverting to previous revisions, XML 127
 - rolling back object revisions, JSP 999
 - rolling back object revisions, XML 256
 - setting revisions, XML 298
 - stopping, JSP 900
 - stopping, XML 132
 - unlocking, XML 341
 - verifying revision tracking, XML 228
- roles
 - adding roles, XML 477
 - clearing from list, XML 478
 - deleting, XML 479
 - gathering variables, XML 480
 - listing roles, XML 480
 - scattering variables, XML 483
 - setting role list to "any", XML 483
 - verifying list descriptions, XML 482
 - verifying roles, XML 481
- root directories, XML 368
- root node, JSP 1093
- root node, XML 499
- rows
 - adding multiple rows, JSP 867
 - adding multiple rows, XML 83
 - adding, JSP 865, 881
 - adding, XML 80, 105
 - copying, JSP 889
 - copying, XML 116
 - creating, XML 78
 - deleting multiple rows in tables, JSP 876
 - deleting multiple rows in tables, XML 97
 - deleting revisions, JSP 873
 - deleting revisions, XML 93
 - deleting, JSP 874, 881
 - deleting, XML 78, 95, 105
 - locking records in tracked tables, JSP 885
 - locking records in tracked tables, XML 111
 - modifying fields, JSP 878
 - modifying fields, XML 101
 - modifying multiple rows, JSP 880, 905
 - modifying multiple rows, XML 103, 139
 - modifying values, XML 235
 - modifying, JSP 881
 - modifying, XML 78, 105
 - releasing locks, JSP 892
 - releasing locks, XML 120, 271
 - replacing multiple, JSP 894
 - replacing multiple, XML 124
 - reverting to previous revisions, XML 127, 281
 - revision history, JSP 884
 - revision history, XML 109, 188
 - selecting, JSP 881
 - selecting, XML 105
 - setting revisions, JSP 898
 - setting revisions, XML 130, 298
 - setting, XML 187, 295
 - unlocking, JSP 901

- unlocking, XML 133, 341
- updating, JSP 902, 903, 907
- updating, XML 134, 137, 141
- uploading text, JSP 868
- uploading text, XML 85

S

- scattering assets, JSP 1051

- scattering assets, XML 409

searches

- adding attribute constraints to searchstates, JSP 1175, 1180, 1181, 1182
- adding attribute constraints to searchstates, XML 739, 740, 741
- adding entries to indexes, JSP 922
- adding entries to indexes, XML 208
- adding index names and rich-text expressions to search criteria, JSP 1178
- adding index names and rich-text expressions to search criteria, XML 738
- adding nested searchstate constraints, JSP 1176
- adding nested searchstate constraints, XML 736
- adding range constraints for specific attribute names, JSP 1177
- adding range constraints for specific attribute names, XML 737
- checking searchstates for specific attributes, JSP 1184
- checking searchstates for specific attributes, XML 744
- converting Java SQL date formats, JSP 945
- converting Java SQL date formats, XML 285
- converting searchstate objects into strings, JSP 1186
- converting searchstate objects into strings, XML 746
- creating assetset objects using search criteria, JSP 1105
- creating assetset objects using search criteria, XML 669
- creating empty searchstate objects, JSP 1183
- creating empty searchstate objects, XML 742
- creating search indexes, JSP 924
- creating search indexes, XML 211

- defining index fields for Verity, JSP 915
- deleting attribute constraints, JSP 1183
- deleting attribute constraints, XML 743
- deleting indexes, JSP 926
- deleting indexes, XML 213
- duplicating searchstate objects, JSP 1184
- duplicating searchstate objects, XML 744
- initializing searches from strings, JSP 1186
- initializing searches from strings, XML 746
- removing index entries, JSP 929
- removing index entries, XML 217
- replacing indexes, JSP 930
- replacing indexes, XML 219
- searching for entries with specific attributes, JSP 855
- searching for entries with specific attributes, XML 173
- searching indexes based on queries, JSP 943
- searching indexes based on queries, XML 282
- specifying arguments for indexadd tags, JSP 954
- verifying indexes, JSP 927
- verifying indexes, XML 214

searchstates

- overview, JSP 1175
- overview, XML 735

segments

- calculating, XML 764
- determining membership, JSP 1211
- determining membership, XML 770
- listing visitor segments, JSP 1205
- listing visitor segments, XML 768
- retrieving, JSP 1209
- retrieving, XML 768

session IDs, JSP 1070

session IDs, XML 464, 466

session variables

- displaying values, XML 149

sessions

- expiration of pagelets and blobs, XML 791
- flushing session storage, JSP 1187
- flushing session storage, XML 747

- saving the contents of commerce context, JSP 1137
- saving the contents of commerce context, XML 699
- setting session variables, JSP 950
- setting session variables, XML 296
- shell commands
 - executing, XML 306
- shipper 1285
- shipping
 - charges 1302
- shopping carts
 - adding new carts, JSP 1132
 - adding order items, JSP 1110
 - adding order items, XML 675
 - applying discounts, JSP 1117
 - applying discounts, XML 680
 - CART.CHECK operations 1285
 - checking against Transact 1285
 - checking against Transact, XML 676
 - checking parameter errors, JSP 1118
 - checking parameter errors, XML 681
 - checking, JSP 1112
 - checking, XML 676
 - clearing discounts, JSP 1113
 - clearing discounts, XML 678
 - committing 1287
 - cost and payment order item parameters 1293
 - creating empty cart objects, JSP 1114
 - creating empty cart objects, XML 679
 - creating objects from text strings, JSP 1116
 - creating objects from text strings, XML 682
 - custom field order item parameters 1295
 - deleting items, JSP 1115
 - deleting items, XML 679
 - deleting, JSP 1133
 - deleting, XML 695
 - emptying, JSP 1113
 - emptying, XML 677
 - finding abandoned carts, JSP 1135
 - finding abandoned carts, XML 697
 - general order item parameters 1292
 - listing item discount totals, JSP 1119
 - listing item discount totals, XML 682
 - merging, JSP 1129
 - merging, XML 691
 - optional parameters 1283
 - order item delivery parameters 1293
 - order item errors, JSP 1119
 - order item errors, XML 683
 - order item parameters 1277, 1289, 1296
 - other returned parameters 1287
 - overview, JSP 1110
 - overview, XML 674
 - parameters returned by committing carts 1287
 - product order item parameters 1292
 - required parameters 1279
 - retrieving all item parameters, JSP 1121
 - retrieving all item parameters, XML 684
 - retrieving all items in carts, JSP 1122
 - retrieving all items in carts, XML 685
 - retrieving and naming carts, JSP 1136
 - retrieving and naming carts, XML 698
 - retrieving carts from cart sets, JSP 1134
 - retrieving carts from cart sets, XML 695
 - retrieving discounts, JSP 1117
 - retrieving discounts, XML 680
 - retrieving final totals, JSP 1118
 - retrieving final totals, XML 681
 - retrieving generic parameters, JSP 1124
 - retrieving generic parameters, XML 687
 - retrieving identifiers for carts in cart sets, JSP 1133
 - retrieving identifiers for carts in cart sets, XML 694
 - retrieving item totals, JSP 1123
 - retrieving item totals, XML 686
 - retrieving legal parameter values, JSP 1123
 - retrieving legal parameter values, XML 686
 - retrieving legal values, JSP 1120
 - retrieving legal values, XML 683
 - retrieving lists of discounts, JSP 1116
 - retrieving lists of discounts, XML 682
 - retrieving parameter selections, JSP 1125
 - retrieving parameter selections, XML 688
 - retrieving preliminary totals, JSP 1126
 - retrieving preliminary totals, XML 688
 - retrieving shipping totals, JSP 1126
 - retrieving shipping totals, XML 689

- retrieving shopping carts with session identifiers (visitor IDs), JSP 1137
- retrieving shopping carts with session identifiers (visitor IDs), XML 698
- retrieving single item parameters, JSP 1120
- retrieving single item parameters, XML 684
- retrieving stores, JSP 1127
- retrieving stores, XML 689
- retrieving tax totals, JSP 1127
- retrieving tax totals, XML 690
- retrieving transaction identifiers, JSP 1128
- retrieving transaction identifiers, XML 690, 691
- sending shopping carts to Transact, JSP 1114
- sending shopping carts to Transact, XML 678
- setting item prices and quantities, JSP 1130
- setting item prices and quantities, XML 692
- setting order item parameters, JSP 1129
- setting order item parameters, XML 691
- setting single cart parameters, JSP 1130
- setting single cart parameters, XML 692
- setting specified cart parameters, JSP 1131
- setting specified cart parameters, XML 693
- subscription order item parameters 1294
- taxation order item parameters 1295
- updating shopping carts, JSP 1138
- updating shopping carts, XML 700
- writing cart objects into text strings, JSP 1131
- writing cart objects into text strings, XML 693
- site node, JSP 1085
- site node, XML 491
- site plan
 - placing pages in a hierarchy, JSP 1092
 - removing nodes from tree, JSP 1095
 - retrieving field data from node, JSP 1085
 - retrieving field data from node, XML 491
 - retrieving nodes, JSP 1089
 - retrieving nodes, XML 495
 - returning unplaced pages, JSP 1096
 - root node, JSP 1093
 - root node, XML 499
 - setting node parents, JSP 1094
- SitePlanTree table, JSP 1022
- SitePlanTree table, XML 373
- sites
 - adding to lists, XML 485
 - clearing lists, XML 485
 - deleting from lists, XML 485
 - gathering variables, XML 486
 - listing, XML 486
 - nodes, JSP 1036
 - nodes, XML 390
 - scattering variables, XML 488
 - setting list to "any" role, XML 488
 - verifying in a list, XML 487
 - verifying list descriptions, XML 487
- SOAP
 - creating an error message, XML 810, 1234
 - defining a SOAP header, XML 808, 811, 1232, 1235
 - indicating the message body, XML 809, 1233
 - processing element content, XML 812, 1236
- SQL
 - converting Java SQL dates, JSP 945
 - converting Java SQL dates, XML 285
 - executing inline statements, JSP 952
 - executing inline statements, XML 203
 - executing queries, XML 74
 - executing statements, XML 181
 - retrieving and executing queries, JSP 860
 - retrieving queries, XML 74
- store parameters 677, 1112
- streaming
 - flushing body content, JSP 857
 - flushing data streams, JSP 917
 - inserting page evaluations in to output streams, JSP 932
 - inserting page evaluations in to output streams, XML 221
 - streaming field values from objects, JSP 1007

- streaming field values from objects, XML 264
 - strings
 - See also substrings*
 - building URL strings, JSP 914
 - building URL strings, XML 193
 - converting Content Server objects, JSP 1188
 - converting Content Server objects, XML 748
 - converting doubles to currency strings, JSP 1155
 - converting doubles to currency strings, XML 716
 - converting lists to strings, JSP 1191
 - converting lists to strings, XML 751
 - converting monetary strings to decimals, JSP 1157
 - converting monetary strings to decimals, XML 717
 - converting searchstate objects, JSP 1186
 - converting searchstate objects, XML 746
 - converting strings for passage through URLs, JSP 1193
 - converting strings for passage through URLs, XML 753
 - converting strings to lowercase, XML 309
 - converting strings to uppercase, XML 310
 - converting to objects, JSP 1190
 - converting to objects, XML 750
 - creating csuser objects from text strings, JSP 1146
 - creating csuser objects from text strings, XML 707
 - creating shopping cart objects from text strings, JSP 1116
 - creating shopping cart objects from text strings, XML 682
 - ending with substrings, XML 178
 - formatting for CALLSQL JSP 934
 - formatting for CALLSQL, XML 198
 - generating MAC query strings, JSP 1140
 - generating MAC query strings, XML 702
 - generating MAC strings, JSP 1141
 - generating MAC strings, XML 703
 - generating unique strings within Java VMs, XML 369
 - initializing searches from strings, JSP 1186
 - initializing searches from strings, XML 746
 - retrieving indexes, XML 216
 - retrieving input for variables, JSP 1188
 - retrieving input for variables, XML 748
 - retrieving input string lengths using variables, JSP 1191
 - retrieving input string lengths using variables, XML 751
 - setting alias strings, JSP 1199
 - setting alias strings, XML 759
 - SQL where clause, JSP 953
 - SQL where clause, XML 204, 299
 - substituting, XML 303
 - substrings, XML 70
 - unpacking URL strings, JSP 1193
 - unpacking URL strings, XML 753
 - verifying items, XML 225
 - writing cart objects into text strings, JSP 1131
 - writing cart objects into text strings, XML 693
 - writing csuser objects to text strings, JSP 1153
 - writing csuser objects to text strings, XML 714
 - writing log files, JSP 936
 - writing log files, XML 234
 - writing strings to file, XML 350
 - writing strings to log files, JSP 936
 - writing strings to log files, XML 234
 - substrings
 - finding, XML 70
 - substituting, XML 303
 - variables, XML 305
- ## T
- tax 1286
 - text format
 - converting lists to strings, JSP 1191
 - converting lists to strings, XML 751
 - retrieving input strings, JSP 1191
 - retrieving input strings, XML 751
 - time
 - naming current time, JSP 1014
 - naming current time, XML 309

- retrieving AM/PM values, XML 353
- retrieving elapsed time, JSP 1013
- retrieving elapsed time, XML 308
- retrieving value of hour, XML 359
- retrieving value of minute, XML 361
- timestamps
 - retrieving most recent visitor time stamps, XML 773
 - retrieving visitor time stamps, JSP 1214
 - retrieving visitor time stamps, XML 773
- TimeZoneIDs 1333
- Transact 1277
 - changing payment account data, JSP 1150
 - changing payment account data, XML 711
 - checking shopping carts 1285
 - checking shopping carts against Transact, JSP 1112
 - checking shopping carts against Transact, XML 676
 - clearing visitor's commerce identifiers, JSP 1200
 - clearing visitor's commerce identifiers, XML 759
 - committing shopping carts 1287
 - creating cookies for Transact recognition, JSP 1139
 - creating cookies for Transact recognition, XML 701
 - deleting payment accounts, JSP 1145
 - deleting payment accounts, XML 706
 - Message Authentication Code (MAC), JSP 1139
 - Message Authentication Code (MAC), XML 701
 - modifying buyer data, JSP 1149
 - modifying buyer data, XML 710
 - Order Entry API (OEAPI) 1277, 1289
 - registered buyers, JSP 1143
 - registered buyers, XML 704
 - registering new buyers, JSP 1151
 - registering new buyers, XML 712
 - retrieving Message Authentication Codes (MAC), JSP 1141
 - retrieving Message Authentication Codes (MAC), XML 703
 - retrieving Transact transaction identifiers from shopping carts, JSP 1128
 - retrieving Transact transaction identifiers from shopping carts, XML 690
 - retrieving visitor commerce identifiers, JSP 1198
 - retrieving visitor commerce identifiers, XML 757
 - sending shopping carts to Transact, JSP 1114
 - sending shopping carts to Transact, XML 678
 - setting single buyer parameters, JSP 1152
 - setting single buyer parameters, XML 713
 - setting specified buyer parameters, JSP 1152
 - setting specified buyer parameters, XML 713
 - setting visitor's commerce identifiers, JSP 1200
 - setting visitor's commerce identifiers, XML 759
- transact.ini 677, 1112
- trees
 - copying nodes, JSP 962
 - copying nodes, XML 319
 - creating tree tables, JSP 963
 - creating tree tables, XML 320
 - deleting multiple nodes, JSP 965
 - deleting multiple nodes, XML 322
 - deleting nodes, JSP 964
 - deleting nodes, XML 321
 - getting node parent records, JSP 971
 - getting node parent records, XML 330
 - listing ancestor nodes, JSP 974
 - listing ancestor nodes, XML 335
 - listing nodes, JSP 967
 - listing nodes, XML 325
 - listing, JSP 972
 - listing, XML 332
 - moving nodes, JSP 973
 - moving nodes, XML 334
 - retrieving child nodes, JSP 968
 - retrieving child nodes, XML 326
 - retrieving nodes from tables, JSP 970
 - retrieving nodes from tables, XML 329
 - verifying node paths, JSP 978
 - verifying node paths, XML 340

U**URLs**

- building URL strings, JSP 914
- building URL strings, XML 193
- converting strings for passage through URLs, JSP 1193
- converting strings for passage through URLs, XML 753
- converting strings, JSP 1193
- converting strings, XML 753
- creating BlobServer URLs, JSP 1070
- creating BlobServer URLs, XML 463
- dynamic used on a static site, JSP 1070
- dynamic used on a static site, XML 464, 466
- encoding arguments in URLs, JSP 1075
- encoding arguments in URLs, XML 469
- encoding characters, XML 459

users

- adding to lists, XML 507
- adding, JSP 869, 1015
- adding, XML 87, 344
- authenticating, JSP 1143
- authenticating, XML 704
- authorizing, JSP 886
- authorizing, XML 112
- changing user passwords, JSP 1016
- changing user passwords, XML 345
- clearing lists, XML 507
- clearing roles, XML 418
- deleting from lists, XML 507
- deleting payment accounts, JSP 1145
- deleting payment accounts, XML 706
- deleting, JSP 877, 1017
- deleting, XML 100, 346
- gathering variables, XML 508
- listing user IDs, XML 508
- logging in as a particular user, JSP 1020
- logging in as a particular user, XML 349
- logging in users, JSP 1017
- logging in users, XML 346
- logging out users, JSP 1018
- logging out users, XML 347
- logging out, JSP 887
- logging out, XML 113
- modifying user accounts, JSP 1019
- modifying user accounts, XML 348
- retrieving assigning user IDs, XML 536

- retrieving Commerce Connector user information, JSP 1143
- retrieving Commerce Connector user information, XML 704
- retrieving email addresses, XML 418
- retrieving generic parameter specifications, JSP 1148
- retrieving generic parameter specifications, XML 709
- retrieving names, XML 420
- retrieving parameter selections for csuser objects, JSP 1148
- retrieving parameter selections for csuser objects, XML 709
- retrieving roles, XML 420
- retrieving sites, XML 421
- retrieving user IDs, XML 419
- retrieving users for specified assignments, XML 535
- scattering variables, XML 508
- setting email addresses, XML 421
- setting roles, XML 423
- setting usernames, XML 422
- verifying ACLs, XML 343

V

- value-added tax (VAT) 1304, 1305

values

- calculating, JSP 1108
- calculating, XML 673
- counting attribute values, JSP 1100
- counting attribute values, XML 664
- declaring values for parent tags, JSP 858
- declaring values for parent tags, XML 69, 191
- scoping, JSP 958
- scoping, XML 71, 78, 145, 179, 312

variables

- deleting session variables, JSP 940
- deleting session variables, XML 274
- deleting, JSP 941
- deleting, XML 275
- displaying values, XML 149
- rendermode, JSP 1062
- rendermode, XML 455
- resolving multiple, JSP 941
- resolving multiple, XML 202
- retrieving input strings, JSP 1188
- retrieving input strings, XML 748

- retrieving session variables, JSP 919
 - retrieving variable values, JSP 920
 - retrieving variable values, XML 196
 - returning input strings, JSP 1191
 - returning input strings, XML 751
 - returning variables to a stacked state, JSP 940
 - saving to a stack, JSP 939
 - setting session variables, JSP 950
 - setting session variables, XML 296
 - setting, JSP 951
 - setting, XML 297
 - substituting values, XML 277, 278
 - substrings, XML 305
 - verifying session variables, XML 228
 - verifying, XML 229
 - Verity Search Engine
 - defining index fields, JSP 915
 - visitor data
 - clearing commerce identifiers, JSP 1200
 - clearing commerce identifiers, XML 759
 - clearing visitor access identifiers, JSP 1198
 - clearing visitor access identifiers, XML 758
 - context, JSP 1204
 - flushing visitor data, JSP 1196
 - flushing visitor data, XML 756
 - retrieving access identifiers, JSP 1197
 - retrieving access identifiers, XML 756
 - retrieving commerce identifiers, JSP 1198
 - retrieving commerce identifiers, XML 757
 - retrieving scalar attributes, XML 775
 - retrieving visitor aliases, JSP 1197
 - retrieving visitor aliases, XML 757
 - saving history types, JSP 1217
 - saving history types, XML 776
 - saving objects as visitor attributes, JSP 1217
 - saving objects as visitor attributes, XML 776
 - saving visitor attributes, JSP 1218
 - saving visitor attributes, XML 777
 - setting alias strings, JSP 1199
 - setting alias strings, XML 759
 - setting visitor access identifiers, JSP 1198
 - setting visitor access identifiers, XML 758
 - setting visitor's commerce identifiers, JSP 1200
 - setting visitor's commerce identifiers, XML 759
 - slotting numbers into calculated intervals, XML 786
 - visitor data manager, JSP 1195, 1212
 - visitor data manager, XML 755, 771
 - visitors
 - context, XML 763
 - creating lists of promotions for eligible, XML 766
 - creating new visitor IDs, JSP 1137
 - creating new visitor IDs, XML 699
 - creating promotions, JSP 1205
 - creating promotions, XML 763
 - discarding visitor IDs, JSP 1137
 - discarding visitor IDs, XML 699
 - listing visitor segments, JSP 1205
 - listing visitor segments, XML 764, 768
 - retrieving history, JSP 1213
 - retrieving history, XML 772
 - retrieving most recent visitor time stamps, XML 773
 - retrieving scalar visitor attributes, JSP 1216
 - retrieving scalar visitor attributes, XML 775
 - retrieving timestamps, JSP 1214
 - retrieving timestamps, XML 773
 - retrieving visitor timestamp history, JSP 1214
 - summing visitor history, JSP 1215
 - summing visitor history, XML 774
- ## W
- workflow
 - abstaining from voting, XML 548
 - adding assets to a group, XML 549
 - adding assets to groups, XML 550
 - cancelling assignments, XML 553
 - cancelling workflow assignments, XML 551
 - creating assignments, XML 555
 - creating conditions, XML 555
 - creating deadlock actions, XML 556

- creating delegation actions, XML 556
- creating functions, XML 557
- creating group actions, XML 557
- creating groups, XML 557
- creating processes, XML 558
- creating states, XML 558
- creating step actions, XML 559
- deleting assignment actions, XML 560
- deleting conditions, XML 560
- deleting deadlock actions, XML 560
- deleting delegate actions, XML 561
- deleting functions, XML 561
- deleting group actions, XML 561
- deleting groups, XML 561
- deleting processes, XML 562
- deleting states, XML 562
- deleting step actions, XML 562
- executing timed actions, XML 563
- listing candidate users, XML 563, 570
- listing group members, XML 578
- listing legal steps, XML 579
- listing workflow group participants, XML 603
- loading assignment actions, XML 566, 567
- loading assignment histories, XML 581
- loading assignments, XML 566
- loading conditions, XML 567, 571
- loading deadlock actions, XML 567, 572
- loading delegatee candidates, XML 573
- loading delegation actions, XML 568, 572, 573
- loading functions, XML 564, 576, 577
- loading group actions, XML 568, 577
- loading groups, XML 568, 578, 579
- loading participants, XML 581, 582
- loading processes, XML 564, 580, 583
- loading states, XML 565, 584, 585
- loading step actions, XML 569, 585
- loading steps, XML 586
- removing assets from groups 588
- removing objects, XML 553
- removing timed actions from queue, XML 563
- retrieving asset IDs, XML 532
- retrieving asset types, XML 532
- retrieving assigning user's ID, XML 536
- retrieving assignment dates, XML 534
- retrieving assignment due dates, XML 537
- retrieving assignment status, XML 545
- retrieving comments for assignments, XML 537
- retrieving descriptions of workflow groups, XML 601
- retrieving disposition comments, XML 539
- retrieving disposition dates, XML 539
- retrieving dispositions of assignments, XML 538
- retrieving edited ACLs for workflow groups, XML 601
- retrieving group IDs for assignments, XML 542
- retrieving group IDs, XML 579
- retrieving grouped comments, XML 541
- retrieving grouped due dates, XML 541
- retrieving object names for display in the user interface, XML 522
- retrieving objects of assignments, XML 534
- retrieving process IDs, XML 582
- retrieving queued workflow steps, XML 544
- retrieving reason asset cannot move to next state, XML 569
- retrieving state IDs, XML 582
- retrieving users for specified assignments, XML 535
- retrieving workflow group names, XML 604
- retrieving workflow object names, XML 522
- retrieving workflow process IDs, XML 605
- retrieving workflow states, XML 544
- retrieving workflow step IDs, XML 543
- saving assignment actions, XML 589
- saving conditions, XML 589
- saving deadlock actions, XML 590
- saving delegation actions, XML 590
- saving functions, XML 590
- saving group actions, XML 591
- saving groups, XML 591
- saving processes, XML 592
- saving states, XML 592
- saving step actions, XML 592
- setting deadlines, XML 589

- setting delegation permissions, XML 559
- setting participants, XML 591
- setting user's permissions, XML 554
- starting, XML 593
- verifying assignments, XML 587
- verifying deadlock actions, XML 588
- verifying deletion permissions, XML 551
- verifying editing permissions, XML 552
- verifying function legality, XML 586
- verifying process permissions, XML 552
- workflow actions
 - adding to lists, XML 529
 - gathering fields, XML 524
 - gathering variables, XML 529
 - loading from lists, XML 530
 - removing from lists, XML 529, 530
 - retrieving arguments, XML 525
 - retrieving descriptions, XML 525
 - retrieving elementnames, XML 525
 - retrieving names, XML 526
 - retrieving object IDs, XML 526
 - scattering variables, XML 526, 531
 - setting arguments, XML 527
 - setting descriptions, XML 527
 - setting elementnames, XML 528
 - setting names, XML 528
- workflow assignments
 - retrieving disposition user IDs, XML 540
 - retrieving IDs, XML 543
 - scattering variables, XML 546
- workflow function
 - setting names, XML 598
- workflow privileges
 - adding privileges, XML 618
 - clearing privileges, XML 618
 - gathering variables, XML 613, 619
 - listing privileges, XML 619
 - removing privileges from lists, XML 620
 - retrieving function IDs, XML 614
 - retrieving roles, XML 614
 - retrieving state IDs, XML 615
 - scattering variables, XML 615, 620
 - setting function IDs, XML 616
 - setting permissions, XML 616
 - setting roles, XML 616
 - setting state IDs, XML 617
 - verifying permissions, XML 614
- workflow process
 - adding steps, XML 622
 - deleting steps, XML 623
 - gathering values, XML 624
 - retrieving administrator's roles, XML 625
 - retrieving authorized roles, XML 630
 - retrieving delegation actions, XML 625
 - retrieving function privileges, XML 626
 - retrieving process descriptions, XML 627
 - retrieving process IDs, XML 626
 - retrieving process names, XML 628
 - retrieving startstep IDs, XML 626, 627
 - retrieving steps, XML 630, 631
 - retrieving valid asset types, XML 632
 - retrieving valid roles, XML 629
 - retrieving valid sites, XML 629
 - scattering variables, XML 632
 - setting administrator's roles, XML 633
 - setting asset types, XML 637
 - setting delegation actions, XML 634
 - setting descriptions, XML 635
 - setting function privileges, XML 634
 - setting names, XML 635
 - setting roles, XML 636
 - setting sites, XML 636
 - setting steps, XML 637
- workflow state
 - gathering variables, XML 639
 - retrieving actions, XML 640
 - retrieving descriptions, XML 641
 - retrieving estimated time, XML 640
 - retrieving names, XML 641
 - retrieving state IDs, XML 640
 - scattering variables, XML 641
 - setting actions, XML 642
 - setting descriptions, XML 643
 - setting estimated time, XML 642
 - setting names, XML 643
- workflow step
 - gathering variables, XML 644
 - listing authorized roles, XML 645
 - listing conditions, XML 645

- listing deadlock actions, XML 646
- listing step actions, XML 645
- retrieving end states, XML 647
- retrieving group lock flags, XML 646
- retrieving IDs, XML 646
- retrieving names, XML 647
- retrieving notify roles, XML 647
- retrieving override flags, XML 648
- retrieving start states, XML 648
- retrieving types, XML 649
- scattering variables, XML 649
- setting authorization roles, XML 650
- setting conditions, XML 651
- setting deadlock actions, XML 651
- setting end states, XML 651
- setting group lock flags, XML 652
- setting names, XML 652
- setting notify roles, XML 652
- setting override flags, XML 653
- setting start states, XML 653
- setting step actions, XML 650
- setting types, XML 653

X

- XML parser
 - invoking with XML tags 801
- XML processing
 - tag mapping, XML 351
- XML tags 49
 - XML parsing 801
 - XML transformation 801
- XSLT processor
 - invoking with XML tags 801