Content Server Enterprise Edition

Version: 5.5

COM Interfaces Reference

Document Revision Date: Oct. 29, 2003

COM Interfaces Reference
Document Revision Date: Oct. 29, 2003
Product Version: 5.5

FatWire Technical Support
Web: http://www.fatwire.com/Support

FatWire Headquarters
FatWire, inc.
330 Old Country Road
Suite 207
Mineola, NY 11501
Web: www.fatwire.com

Table of

Contents

Chapter 1

Programming Overview

This chapter describes how the COM **(**Component Object Model) interface to Content Server Enterprise Edition enables you to use CSEE with Microsoft ASP (active server pages).

What Is the COM Interface?

Microsoft COM is a framework for developing and supporting component objects. COM, which is a popular programming environment, is useful for communication with Content Server because it enables data exchange between distributed objects.

The COM Interface is the FatWire™ specification for using COM to access a subset of standard Content Server functions from any COM client, including applications written as Active Server Pages. As an API (application program interface), it comprises collections of objects and methods that enable you to extract asset information from the Content Server database.

Accessible Information

Any COM client that follows the COM Interface specification can access the following information from the Content Server database:

- Site map of a CSEE site.
- Blob data, such as a PDF file.
- List of all the valid asset types and asset subtypes at a CSEE site.
- List of assets that match specified search criteria.
- Metadata associated with a particular asset.

ASP, COM, and Content Server

The COM Interface for Content Server is designed to enable application integration via Microsoft Active Server Pages (ASP). When using Content Server as the native delivery engine is not desired, ASP and the VBScript server-side scripting language enable you to create limited dynamic web applications.

An ASP page is an HTML page that contains server-side scripts that are processed by the web server before being sent to the user's browser. You can combine ASP with COM objects and Hypertext Markup Language (HTML) to create dynamic web pages. For example, you can use the COM Interface to retrieve a page or specified page assets from a remote Content Server site and write an ASP program to assemble them.

With server-side ASP, the web server assembles pages for delivery, and Content Server manages the content repository. ASP dynamically renders the page on the web server based on information COM objects get from the content repository.

The roundtrip process works as follows:

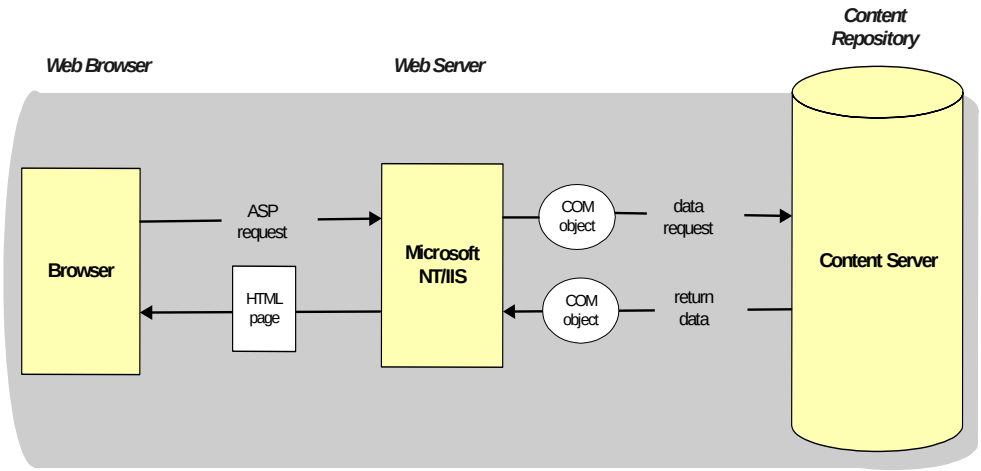

Figure 1: ASP Request and Data Flow

1. User clicks on link that requests an ASP page.
2. Web server evaluates the specified ASP page from top to bottom, runs the specified server-side scripts, and calls the appropriate COM object.
3. COM object calls CS-Direct seed classes.
4. Seeds invoke the specified Content Server action.
5. Content Server returns requested data (name/value pairs) to the appropriate COM object.
6. Web server dynamically assembles requested page in HTML format, including return data.
7. Web server returns assembled HTML page to the requesting browser.

Installing the COM Interface Client

Your COM Interface shipment includes an executable file that contains the COM Interface client software. The client must be installed on the machine that runs the IIS web server, on which you intend to develop your ASP scripts. Content Server can be installed on any machine.

On the server side, COM seed classes are installed during Content Server installation. If you have installed Content Server version 4.0.3 or higher, no further installation or configuration for Content Server is required.

To install the COM Interface client, follow these steps:

1. From Windows Explorer, navigate to the `CSCom.exe` file on the installation media or in your download directory.
2. Double-click the `CSCom.exe` file to start the standard InstallShield installation procedure.
3. Respond to the installation prompts, as follows:
   a. Review and accept license requirements.
   b. Specify the directory where the client will run.
   c. Install the software.

Coding Your Program

The COM Interface is a collection of interrelated methods and objects, logically grouped together according to their function. Methods return either instances of objects or Content Server data. Most objects are either inputs to other methods, outputs from methods, or both outputs of methods and inputs to methods. Objects are either created using supplied COM Interface methods or directly instantiated in your ASP code.

To develop a COM Interface program using VBScript in an Active Server Page, follow these general steps:

1. Create an HTML page with title, header, and so on.
2. Instantiate the CS object. The CS object contains methods that manage user sessions and allow instances of other required objects. Because this object contains additional methods that are inherited by the rest of the COM Interface methods, it must be instantiated first in all client programs.

   For example:

   ```
   Set CS = CreateObject (“CSDirect.CS”)
   ```
3. Pass user credentials and log in to Content Server.
4. Select the methods you need to use for your program and instantiate them.
5. Develop your application logic in VBScript, and call whichever COM Interface methods are required for your application.

Error Handling

Most methods throw a `COleDispatchException` on error. The exception object contains text that describes the reason for the failure.

Tasks and Methods

The following table lists common operations you can perform with the COM Interface and the associated methods that you use to accomplish them.

Table 1: Tasks and Required Methods

Task	Required Methods

Return property names and values of fields.	`GetAssetManager`
	`Load`
	`GetTextProperties`
	`GetPropertyNames`
	`GetProperty`

Return upload data from the `urlcolumn` field.	`GetAssetManager`
	`Load`
	`GetBinaryProperties`
	`GetProperty`
	`Save`

Return a blob image.	`GetIPSManager`
	`GetBlob`
	`Save`

Return the site parent of a given page.	`GetAssetManager`
	`GetSiteParent`
	`GetTextProperties`
	`GetPropertyNames`
	`GetProperty`

Return a list of assets for a given page.	`GetAssetManager`
	`FieldItem object`
	`ObjectCollection object`
	`AddObject`
	`List`
	`GetProperty`

Return a mungoblob (for example, a large binary object like a PDF file).	`GetIPSManager`
	`GetMungoBlob`
	`Save`

Return children for an asset.	`GetAssetManager` `GetChildren`

Task	Required Methods

Return site plan children for a particular node ID.	`GetSitePlanManager`
	`GetChildren`
	`GetProperty`
	`GetProperties`

Return a list of assets for a specified range-constraint searchstate.	`GetAssetSearchManager`
	`SearchState object`
	`ConstraintRange object`
	`SetLowerConstraint`
	`SetUpperConstraint`
	`AddAndConstraint`
	`AssetSortItem object`
	`ObjectCollection object`
	`AddObject`
	`GetAssetList`

Return multiple values of attributes based on a standard constraint.	`GetAssetSearchManager`
	`SearchState object`
	`ConstraintStandard object`
	`AddAndConstraint`
	`AssetSortItem object`
	`AddObject`
	`GetMultipleValues`
	`GetPropertyNames`
	`GetProperty`

Return number of assets given match items.	`GetAssetSearchManager`
	`AssetMatchItem object`
	`ObjectCollection object`
	`AssetSortItem object`
	`AddObject`
	`GetAssetCount`

Return attribute values based on a rich text constraint.	`GetAssetSearchManager`
	`SearchState object`
	`ConstraintRichText object`
	`AddAndConstraint`
	`GetAttributeValues`
	`GetProperty`

Task	Required Methods
Return attribute values given a range constraint.	GetAssetSearchManager ConstraintNested object SearchState object ConstraintStandard ConstraintRange AddAndConstraint AssetSortItem object ObjectCollection object AddObject GetAssetCount GetMultipleValues
Return multiple values given AndOrConstraint with Standard and Like Constraints.	GetAssetSearchManager SearchState object ConstraintStandard object AddOrConstraint ConstraintLike AssetSortItem object ObjectCollection object GetMultipleValues GetPropertyNames

Example Programs

This section describes two complete programs that illustrate how to instantiate required objects, log in users, and apply several COM Interface methods that return information from the Content Server database.

Note

In addition to the programs provided here, the individual description for each COM Interface method includes a code snippet that demonstrates use of that method in context.

Displaying Asset Data

The following program loads and displays the `name`, `description`, and `urlbody` fields and a blob image associated with an article:

```
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft FrontPage 5.0">
<META HTTP-EQUIV="Content-Type" content="text/html; charset=iso-
8859-1">
<TITLE>asp test</TITLE>
</HEAD>
```

```
<BODY>

<!-- Insert HTML here -->
<!-- #include file="./connectionparams.inc" -->
<%
dim CS
dim AM
dim List
dim TextProperties
dim propnames
dim Asset
dim BinaryProperties
set CS = CreateObject ("CSDirect.CS")
if CS.Login (strUsername, strPassword, strURL) then

   set AssetManager = CS.GetAssetManager()

   set Asset = AssetManager.Load("Article", "984156689074", "",
"", "", False)

   set TextProperties = Asset.GetTextProperties()

   set propnames = TextProperties.GetPropertyNames()

%>
<TABLE BORDER=3 CELLPADDING=2 cellspacing="4">
<TR>
   <TH>text property name</TH>
</TR>

<%
   for each propname in propnames
%>
<TR>
<TD><% = propname %>  </TD>
</TR>

<%
   next
%>
</table>

name property value is: <%=TextProperties.GetProperty("name")%>
<BR><BR>

<%
   set BinaryProperties = Asset.GetBinaryProperties()
   set propnames = BinaryProperties.GetPropertyNames()
%>
<TABLE BORDER=3 CELLPADDING=2 cellspacing="4">
<TR>
   <TH>binary property name</TH>
```

```
</TR>

<%   for each propname in propnames
%>
<TR>
<TD><% = propname %>  </TD>
</TR>

<%
   next
%>

<%
   set text = BinaryProperties.GetProperty("urlbody")
   text.Save("c:/temp/t1.txt")
%>

<%
   set IPSManager = CS.GetIPSManager()
   set image = IPSManager.GetBlob("ImageFile", "urlpicture", "id",
"985668092853", "image/jpeg")
   image.Save("c:/temp/g1.jpg")
%>

<TABLE BORDER=3 CELLPADDING=2 cellspacing="4">
<TR>
   <TH>name</TH><TH>description</TH><TH>Body</TH><TH>Image</TH>
</TR>
<TR>
<TD><% = TextProperties.GetProperty("name") %>  </TD>
<TD><% = TextProperties.GetProperty("description") %>  </TD>
<TD><A HREF=c:/temp/t1.txt>BodyText</A>  </TD>
<TD><IMG SRC=c:/temp/g1.jpg></IMG>  </TD>
</TR>


<%
else
   response.write ("login failed")
end if

%>

</BODY>
</HTML>
```

Returning an Asset List with a SearchState

The following program includes methods that specify conditions for a SearchState, which applies to the requested return data:

```
<HTML>
<HEAD>
```

```
<META NAME="GENERATOR" Content="Microsoft FrontPage 5.0">
<META HTTP-EQUIV="Content-Type" content="text/html; charset=iso-
8859-1">
<TITLE>asp test</TITLE>
</HEAD>
<BODY>

<!-- Insert HTML here -->
<!-- #include file="./connectionparams.inc" -->
<%
dim CS
dim SearchState
dim SearchManager
dim StandardConstraint
dim SortItem1
dim SortItem2
dim SortItems
dim Props
dim PropSets
dim propnames
dim propname
dim propsetcollection
dim pset

set CS = CreateObject ("CSDirect.CS")
if CS.Login (strUsername, strPassword, strURL) then

   set SearchManager = CS.GetAssetSearchManager()

   set SearchState = CreateObject("CSDirect.SearchState")
   set RangeConstraint = CreateObject("CSDirect.ConstraintRange")

   RangeConstraint.TypeName = "PAttributes"
   RangeConstraint.Attribute = "CommissionType"
   RangeConstraint.SetLowerConstraint "2.0", false
   RangeConstraint.SetUpperConstraint "4.0", false

   SearchState.AddAndConstraint(RangeConstraint)

   set SortItem1 = CreateObject("CSDirect.AssetSortItem")

   SortItem1.attributename = "CommissionType"
   SortItem1.attributetypename = "PAttributes"

   set SortItems = CreateObject("CSDirect.ObjectCollection")

   SortItems.AddObject(SortItem1)

   set propsetcollection = SearchManager.GetAssetList(SortItems,
"", SearchState, "Products", "", 10, "fr_FR")
%>
   <TABLE BORDER=3 CELLPADDING=2 cellspacing="4">
```

```
   <TR>
      <TH>assettype </TH>
      <TH>assetid </TH>
   </TR>

<%
   for each pset in propsetcollection
%>

   <TR>
   <TD><% = pset.GetProperty("assettype") %>  </TD>
   <TD><% = pset.GetProperty("assetid") %>  </TD>
   </TR>
<%   next
%>

</TABLE> <br><br>

<%
else
   response.write ("login failed")
end if
%>

</BODY>
</HTML>
```

Installing Sample ASP Files

This section contains instructions for installing and configuring sample ASP programs that make COM calls against the sample sites included with Content Server 5.0. The supplied ASP programs complement the HelloAssetWorld and Burlington Financial sample sites.

About the HelloAssetWorld Sample Site

HelloAssetWorld is a sample Content Server site that demonstrates basic Content Server services, including access to them via ASP and a subset of the supplied COM Interface methods. Although the sample site is not required to use the COM Interface methods, FatWire recommends that you install it to get a working model on which you can base your development work. HelloAssetWorld requires server-side (Content Server) and client-side (web server) configuration.

To install HelloAssetWorld on the Content Server side, select the sample site option when you install the CS-Direct application. The sample site for Content Server includes the following components:

- HelloAssetWorld assets
- HelloAssetWorld sample elements
- HelloAssetWorld images

Your COM Interfaces shipment includes sample ASP files for HelloAssetWorld that must be installed on the COM client computer.

About the Burlington Financial Sample Site

Your COM Interfaces client software includes test programs that exercise each COM Interface method. The test programs use the database for the Burlington Financial sample site, which is another sample site installed as part of Content Server 5.0. Although the test programs are not required to use HelloAssetWorld, you can install them on your IIS/COM machine and run them against data used for the Burlington Financial site. Unlike HelloAssetWorld, however, the Burlington Financial site is not constructed from ASP.

Testing HelloAssetWorld

HelloAssetWorld requires Content Server 5.0—ensure that you are running Content Server Enterprise Edition version 5.0 and that you have installed the HelloAssetWorld sample files during CS-Direct installation. Before you begin, you must also install the COM client software on the machine on which you intend to run the sample ASP programs. There is no server-side installation required for COM. For client installation instructions, see “Installing the COM Interface Client” on page 9.

To verify that the HelloAssetWorld site for Content Server is working:

1. Log in to the CS-Direct application as user `admin` with password `xceladmin`.
2. Select the **Admin** tab.
3. From the tree view, select **Sites**, and verify that **HelloAssetWorld** is displayed.
4. Log out of the site so that you are no longer the `admin` user.
5. Log in to the HelloAssetWorld site as user `Coco` with password `hello`. As user Coco, you can explore the site configuration as you please.

Installing HelloAssetWorld ASP Files

The HelloAssetWorld ASP files make COM function calls against the HelloAssetWorld sample site installed with Content Server. To install the HelloAssetWorld ASP files on the IIS web server:

1. Create a virtual directory to a local folder.

   For example: `c:\Inetpub\wwwroot\ASPHelloWorld` on IIS.

   Ensure that IIS directory browsing is enabled.

2. Locate the HelloAssetWorld ASP files. The sample ASP files are in the same directory where you installed the COM client:

   `com_client_dir``\ASPsamples\HelloAssetWorld`

3. Copy the `ASPHelloWorld` ASP files to the local folder that you just created.

4. Create a virtual directory called `textfiles` that points to a temporary directory (for example, `c:\temp`). This directory, which is referenced in the HelloAssetWorld ASP files that use the `Save` method, stores binary files from the save operation.

5. Edit the `connectionparams.inc` file, which is located in the directory where you copied the HelloAssetWorld sample files. The `connectionparams.inc` file is an include file that supplies the user name, password, and url of the web-server computer to each HelloAssetWorld ASP program. Change the default entries in the file to correspond to the actual user name, password, and URL for your system.

6. Run the `HelloWorldPage.asp` script. For example:

   `http://`*`hostname`*`/ASPHelloWorld/HelloWorldPage.asp`

Installing Burlington Financial ASP Files

The Burlington Financial ASP files make COM function calls against the Burlington Financial sample site data installed with Content Server. The sample `.asp` test files are not required for the HelloAssetWorld sample site. Optionally, you can install the sample test files on the Microsoft IIS web server and run them against the Burlington Financial site. Burlington Financial is preinstalled and configured if you chose to install it during CS-Direct installation.

To install the ASP sample test files on the IIS web server:

1. Create a virtual directory called `ASPSamples` that points to a local folder (for example, `c:\Inetpub\wwwroot\ASPSamples`) on IIS. Ensure that IIS directory browsing is enabled.

2. Locate the Burlington Financial sample ASP files. The sample ASP files are in the same directory where you installed the COM client:

   `com_client_dir``\ASPsamples\BurlingtonFiancial`

3. Copy the sample ASP files for the Burlington Financial sample site from the `\BurlingtonFinancial` directory to the local folder that you just created.

4. Create a virtual directory called textfiles that points to a temporary directory (for example, `c:\temp`). This directory, which is referenced in the sample ASP files that use the `Save` method, stores binary files from the Save operation.

5. Edit the `connectionparams.inc` file, which is located in the directory where you copied the sample ASP files for the Burlington Finanacial site. The `connectionparams.inc` file is an include file that supplies the user name, password, and url of the computer that contains the Burlington Financial site to each test ASP file. Change the default entries in the file to correspond to the actual user name, password, and URL for your system.
6. Run the sample test files.

Chapter 2

Methods

This chapter describes the methods available to COM clients for accessing selected information from a Content Server delivery system.

Methods by Task

The following tables group related methods according to their function:

- Asset
- Asset Management
- Asset Search Management
- Blob
- IPS Management
- ObjectCollection
- Property Set
- SearchState
- Session Management
- SitePlan Management

Asset

Asset methods act on the output of the AssetManager `Load` and `GetSiteManager` operations. Each method corresponds to a particular type of load output, which can be a list of strings, list of binaries, or a list of lists.

GetBinaryProperties	GetListProperties	GetTextProperties

Asset Management

AssetManager methods retrieve main table fields, children, parents, and listing assets for assets and flex assets.

GetAssetManager	GetChildren	GetSiteNode	GetSiteParent
List	Load		

Asset Search Management

AssetSearchManager methods cover the functions required to retrieve attribute values for flex assets.

GetAssetCount	GetAssetSearchManager	GetAttributeValues	GetMultipleValues

Blob

The Blob method saves a blob object returned from an IPSManager blob operation to a file.

Save

IPS Management

IPSManager methods cover basic Content Server functions, such as returning a blob stored in a particular table for assets and flex assets, and searching indexes generated by a search engine.

GetBlob	GetIPSManager	GetMungoBlob	Save

ObjectCollection

The `AddObject` method adds an object to the `ObjectCollection` instance. ObjectCollection objects are passed as input to the `List` method and to the AssetSearchManager methods.

AddObject

Property Set

PropertySet methods return column names and values. Once you identify the property name, you can use it to return its value.

GetProperty	GetPropertyNames

SearchState

The SearchState methods are used to build a searchstate that in turn is used to generate a collection of asset properties. As inputs to the `SearchState` object, these methods are partial inputs to the AssetSearchManager methods.

AddAndConstraint	AddOrConstraint	SetLowerConstraint	SetUpperConstraint

Session Management

SessionManager methods log users in and out of sessions. An active session must be established to use the COM Interface methods.

Login	Logout

SitePlan Management

SitePlan methods enable you to examine the page hierarchy of the site. In conjunction with each other, they return the children and properties of any node in the hierarchy.

GetChildren	GetProperties	GetSitePlanManager

Asset Methods

Asset methods act on the output of the AssetManager `Load` and `GetSiteParent` operations. Each method corresponds to a particular type of load output, which can be a list of strings, list of binaries, or a list of lists.

The Asset group comprises the following methods:

- GetBinaryProperties
- GetListProperties
- GetTextProperties

GetBinaryProperties

Retrieves a `PropertySet` object that contains the binary properties for the asset.

Syntax

```
GetBinaryProperties()
```

Parameters

None.

Description

Retrieves a `PropertySet` object that contains the binary properties for the asset. Each property in the PropertySet will return a blob object.

Returns

`PropertySet` object containing blob data.

Exceptions

Possible error values include:

Error Text
Missing required table in response: tablename

Example

The following code instantiates the `CS` object, logs the user in, instantiates the `AssetManager` object, and loads an article asset named `Nigeria-A73-2001Mar9`. The `GetBinaryProperties` method returns binary properties for the loaded asset and outputs the required `PropertySet` object. The remaining code displays the names of the properties from the `PropertySet` object in an HTML table.

```
<%
set CS = CreateObject ("CSDirect.CS")
if CS.Login (strUsername, strPassword, strURL) then

   set AssetManager = CS.GetAssetManager()
```

```
   set Asset = AssetManager.Load("Article", "", "name", "Nigeria-
A73-2001Mar9", "", True)

   set BinaryProperties = Asset.GetBinaryProperties()
   set propnames = BinaryProperties.GetPropertyNames()
%>
<TABLE BORDER=3 CELLPADDING=2 cellspacing="4">
<TR>
   <TH>binary property name</TH>
</TR>

<%   for each propname in propnames
%>
<TR>
<TD><% = propname %>  </TD>
</TR>

<%
next
%>
```

GetListProperties

Retrieve a PropertySet that contains the list properties for the asset. Each property in the PropertySet returns a PropertySet collection, which represents a list.

Syntax

`GetListProperties()`

Parameters

None.

Returns

`PropertySet` object.

Exceptions

Possible error values include:

Error Text
Missing required table in response: tablename

Example

The following code instantiates the `CS` object, logs the user in, instantiates the `AssetManager` object, and loads an `AdvCols` asset named `General Funds`. The `GetListProperties` method returns list properties for the loaded asset and outputs the required `PropertySet` object. The remaining code displays the names of the properties from the `PropertySet` object in an HTML table.

```
<%set CS = CreateObject ("CSDirect.CS")
if CS.Login (strUsername, strPassword, strURL) then

   set AssetManager = CS.GetAssetManager()

   set Asset = AssetManager.Load("AdvCols", "", "name", "General
Funds", "", False)
   set ListProperties = Asset.GetListProperties()
   set propnames = ListProperties.GetPropertyNames()
%>
<TABLE BORDER=3 CELLPADDING=2 cellspacing="4">
<TR>
   <TH>List property name</TH>
</TR>

<%
   for each propname in propnames
%>
<TR>
<TD><% = propname %>  </TD>
</TR>

<%
   next
%>

<%
else
   response.write ("login failed")
end if

%>
```

GetTextProperties

Retrieves a `PropertySet` object containing the text properties for the asset.

Syntax

```
GetTextProperties()
```

Parameters

None.

Description

Retrieves a `PropertySet` object containing the text properties for the asset. Each property in the PropertySet returns a string.

Returns

`PropertySet` object.

Exceptions

Possible error values include:

Error Text
Missing required table in response: tablename

Example

The following code instantiates the `CS` object, logs the user in, instantiates the `AssetManager` object, and loads an article asset named `Nigeria-A73-2001Mar9`. The `GetTextProperties` method returns text properties for the loaded asset and outputs the required `PropertySet` object.

```
<%
set CS = CreateObject ("CSDirect.CS")
if CS.Login (strUsername, strPassword, strURL) then

   set AssetManager = CS.GetAssetManager()

   set Asset = AssetManager.Load("Article", "", "name", "Nigeria-
A73-2001Mar9", "", True)

   set TextProperties = Asset.GetTextProperties()

   set propnames = TextProperties.GetPropertyNames()

%>
```

AssetManager Methods

AssetManager methods cover the functions required to retrieve attribute values for assets and flex assets. The AssetManager object must be instantiated before using any of the AssetManager methods.

The following method retrieves the AssetManager object:

- GetAssetManager

The remaining four asset manager methods are functions that list and load assets:

- GetChildren
- GetSiteNode
- GetSiteParent
- List
- Load

GetAssetManager

Retrieves an instance of the `AssetManager` object.

Syntax

```
GetAssetManager()
```

Parameters

None.

Description

`GetAssetManager` creates an instance of the AssetManager object, which is required to list and load assets. Because it contains methods that are inherited, you must instantiate this object before calling associated asset management functions.

Inputs

None.

Returns

Instance of the AssetManager object.

Exceptions

Possible error values include:

Error Text
Not logged in.

Example

The following code instantiates the `CS` object, logs in a user, and instantiates the `AssetManager` object.

```
<%
set CS = CreateObject ("CSDirect.CS")
if CS.Login (strUsername, strPassword, strURL) then

   set AssetManager = CS.GetAssetManager()
<!-- instantiate your COM Interface methods here --!>
else
   response.write ("logout failed")
end if
%>
```

See Also

Methods used in conjunction with the AssetManager object:

GetChildren

GetSiteNode

GetSiteParent

List

Load

GetChildren

Queries the AssetRelationTree table, builds a list of child assets for the specified parent asset, and loads asset data into the `PropertySet` COM object.

Syntax

```
GetChildren(Type, [ObjectID], [Field], [Value], [Code],
[ChildType], [ChildID], [OrderBy])
```

Parameters

`Type (required)`

(String) The asset type of the asset that you want to retrieve child properties. Standard asset types include `article`, `image`, `page`, `collection`, and `query`. The list of children is a join of the AssetRelationTree and the asset table for the type specified.

Typically, you provide `Type` and `ObjectID` to request a specific child asset. When you know that a specific field/value pair can uniquely identify the asset, you can provide `Type` and the field/value pair instead. Either the `ObjectID` or the `Field` and `Value` combination, but not both, are required to load an asset.

`ObjectID (optional)`

(String) The unique identifying number that references the Asset object. Not required if you use Type with the Field and Value paired parameters (defined below).

`Field (optional)`

(String) A field is any one of the column names for the asset. For example, standard fields for CS-Direct assets include name, template, status, description, `subtype`, `category`, `modified`, `headline`, `byline`, and `body`. Use the field name in conjunction with the `Value` parameter as the name portion of a name/value pair. The field name and its corresponding value uniquely identifies the asset to be loaded. Note that if the field/value pair that you supply identifies more than one asset, the `Load` method uses the first one that it finds.

Not required if you use `Type` with the `ObjectID` parameter.

`Value (optional)`

(String) Value that corresponds to the field specified by the `Field` parameter. Paired with the field name, this parameter uniquely identifies an asset by supplying the associated value.

Not required if you use `Type` with the `ObjectID` parameter.

`Code (optional)`

(String) Restricts the list of children to children whose pages are designated as either placed or unplaced relationships. Placed pages are those that have been set to a particular level in the tree hierarchy. Unplaced pages are those pages that have yet to be placed, or that are intentionally left unplaced.

Valid values are either `Placed` or `Unplaced`.

`ChildType (optional)`

(String) The asset type of child that you want to retrieve. The child asset type, which can be the same as the `Type` parameter, depends on the asset that you are passing. For example, common asset types include `article`, `image`, `page`, `collection`, and `query`. Other asset types include flex assets and CS-Engage assets. If no `ChildType` parameter is passed, the `GetChildren` method returns all children for the asset by default.

`ChildID (optional)`

(String) The object ID of the specific child node to return. If you supply a child ID, you must also supply a child type. If no child ID is specified, the `GetChildren` method returns all children for the asset.

`OrderBy (optional)`

(String) The fields to sort the list by, and whether the sort result on those fields is `ascending` or `descending`. For example, you can specify `ID`, `name`, `date`, `created by user`, and so on. 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 `nrank, ascending`, the list is sorted by rank starting at number *1*. If you want the list sorted by descending rank, specify `nrank, descending`.

Description

This method 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 with the `Load` method before you can invoke this method to query for and list its children. The list is a standard Content Server list.

Typical use for this method is 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 `Type` 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 that case, you do not need to use the `Load` method for a child asset of that type if that asset type stores all of its asset data in the primary asset table. Instead, you can get that information from the list.

Returns

Collection of `PropertySet` objects

Exceptions

Possible error values include:

Error Text
Need type parameter
Need either ID or FIELD, VALUE parameters
Both field and value parameters must be specified
Could not communicate with server
Invalid response from server

Example

This code retrieves a collection of articles, determines the members of the collection, and then displays the `nid` (node ID) field of each article:

```
<%
set CS = CreateObject ("CSDirect.CS")
if CS.Login (strUsername, strPassword, strURL) then

   set AssetManager = CS.GetAssetManager()

   set AssetChildrenList = AssetManager.GetChildren("Collection",
"", "name", "MarketsTop", "", "Article", "", "")
%>

There are <% =AssetChildrenList.Count %>  Asset children<p>
<TABLE BORDER=3 CELLPADDING=2 cellspacing="4">
<TR>
   <TH>nid</TH><TH>ncode</TH>
</TR>
<% for each asset in AssetChildrenList
%>
<TR>
<TD><% = asset.GetProperty("nid") %>  </TD>
<TD><% = asset.GetProperty("ncode") %>  </TD>
</TR>
```

```
<%
next
%>
</TABLE>
```

GetSiteNode

Queries the SitePlanTree table and returns the node ID of the specified page asset.

Syntax

```
GetSiteNode(Type, [ObjectID], [Field], [Value])
```

Parameters

`Type (required)`

(String) The asset type of the asset that you want to retrieve from the database. Standard asset types include `article`, `image`, `page`, `collection`, and `query`.

Typically, you provide `Type` and `ObjectID`. When you know that a specific field/value pair can uniquely identify the asset, however, you can provide `Type` and the field/value pair instead. Either the `ObjectID` or the `Field` and `Value` combination, but not both, are required to load an asset.

`ObjectID (optional)`

(String) The unique identifying number that references the Asset object. Not required if you use Type with the Field and Value paired parameters.

`Field (optional)`

(String) A field is any one of the column names for the asset. For example, standard fields for CS-Direct assets include name, template, status, description, `subtype`, `category`, `modified`, `headline`, `byline`, and `body`. Use the field name in conjunction with the `Value` parameter as the name portion of a name/value pair. The field name and its corresponding value uniquely identifies the asset to be loaded. Note that if the field/value pair that you supply identifies more than one asset, the `Load` method uses the first one that it finds.

Not required if you use `Type` with the `ObjectID` parameter.

`Value (optional)`

(String) Value that corresponds to the field specified by the `Field` parameter. Paired with the field name, this parameter uniquely identifies an asset by supplying the associated value.

Not required if you use `Type` with the `ObjectID` parameter.

Description

The relationships set up between page assets on the site tree in the CS-Direct main window are stored in the SitePlanTree table. The `GetSiteNode` method uses the object ID of a page asset to retrieve its node ID from that table.

With the node ID, you can acquire information about the site's hierarchy to use for display. For example, you can create a navigation bar with links to section pages or a link back to the parent page.

Returns

String containing the site node ID.

Exceptions

Possible error values include:

Error Text
Need type parameter
Need either ID or FIELD, VALUE parameters
Both field and value parameters must be specified
Could not communicate with server
Invalid response from server

Example

The following code loads a page asset and then determines the site node of that page:

```
<%
set CS = CreateObject ("CSDirect.CS")
if CS.Login (strUsername, strPassword, strURL) then

   set AssetManager = CS.GetAssetManager()

   AssetSiteNode = AssetManager.GetSiteNode("Page",
"968685129804", "", "")
%>

Site node for page with id 968685129804 is --- <% = AssetSiteNode
%>
```

GetSiteParent

Queries the SitePlanTree table and then loads the parent page of the specified page asset into memory as the `Asset` object.

Syntax

```
GetSiteParent(Type, [ObjectID], [Field], [Value], [FieldList],
[Exclude])
```

Parameters

`Type (required)`

(String) The asset type of the asset that you want to retrieve from the database. Standard asset types include `article`, `image`, `page`, `collection`, and `query`.

Typically, you provide `Type` and `ObjectID`. When you know that a specific field/value pair can uniquely identify the asset, you can provide `Type` and the field/value pair instead. Either the `ObjectID` or the `Field` and `Value` combination, but not both, are required to load an asset.

`ObjectID (optional)`

(String) The unique identifying number that references the `Asset` object. Not required if you use `Type` with the `Field` and `Value` paired parameters (defined below).

`Field (optional)`

(String) A field is any one of the column names for the asset. For example, standard fields for CS-Direct assets include name, template, status, description, `subtype`, `category`, `modified`, `headline`, `byline`, and `body`. Use the field name in conjunction with the `Value` parameter as the name portion of a name/value pair. The field name and its corresponding value uniquely identifies the asset to be loaded. Note that if the field/value pair that you supply identifies more than one asset, the `Load` method uses the first one that it finds.

Not required if you use `Type` with the `ObjectID` parameter.

`Value (optional)`

(String) Value that corresponds to the field specified by the `Field` parameter. Paired with the field name, uniquely identifies an asset by supplying its associated value.

Not required if you use `Type` with the `ObjectID` parameter.

`FieldList (optional)`

(String) Comma-separated list of fields that you want to include or exclude from the request for asset fields. The `Exclude` parameter, which operates on the field list, determines whether the fields in the list are returned or whether all fields other than those in the list are returned.

`Exclude (optional)`

(Boolean) Depending on whether the value of `Exclude` is `True` or `False`, the `Load` method either returns the fields specified in the `FieldList` parameter or returns the fields not contained in the list.

`True` indicates that the fields in the `FieldList` are to be returned (included). Default value is `True`.

`False` indicates that the fields in `FieldList` are not to be returned (excluded). If `Exclude` is set to `False`, `Load` returns all remaining fields for the asset. That is, it returns fields other than those specified with the `FieldList` parameter.

Description

This method queries the SitePlanTree table and then loads as an object the parent page of the specified page asset. It acts like `ASSET.LOAD`.

You typically use this method 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 method determines the parent page and then loads the parent page in one database query.

Returns

`Asset` object that represents the site parent.

Exceptions

Possible error values include:

Error Text
Need type parameter
Need either ID or FIELD, VALUE parameters
Both field and value parameters must be specified
Missing required table in response: tablename
Could not communicate with server
Invalid response from server

Example

The following code loads a page asset, loads its parent page, extracts the name of the parent page asset, and then displays the name:

```
<%
set CS = CreateObject ("CSDirect.CS")
if CS.Login (strUsername, strPassword, strURL) then

   set AssetManager = CS.GetAssetManager()

   set AssetSiteParent = AssetManager.GetSiteParent("Page",
"968685129226", "", "", "id,name,template", False)

   set AssetProperties = AssetSiteParent.GetTextProperties()

   set propnames = AssetProperties.GetPropertyNames()
%>

<TABLE BORDER=3 CELLPADDING=2 cellspacing="4">
<TR>
   <TH>text property name</TH>
</TR>

<%
   for each propname in propnames
%>
<TR>
<TD><% = propname %>  </TD>
</TR>

<%
   next
%>
```

List

Returns a collection of `PropertySet` objects for a specified asset type.

Syntax

```
List(Type, [OrderBy], [FieldItems])
```

Parameters

`Type (required)`

(String) The asset type of the asset that you want to retrieve from the database. Standard asset types include `article`, `image`, `page`, `collection`, and `query`.

`OrderBy (optional)`

(String) Name of the asset column (field) used to sort the results, specified as a string.

Input. The fields to sort the list by, and whether the sort result on those fields is `ascending` or `descending`. 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 `nrank, ascending`, the list is sorted by rank starting at number *1*. If you want the list sorted by descending rank, specify `nrank, descending`.

`FieldItems (optional)`

(ObjectCollection) ObjectCollection of up to nine FieldItem objects that describe the column name/value pairs to match.

Description

The returned property set collection exactly matches the criteria specified by the `FieldItems` parameter for the supplied asset type. A FieldItem comprises the name of the field and its value. Because you are trying to match the value, you must know it beforehand. If no name/value pairs are specified as FieldItems, all column names and their associated values are returned.

Inputs

`List` accepts the following object as input: ObjectCollection

Returns

Collection of `PropertySet` objects that describe the assets returned.

Exceptions

Possible error values include:

Error Text
Need type parameter
Item in array not FieldItem
Too many items in input array
Could not communicate with server

Error Text
Invalid response from server

Example

The following code instantiates the `CS` object, logs in a user, instantiates the `AssetManager` object, creates a `FieldItem` object, and passes the `FieldItem` as an input to a list. Finally, it returns the number of assets of type `page`.

```
set CS = CreateObject ("CSDirect.CS")
if CS.Login (strUsername, strPassword, strURL) then

   set AssetManager = CS.GetAssetManager()

   set Field1 = CreateObject("CSDirect.FieldItem")
   Field1.name = "id"
   Field1.value = "968685128734"

   set FieldList = CreateObject("CSDirect.ObjectCollection")
   FieldList.AddObject(Field1)

   set AssetList = AssetManager.List("Page", "id", FieldList)
%>

There are <% =AssetList.Count %>  Assets<p>
```

See Also

Methods that operate property sets include:

GetProperty

GetPropertyNames

Load

Queries the database for the specified asset and then loads an instance of the asset into memory within the `Asset` object.

Note

This method does not load flex assets.

Syntax

```
Load(Type, [ObjectID], [Field], [Value], [FieldList], [Exclude])
```

Parameters

`Type (required)`

(String) The asset type of the asset that you want to retrieve from the database. Standard asset types include `article`, `image`, `page`, `collection`, and `query`.

Typically, you provide `Type` and `ObjectID`. When you know that a specific field/value pair can uniquely identify the asset, you can provide `Type` and the field/value pair instead. Either the `ObjectID` or the `Field` and `Value` combination, but not both, are required to load an asset.

`ObjectID (optional)`

(String) The unique identifying number that references the asset object. Not required if you use `Type` with the `Field` and `Value` paired parameters.

`Field (optional)`

(String) A field is any one of the column names for the asset. For example, standard fields for CS-Direct assets include name, template, status, description, `subtype`, `category`, `modified`, `headline`, `byline`, and `body`. Use the field name in conjunction with the `Value` parameter as the name portion of a name/value pair. The field name and its corresponding value uniquely identifies the asset to be loaded. Note that if the field/value pair that you supply identifies more than one asset, the `Load` method uses the first one that it finds.

Not required if you use `Type` with the `ObjectID` parameter.

`Value (optional)`

(String) Value that corresponds to the field specified by the `Field` parameter. Paired with the field name, this parameter uniquely identifies an asset by supplying the associated value.

Not required if you use `Type` with the `ObjectID` parameter.

`FieldList (optional)`

(String) Comma-separated list of fields that you want to include or exclude from the request for asset fields. The `Exclude` parameter, which operates on the `FieldList`, determines whether the fields in the list are returned or whether all fields other than those in the list are returned.

`Exclude (optional)`

(Boolean) Depending on whether the value of `Exclude` is `True` or `False`, the `Load` method either returns the fields specified in the `FieldList` or returns the fields that are not contained in the list.

`True` indicates that the fields in the `FieldList` are to be returned (included). Default value is `True`.

`False` indicates that the fields in the `FieldList` are not to be returned (excluded). If `Exclude` is set to `False`, `Load` returns all remaining fields for the asset; that is, it returns the fields that are not specified in the `FieldList`.

Description

Based on the specified parameters, `Load` fills the `Asset` object with data. It executes a database query to retrieve an instance of the specified asset and then saves the instance in memory.

Multiple parameters provide different ways to identify the asset to load. Enter the parameter or parameter combination that is the most convenient identifier for your application, and leave blank placeholders for the unused parameters. Although only the `Type` parameter is required, you must also supply at least the `ObjectID` or the `Field` and `Value` paired parameters. You can identify the asset that you want to load by

specifying its asset type and object ID, or by specifying its asset type combined with a field/value pair that uniquely identifies the asset.

Typically, each COM Interface client calls the `Load` method to start so that subsequent code can extract and display the loaded data on pages for your online site. All input parameters are strings except for `Exclude`, which is a Boolean value.

Note that the `Load` method does not load information from the AssetPublication table or the AssetRelationTree table. If you need information from the AssetRelationTree table, use the GetChildren method. If you need information from other auxiliary tables, you can issue queries using the object ID of the loaded asset.

Returns

`Asset` object that is loaded with the requested field values.

Exceptions

Possible error values include:

Error Text
Need type parameter
Need either ID or FIELD, VALUE parameters
Both field and value parameters must be specified
Missing required table in response: tablename
Could not communicate with server
Invalid response from server

Example

The following code loads an article asset, identifying it by type and object ID:

```
<%
set CS = CreateObject ("CSDirect.CS")
if CS.Login (strUsername, strPassword, strURL) then

   set AssetManager = CS.GetAssetManager()

   set AssetLoad = AssetManager.Load("Article", "984156689074",
"", "", "", False)

   set AssetProperties = AssetLoad.GetTextProperties()

   set properties = AssetProperties.GetPropertyNames()
%>
```

And this code loads a page asset, identifying it by type and a field/value pair:

```
<%
set CS = CreateObject ("CSDirect.CS")
```

```
if CS.Login (strUsername, strPassword, strURL) then

   set AssetManager = CS.GetAssetManager()

   set Asset = AssetManager.Load("Article", "", "name", "Nigeria-
A73-2001Mar9", "", True)

   set TextProperties = Asset.GetTextProperties()

   set propnames = TextProperties.GetPropertyNames()

%>

<%
   set BinaryProperties = Asset.GetBinaryProperties()
   set propnames = BinaryProperties.GetPropertyNames()
%>

<%
   set text = BinaryProperties.GetProperty("urlbody")
   text.Save("c:/temp/t1.txt")
%>
```

See Also

GetAssetManager

AssetSearchManager Methods

AssetSearchManager methods cover the functions required to retrieve attribute values for flex assets. The AssetSearchManager group comprises the following methods:

GetAssetCount

Returns a count of assets that exactly match the criteria specified by the `SearchState` parameter.

Syntax

```
GetAssetCount([SortItems], [MatchItems], [SearchState],
[AssetTypes], [Locale])
```

Parameters

`SortItems (Optional)`

(`ObjectCollection` object) Input parameter. Name of the list that determines sort order. This is an `ObjectCollection` of `AssetSortList` objects that describe how the returned assets are to be sorted.

The `SortItem` object has three properties:

`attributetypename`

`attributename` – Either name of attribute, sort by, or one of the following special values: `_ASSETTYPE_` (order by asset type) or `_RATING_` (order by asset rating—CS-Engage only)

`attributesortdirection` – Can be either `ascending` or `descending`

`MatchItems (Optional)`

(`ObjectCollection` object) Input parameter.Instance of the `AssetMatchItem` object. The `AssetMatchItem` object contains two properties: `assettype` and `assetid`. This is an `ObjectCollection` made up of `AssetMatchItem` objects that describe the assets to match. The list is used to create the assetset.

Specify either `MatchItems` or `SearchState`, but not both. If both parameters are supplied, `MatchItems` is used by default.

`SearchState (Optional)`

(`SearchState` object) Input parameter. `SearchState` object that contains the asset criteria to match. The `SearchState` object describes the search constraint.

Specify either `SearchState` or `MatchItems`, but not both. If both parameters are supplied, `MatchItems` is used by default.

`AssetTypes (Optional)`

(String) Input parameter. Name of a list of asset types to include in building the asset count. Comma separated list of flex asset types to match. The list has one column called `assettype`. If null, then all assets in the system are considered.

`Locale (Optional)`
(String) Language and country specification associated with the asset. Locale ensures that information and figures on a page are presented to users according to accepted conventions in their country. A locale specification comprises two-character language and country codes separated by an underscore character and enclosed in quotes. For example, for English speakers in the United States:`"en_us"`

Returns

Count of assets that match the critera specified by the `SearchState` parameter.

Exceptions

Possible error values include:

Error Text
AssetMatchItems require valid assettype and assetid properties
Invalid entry in asset match array
Need MatchItems or SearchState parameter
AssetSearchItems requires a non-empty AttributeName parameter
Invalid entry in sort item array
Could not communicate with server
Invalid response from server

Example

The following code creates a match item given asset ID and asset type, and creates a sort item given attribute name and attribute type name. It then passes the `MatchItem` and item objects as input to `GetAssetCount`. Lastly, the code displays the return value from `GetAssetCount`, which is the total number of assets.

```
<%
set CS = CreateObject ("CSDirect.CS")
if CS.Login (strUsername, strPassword, strURL) then

   set SearchManager = CS.GetAssetSearchManager()

   set MatchItem1 = CreateObject("CSDirect.AssetMatchItem")

   MatchItem1.assetid = "993403853236"
   MatchItem1.assettype = "Products"

   set MatchItems = CreateObject("CSDirect.ObjectCollection")

   MatchItems.AddObject(MatchItem1)

   set SortItem1 = CreateObject("CSDirect.AssetSortItem")
```

```
   SortItem1.attributename = "MinimumBalance"
   SortItem1.attributetypename = "PAttributes"

   set SortItems = CreateObject("CSDirect.ObjectCollection")
   SortItems.AddObject(SortItem1)
      numassets = SearchManager.GetAssetCount("", MatchItems, "",
"Products", "")
%>
There are <% =numassets %> assets
```

GetAssetList

Returns a collection of flex assets that exactly match the specified criteria.

Syntax

```
GetAssetList([SortItems], [MatchItems], [SearchState],
[AssetTypes] ,[Method], [MaxCount], [Locale])
```

Parameters

`SortItems (Optional)`

(`ObjectCollection` object) Name of the list that determines sort order. This is an `ObjectCollection` of `AssetSortList` objects that describe how the returned assets are to be sorted.

The `SortItem` object has three properties:

- `attributetypename`
- `attributename` – Either name of attribute, sort by, or one of the following special values: `_ASSETTYPE_` (order by asset type) or `_RATING_` (order by asset rating—CS-Engage only)
- `attributesortdirection` – Can be either `ascending` or `descending`

`MatchItems (Optional)`

(`ObjectCollection` object) Instance of the `AssetMatchItem` object, which is an `ObjectCollection` made up of `AssetMatchItem` objects that describe the assets to match. The `AssetMatchItem` object contains two properties: `assettype` and `assetid`. The list is used to create the asset set.

Specify either `MatchItems` or `SearchState`, but not both. If both parameters are supplied, `MatchItems` is used by default.

`SearchState (Optional)`

(`SearchState` object) `SearchState` object that contains the asset criteria to match. The `SearchState` object describes the search constraint.

Specify either `SearchState` or `MatchItems`, but not both. If both parameters are supplied, `MatchItems` is used by default.

`AssetTypes (Optional)`

(String) Input parameter. Name of a list of asset types to include in building the set. Comma-separated list of flex asset types to match. The list has one column called `assettype`. If null, then all assets in the system are considered.

`Method (Optional)`
(String) Must be either random or highest. Required only if the value of `maxcount` is less than the number of items described. The `method` parameter can have one of the following values:

- `random` – for random weighted selection based on rating
- `highest` – for best selection based on rating

This parameter is meaningful only for use with the CS-Engage product.

`MaxCount (optional)`
(int) Maximum number of rows to return in the list. A value of 0 (zero) indicates all. If the count is less than the number that otherwise is returned, then items are selected according to the specified `method` argument.

`Locale (Optional)`
(String) Language and country specification associated with the asset. Locale ensures that information and figures on a page are presented to users according to accepted conventions in their country. A locale specification comprises two-character language and country codes separated by an underscore character and enclosed in quotes. For example, for English speakers in the United States:`"en_us"`

Description

Returns a collection of flex assets that exactly match the criteria specified by either the `SearchState` parameter or the AssetList. The method only works with flex assets, which are assets supplied with the CS-Direct Advantage and CS-Engage products.

Returns

A collection of `PropertySet` objects that describes the assets returned.

Exceptions

Possible error values include:

Error Text
AssetMatchItems require valid assettype and assetid properties
Invalid entry in asset match array
Need match items or search state parameter
AssetSearchItems require a nonempty attributename parameter
Invalid entry in sort item array
Could not communicate with server
Invalid response from server

Example

The following code demonstrates `GetAssetList`:

```
set CS = CreateObject ("CSDirect.CS")
```

```
if CS.Login (strUsername, strPassword, strURL) then

   set SearchManager = CS.GetAssetSearchManager()

   set SearchState = CreateObject("CSDirect.SearchState")
   set RangeConstraint = CreateObject("CSDirect.ConstraintRange")

   RangeConstraint.TypeName = "PAttributes"
   RangeConstraint.Attribute = "CommissionType"
   RangeConstraint.SetLowerConstraint "2.0", false
   RangeConstraint.SetUpperConstraint "4.0", false

   SearchState.AddAndConstraint(RangeConstraint)

   set SortItem1 = CreateObject("CSDirect.AssetSortItem")

   SortItem1.attributename = "CommissionType"
   SortItem1.attributetypename = "PAttributes"

   set SortItems = CreateObject("CSDirect.ObjectCollection")

   SortItems.AddObject(SortItem1)

   set propsetcollection = SearchManager.GetAssetList(SortItems,
"", SearchState, "Products", "", 10, "fr_FR")
%>
   <TABLE BORDER=3 CELLPADDING=2 cellspacing="4">
   <TR>
      <TH>assettype </TH>
      <TH>assetid </TH>
   </TR>

<%
   for each pset in propsetcollection
%>

   <TR>
   <TD><% = pset.GetProperty("assettype") %>  </TD>
   <TD><% = pset.GetProperty("assetid") %>  </TD>
   </TR>
<%   next
%>

</TABLE>
```

GetAssetSearchManager

Retrieves an instance of the `AssetSearchManager` object.

Syntax

```
GetAssetSearchManager()
```

Parameters

None.

Description

Retrieves an instance of the `AssetSearchManager` object. The `AssetSearchManager` object is used to list and retrieve values from assets given a search criteria (`SearchState` object). Because it contains methods that are inherited by the other AssetSearchManager methods, you must retrieve and instantiate this object before using any other AssetSearchManager function.

Inputs

None.

Returns

Instance of the `AssetSearchManager` object.

Exceptions

Possible error values include:

Error Text
Not logged in.

Example

The following code instantiates the `CS` object, logs in the user, and instantiates the `SearchManager` object.

```
<%
set CS = CreateObject ("CSDirect.CS")
if CS.Login (strUsername, strPassword, strURL) then
   set SearchManager = CS.GetAssetSearchManager()

<!-- instantiate your COM Interface methods here -->
%>
```

See Also

Methods used in conjunction with the `AssetSearchManager` object:

GetAssetCount

GetAssetSearchManager

GetAttributeValues

GetMultipleValues

GetAttributeValues

Returns a single asset value for each asset that matches the specified `SearchState` parameter.

Syntax

```
GetAttributeValues(AttributeName, [MatchItems], [SearchState],
[AssetTypes], [TypeName], [Direction], [Locale])
```

Parameters

`AttributeName (required)`
: (String) Name of the attribute to retrieve

`MatchItems (optional)`
: (`ObjectCollection` object) Input parameter. Name of a list containing the columns `assettype` and `assetid`. This is an `ObjectCollection` made up of `AssetMatchItems` objects that describe the assets to match. The list is used to create the assetset.

  Specify either `MatchItems` or `SearchState`, but not both. If both parameters are supplied, `MatchItems` is used by default.

`SearchState (optional)`
: (SearchState object) Input parameter. SearchState object that contains the Asset criteria to match. The SearchState object describes the search constraint.

  Specify either `SearchState` or `MatchItems`, but not both. If both parameters are supplied, `MatchItems` is used by default.

`AssetTypes (optional)`
: (String) Input parameter. Name of a list of asset types to include in building the set. Comma-separated list of flex asset types to match. The list has one column called `assettype`. If null, then all assets in the system are considered.

`TypeName (optional)`
: (String) Name of the attribute type, either product attribute or content attribute. Paying attention to case, specify one of the following valid attribute types: `PAttribute` or `CAttribute`. `PAttribute` indicates the product flex asset type. `CAttribute` specifies the content flex asset type.

`Direction (optional)`
: (String) Either ascending or descending.

`Locale (optional)`
: (String) Language and country specification associated with the asset. Locale ensures that information and figures on a page are presented to users according to accepted conventions in their country. A locale specification comprises two-character language

and country codes separated by an underscore character and enclosed in quotes. For example, for English speakers in the United States:`"en_us"`

Returns

Collection of `PropertySet` objects that the specified attribute for each asset that matches the criteria of the `SearchState` parameter.

Exceptions

Possible error values include:

Error Text
AssetMatchItems require valid assettype and assetid properties
Invalid entry in asset match array
Need asset parameter
Need match items or search state parameter
Could not communicate with server
Invalid response from server

Example

The following code creates a searchstate using a rich-text constraint, passes the searchstate to the `GetAttributeValues` method, and returns values of the attributes in ascending order.

```
<%
set CS = CreateObject ("CSDirect.CS")
if CS.Login (strUsername, strPassword, strURL) then

   set SearchManager = CS.GetAssetSearchManager()

   set SearchState = CreateObject("CSDirect.SearchState")
   set RichTextConstraint =
CreateObject("CSDirect.ConstraintRichText")

   RichTextConstraint.TypeName = "PAttributes"
   RichTextConstraint.Attribute = "FundFamily"
   RichTextConstraint.Confidence = 0.1
   RichTextConstraint.Value = "Penrod"

   SearchState.AddAndConstraint(RichTextConstraint)

   set PropSet = SearchManager.GetAttributeValues("FundFamily",
"", SearchState, "Products", "PAttributes", "ascending", "")

   for each propname in PropSet
%>
```

```
   <TABLE BORDER=3 CELLPADDING=2 cellspacing="4">
   <TR>
      <TH>value </TH>
   </TR>
   <TR>
   <TD><% = propname.GetProperty("value") %>  </TD>
   </TR>
<%   next
%>
</TABLE>
```

GetMultipleValues

Returns multiple asset values for each asset that matches the specified `SearchState` parameter.

Syntax

```
GetMultipleValues(Prefix, [SortItems], [MatchItems],
[SearchState], [AssetTypes], ByAsset, [Locale])
```

Parameters

`Prefix (required)`
(String) String to prefix on each property returned.

`SortItems (optional)`
(`ObjectCollection` object) Input parameter. Name of the list that determines sort order. This is an `ObjectCollection` of `AssetSortList` objects that describe how the returned assets are to be sorted.

The `SortItem` object has three properties:

`attributetypename`

`attributename` – Either name of attribute, sort by, or one of the following special values: `_ASSETTYPE_` (order by asset type) or `_RATING_` (order by asset rating; for use with the CS-Engage product only)

`attributesortdirection` – Can be either `ascending` or `descending`

`MatchItems (optional)`
(`ObjectCollection` object) Input parameter. Instance of the `AssetMatchItem` object. The `AssetMatchItem` object contains two properties: `assettype` and `assetid`. This is an `ObjectCollection` made up of `AssetMatchItem` objects that describe the assets to match. The list is used to create the assetset.

Specify either `MatchItems` or `SearchState`, but not both. If both parameters are supplied, `MatchItems` is used by default.

`SearchState (optional)`
(`SearchState` object) Input parameter. `SearchState` object that contains the Asset criteria to match. The `SearchState` object describes the search constraint.

Specify either `SearchState` or `MatchItems`, but not both. If both parameters are supplied, `MatchItems` is used by default.

`AssetTypes (optional)`
: (String) Input parameter. Name of a list of asset types to include in building the set. Comma-separated list of flex asset types to match. The list has one column called `assettype`. If null, then all assets in the system are considered.

`ByAsset (required)`
: (Boolean) Either `True` or `False`. The value indicates 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`).

`Locale (optional)`
: (String) Language and country specification associated with the asset. Locale ensures that information and figures on a page are presented to users according to accepted conventions in their country. A locale specification comprises two-character language and country codes separated by an underscore character and enclosed in quotes. For example, for English speakers in the United States:`"en_us"`

Returns

A collection of `PropertySet` objects that contain attributes for each asset that matches the criteria for the `SearchState` parameter.

Exceptions

Possible error values include:

Error Text
Need prefix parameter
Need match items or search state parameter
AssetMatchItems require valid assettype and assetid properties
Invalid entry in asset match array
AssetSearchItems require a nonempty attributename parameter
Invalid entry in sort item array
Missing required table in response: tablename
Could not communicate with server
Invalid response from server

Example

The following code creates a searchstate using a standard constraint, and creates a `SortItem` object. It passes the `SearchState` and `SortItem` objects to the `GetMultipleValues` method, and returns names and values of the attributes.

```
<%
set CS = CreateObject ("CSDirect.CS")
if CS.Login (strUsername, strPassword, strURL) then
```

```
   set SearchManager = CS.GetAssetSearchManager()

   set SearchState = CreateObject("CSDirect.SearchState")
   set StandardConstraint =
CreateObject("CSDirect.ConstraintStandard")

   StandardConstraint.TypeName = "PAttributes"
   StandardConstraint.Attribute = "FundFamily"

   SearchState.AddAndConstraint(StandardConstraint)

   set SortItem1 = CreateObject("CSDirect.AssetSortItem")
   set SortItem2 = CreateObject("CSDirect.AssetSortItem")

   SortItem1.attributename = "FundFamily"
   SortItem1.attributetypename = "PAttributes"

   SortItem2.attributename = "FundType"
   SortItem2.attributetypename = "PAttributes"

   set SortItems = CreateObject("CSDirect.ObjectCollection")

   SortItems.AddObject(SortItem1)
   SortItems.AddObject(SortItem2)

   set PropSet = SearchManager.GetMultipleValues("MyArticle",
SortItems, "", SearchState, "Products", False, "")
   set PropNames = PropSet.GetPropertyNames()

   for each propname in PropNames
%>
   <TABLE BORDER=3 CELLPADDING=2 cellspacing="4">
   <TR>
      <TH><%=propname%> </TH>
<%
   set propcollection = PropSet.GetProperty(propname)
   for each props in propcollection
%>
   <TR>
   <TD><% = props.GetProperty("value") %>  </TD>
   </TR>
<%   next
%>
   </TR>
<%next%>

</TABLE>
```

Blob Method

The following blob (binary large object) method saves binary data returned from an IPSManager blob operation to a local file.

Save

Saves binary data from a blob (binary large object) to the specified file name.

Syntax

```
Save(FileName)
```

Parameters

`FileName  (required)`
(String) Complete file name and path location to the blob object.

Description

Saves binary data from a blob (binary large object) to the file name you specify for the `FileName` argument. The Save method operates on a blob object generated by either a `GetBlob` or `GetMungoBlob` operation. Before calling the `Save` method, you must retrieve the blob asset by calling the `GetBlob` method or the `GetMungoBlob` method, as appropriate.

Returns

Void.

Exceptions

Possible error values include:

Error Text
Missing required file name parameter
Binary data decode error

Example

The following code instantiates the `IPSManager` object, returns an image, and saves it to local disk.

```
<%
   set IPSManager = CS.GetIPSManager()
   set image = IPSManager.GetBlob("ImageFile", "urlpicture", "id",
"985668092853", "image/jpeg")
   image.Save("c:/temp/g1.jpg")
%>
```

See Also

GetBlob

GetMungoBlob

IPSManager Methods

IPSManager methods cover basic Content Server functions such as returning a blob stored in a particular table. Separate methods return blobs for standard assets and flex assets. An additional method searches indexes generated by a search engine. Each of these methods requires that you instantiate the `IPSManager` object.

The IPSManager group comprises the following methods:

GetBlob

Returns a blob (binary large object) stored in Content Server.

Syntax

```
GetBlob(BlobTable, BlobCol, BlobKey ,BlobWhere, [BlobHeader])
```

Parameters

`BlobTable (required)`
: (String) The name of the CS-Direct table that stores assets of this type. Typically, this is the CS-Direct `ImageFile` asset type. If you create your own asset type, specify that asset type instead.

`BlobCol (required)`
: (String) The name of the column that contains the binary data. Typically, this is the CS-Direct `urlpicture` column.

  The `BlobKey` parameter is required when you specify `BlobCol`. They are passed together as a name/value pair.

`BlobKey (required)`
: (String) The name of the column used as the primary key. Typically, this is `id`, unless otherwise defined.

  The `BlobCol` parameter is required when you specify BlobKey. They are passed together as a name/value pair.

`BlobWhere (required)`
: (String) Allows the calling program to select a blob based on the database field in which the SQL where operates. You can specify any column name as the location for the where clause.Typically, the value used is the object ID for the image file.

`BlobHeader (Optional)`
: (String) Description of the image format, which corresponds to the mimetype for returned data in the form *`description/extension`*. Specify any one of the following possible values: `image/jpeg`, `image/gif`, `image/jpg`.

Returns

Blob object; for example, a PDF file.

Exceptions

Possible error values include:

Error Text
Need Blob Table parameter
Need Blob Column parameter
Need Blob Key parameter
Need Blob Where parameter
Could not communicate with server
Invalid response from server

Example

The following code loads an article identified by object ID, gets the text properties and binary properties, and loads the blob and saves it to a local disk. Finally, it displays the blob using the HTML image tag.

As required for all blob methods, the following code begins by instantiating the `CS` object, logging in the Content Server user, and instantiating the `IPSManager` object.

```
set CS = CreateObject ("CSDirect.CS")
if CS.Login (strUsername, strPassword, strURL) then

   set AssetManager = CS.GetAssetManager()

   set Asset = AssetManager.Load("Article", "984156689074", "",
"", "", False)

   set TextProperties = Asset.GetTextProperties()

   set propnames = TextProperties.GetPropertyNames()

%>
<TABLE BORDER=3 CELLPADDING=2 cellspacing="4">
<TR>
   <TH>text property name</TH>
</TR>

<%
   for each propname in propnames
%>
<TR>
<TD><% = propname %>  </TD>
</TR>

<%
   next
%>
```

```
</table>

name property value is: <%=TextProperties.GetProperty("name")%>
<BR><BR>

<%
   set BinaryProperties = Asset.GetBinaryProperties()
   set propnames = BinaryProperties.GetPropertyNames()
%>
<TABLE BORDER=3 CELLPADDING=2 cellspacing="4">
<TR>
   <TH>binary property name</TH>
</TR>

<%   for each propname in propnames
%>
<TR>
<TD><% = propname %>  </TD>
</TR>

<%
   next
%>

<%
   set text = BinaryProperties.GetProperty("urlbody")
   text.Save("c:/temp/t1.txt")
%>

<%
   set IPSManager = CS.GetIPSManager()
   set image = IPSManager.GetBlob("ImageFile", "urlpicture", "id",
"985668092853", "image/jpeg")
   image.Save("c:/temp/g1.jpg")
%>

<TABLE BORDER=3 CELLPADDING=2 cellspacing="4">
<TR>
   <TH>name</TH><TH>description</TH><TH>Body</TH><TH>Image</TH>
</TR>
<TR>
<TD><% = TextProperties.GetProperty("name") %>  </TD>
<TD><% = TextProperties.GetProperty("description") %>  </TD>
<TD><A HREF=c:/temp/t1.txt>BodyText</A>  </TD>
<TD><IMG SRC=c:/temp/g1.jpg></IMG>  </TD>
</TR>


<%
else
   response.write ("login failed")
end if
```

```
%>
```

See Also

The GetBlob method is used in conjunction with the Save method.

GetIPSManager

Retrieves an instance of the `IPSManager` object.

Syntax

```
GetIPSManager()
```

Parameters

None.

Description

The `IPSManager` object contains methods for retrieving blobs (binary objects) and performing Content Server search functions. Because it contains methods that are inherited by the other IPSManager methods, you must instantiate this object before using the functions that return binary objects or before using the Search method.

No other objects provide input to `GetIPSManager`.

Returns

Instance of the `IPSManager` object.

Exceptions

Possible error values include:

Error Text
Not logged in.

Example

The following code instantiates the `CS` object, logs in the Content Server user, and instantiates the `IPSManager` object.

```
<%
set CS = CreateObject ("CSDirect.CS")
if CS.Login (strUsername, strPassword, strURL) then
   set IPSManager = CS.GetIPSManager()
<!-- instantiate additional blob methods here -->
else
   response.write ("login failed")
end if
%>
```

See Also

Methods used in conjunction with the `IPSManager` object:

GetBlob

GetMungoBlob

Search

GetMungoBlob

Returns a blob (binary large object) stored as a Content Server flex asset.

Syntax

```
GetMungoBlob(BlobID)
```

Parameters

`BlobID (Required)`
(String) Object ID for the mungoblob flex asset.

Exceptions

Possible error values include:

Error Text
Need Blob Key parameter
Could not communicate with server
Invalid response from server

Example

The following code loads and saves a mungo blob (PDF file) to the local disk and displays the PDF file as an HTML hyperlink.

As required for all blob methods, the example begins by instantiating the `CS` object, logging in the Content Server user, and instantiating the `IPSManager` object.

```
set CS = CreateObject ("CSDirect.CS")
if CS.Login (strUsername, strPassword, strURL) then
%>

<%
   set IPSManager = CS.GetIPSManager()
   set image = IPSManager.GetMungoBlob("1011495632110")
   image.Save("c:/temp/mungoblob.pdf")
%>

<TABLE BORDER=3 CELLPADDING=2 cellspacing="4">
<TR>
   <TH>MungoBlob</TH>
```

```
</TR>
<TR>
<TD><A HREF=c:/temp/mungoblob.pdf>MungoBlob</A>  </TD>
</TR>


<%
else
   response.write ("login failed")
end if

%>
```

See Also

The GetMungoBlob method is used in conjunction with the Save method.

Search

Searches a specified index from either the AltaVista or Verity search engines.

Syntax

```
Search(What, [Relevance],  [QueryParser], [Index], [Charset],
[SearchEngine], [MaxResults])
```

Parameters

`What (Required)`
: (String) Query to submit. The query is a `What` clause that contains search criteria in the language of the search-engine parser.

`Relevance (optional)`
: (String) Search engine relevance term in string format. Can be null.

`QueryParser (optional)`
: (String) Name of the search engine query parser to use.

  The AltaVista search engine supports three search types: `Simple`, `Advanced`, and `Combined`. `Advanced` is the default option if no `QueryParser` parameter is specified.

  The Verity search engine supports three query parsers: `Simple`, `FreeText`, and `BoolPlus`. The `QueryParser` parameter tells the Verity search engine the syntax of the `What` clause. If the `QueryParser` argument is omitted in the call, then the `Simple` parser is used by default.The parser may be specified by the `verity.parser` property in the futuretense.ini file.

`Index (optional)`
: (String) The name of the search index to search. If null, the default index is specified in the ContentServer properties `av.defaultindex` or `verity.defaultindex`, as appropriate.

`Charset (optional)`
: (String) Name of the character set to use for the search.

`SearchEngine (optional)`
: (String) Name of the search engine to use.

`MaxResults (optional)`
: (int) The maximum number of results to return in type integer. Set this to *0* (zero) for unlimited results.

Description

The `Search` method searches an index via the Content Server ICS Search utility. The resultset of the query is stored in a list.

Returns

Returns the name/value pairs of the resultset as a PropertySet collection. If an error occurs, or if no results satisfy the query, then the `Search` method returns a null value.

Exceptions

Possible error values include:

Error Text
Could not communicate with server
Invalid response from server

Example

The following code searches for an AltaVista index given a query specified by the `What` parameter and an index. The example assumes that the AltaVista search is configured for article assets. The example code performs four different searches: a simple search with MaxResults set to `10`, a default (`advanced`) search with MaxResults set to `6`, an advanced search with MaxResults set to `10`, and a combined (simple and advanced) search with MaxResults set to `10`.

```
<%
dim CS
dim search

set CS = CreateObject ("CSDirect.CS")
if CS.Login (strUsername, strPassword, strURL) then
   set IPSManager = CS.GetIPSManager()
   set search = IPSManager.Search("description:Nigeria", True,
"Simple", "c:/ftws/shared/sedb/Article", "", "AV", 10)
%>
Total AV Index Simple search results for description:Nigeria are -
-> <%=search.Count%><BR><BR>

<%   set search = IPSManager.Search("description:Clinton", True,
"",
"c:/ftws/shared/sedb/Article", "", "AV", 6)
%>
Total AV Index default search results for description:Clinton with
maxcount 6 are --> <%=search.Count%><BR><BR>
```

```
<%   set search = IPSManager.Search("updatedby:user", True,
"Advanced", "c:/ftws/shared/sedb/Article", "", "AV", 10)
%>
Total AV Index Advanced search results for updatedby:user are -->
<%=search.Count%><BR><BR>

<%   set search = IPSManager.Search("description:is", True,
"Combined", "c:/ftws/shared/sedb/Article", "", "AV", 10)
%>
Total AV Index Combined search results for description:is are -->
<%=search.Count%><BR><BR>

<%
else
   response.write ("login failed")
end if
%>
```

ObjectCollection Method

The AddObject method adds an object to the `ObjectCollection` object.

AddObject

Adds an object to the `ObjectCollection` instance. ObjectCollection objects are passed as input to the `List` method and the AssetSearchManager methods.

Syntax

`AddObject (object)`

Parameters

`Object (required)`
(Object) COM object to add.

Description

Adds one of the COM objects to the `ObjectCollection` object.

Returns

Void.

Exceptions

There are no errors returned for the `AddObject` method.

Example

The following code creates an `AssetSortItem` object, adds attribute name and attribute type name, creates an `ObjectCollection` object, and using the `AddObject` method adds the `AssetSortItem` object to the object collection.

```
<%
set SortItem1 = CreateObject("CSDirect.AssetSortItem")
   SortItem1.attributename = "MinimumBalance"
   SortItem1.attributetypename = "PAttributes"
   set SortItems = CreateObject("CSDirect.ObjectCollection")
   SortItems.AddObject(SortItem1)
%>
```

See Also

GetAssetCount

GetAssetSearchManager

GetAttributeValues

GetMultipleValues

List

ObjectCollection

PropertySet Methods

A PropertySet is a collection of name/value pairs. PropertySet methods, which operate on PropertySet objects, return column names and values. Once you identify the property name, you can use it to return its value.

The PropertySet group comprises the following methods:

- GetProperty
- GetPropertyNames

The following methods return PropertySet objects:

- GetAssetList
- GetAttributeValues
- GetBinaryProperties
- GetChildren (AssetManager)
- GetChildren (SitePlan)
- GetListProperties
- GetMultipleValues
- GetProperties
- GetTextProperties
- List
- Search

GetProperty

Retrieves the value of a specified property from a `PropertySet` object.

Syntax

```
GetProperty(PropertyName)
```

Parameters

`PropertyName (required)`
(String) Property name to retrieve.

Description

Retrieves the value of the specified property in Variant format.

Returns

Specified property as a Variant data type. If the property does not exist, returns an empty string instead.

Exceptions

No error text is returned.

Example

The following code instantiates the `CS` object, logs the user in, instantiates the `AssetManager` object, and loads a page asset named home. The `GetTextProperties` method returns text properties for the loaded asset and outputs the required `PropertySet` object. In turn, the `GetProperty` method returns the property values contained in the `PropertySet` object, in this case `name`, `description`, `updatedby`, and `updateddate`.

```
<%
set CS = CreateObject ("CSDirect.CS")
if CS.Login (strUsername, strPassword, strURL) then

   set AssetManager = CS.GetAssetManager()

   set AssetLoad = AssetManager.Load("Page", "", "name", "Home",
"name,description,updatedby,updateddate", True)

   set AssetProperties = AssetLoad.GetTextProperties()

%>

<TABLE BORDER=3 CELLPADDING=2 cellspacing="4">
<TR>
   <TH>name</TH><TH>description</TH><TH>updatedby</
TH><TH>updateddate</TH>
</TR>
<TR>
<TD><% = AssetProperties.GetProperty("name") %>  </TD>
<TD><% = AssetProperties.GetProperty("description") %>  </TD>
<TD><% = AssetProperties.GetProperty("updatedby") %>  </TD>
<TD><% = AssetProperties.GetProperty("updateddate") %>  </TD>
</TR>

<%
else
   response.write ("login failed")
end if

%>
```

GetPropertyNames

Retrieves property names included in the `PropertySet` object.

Syntax

```
GetPropertyNames()
```

Parameters

None.

Returns

Collection of strings.

Exceptions

No error text is returned.

Example

The following code instantiates the `CS` object, logs the user in, instantiates the `AssetManager` object, and loads a page asset named home. The `GetTextProperties` method returns text properties for the loaded asset and outputs the required `PropertySet` object. In turn, the `GetPropertyNames` method returns the property names contained in the `PropertySet` object.

```
<%
set CS = CreateObject ("CSDirect.CS")
if CS.Login (strUsername, strPassword, strURL) then

   set AssetManager = CS.GetAssetManager()

   set AssetLoad = AssetManager.Load("Page", "", "name", "Home",
"name,description,updatedby,updateddate", True)

   set AssetProperties = AssetLoad.GetTextProperties()
   set properties = AssetProperties.GetPropertyNames()
%>
```

See Also

GetAssetList

GetAttributeValues

GetProperties

GetBinaryProperties

GetChildren (AssetManager)

GetChildren (SitePlan)

GetListProperties

GetTextProperties

List

Search

SearchState Methods

SearchState methods are used to add data to a `SearchState` object. In turn, the `SearchState` object is an input to the AssetSearchManager methods. Searchstates and the AssetsearchManager methods apply to flex assets only.

The SearchState group comprises the following methods:

- AddAndConstraint
- AddOrConstraint
- SetLowerConstraint
- SetUpperConstraint

AddAndConstraint

Adds a search constraint that will be appended as an `AND` operation to other constraints contained in the `SearchState` instance.

Syntax

`AddAndConstraint(`*`SearchConstraint`*`)`

Parameters

`SearchConstraint (required)`

(SearchConstraint object) Instance of a constraint object to add to the `SearchState` object.

Returns

Void.

Exceptions

Possible error values include:

Error Text
Unrecognized constraint type
Constraint needs Attribute property
Need lower constraint
Need upper constraint
Need confidence property
Need at least one value
Need search state property
Need bucket property

Example

The following code instantiates the `SearchState` object, creates a standard constraint and, using the `AddAndConstraint` method, adds the standard constraint to the `SearchState` object.

```
<%
set SearchManager = CS.GetAssetSearchManager()

   set SearchState = CreateObject("CSDirect.SearchState")
   set StandardConstraint =
CreateObject("CSDirect.ConstraintStandard")

   StandardConstraint.TypeName = "PAttributes"
   StandardConstraint.Attribute = "FundFamily"
   SearchState.AddAndConstraint(StandardConstraint)
%>
```

AddOrConstraint

Add a search constraint that will be appended as an `OR` operation to other constraints contained in the `SearchState` instance.

Syntax

`AddOrConstraint(`*`SearchConstraint`*`)`

Parameters

`SearchConstraint (required)`
(SearchConstraint object) Instance of a constraint object to add to the `SearchState` object.

Returns

Void.

Exceptions

Possible error values include:

Error Text
Unrecognized constraint type
Constraint needs Attribute property
Need lower constraint
Need upper constraint
Need confidence property
Need at least one value
Need search state property

Error Text
Need bucket property

Example

The following code instantiates the `SearchState` object, creates a standard and like constraint and, using the `AddOrConstraint` method, adds the standard constraint to the `SearchState` object.

```
<%
set SearchManager = CS.GetAssetSearchManager()

   set SearchState = CreateObject("CSDirect.SearchState")
   set StandardConstraint =
CreateObject("CSDirect.ConstraintStandard")

   StandardConstraint.TypeName = "PAttributes"
   StandardConstraint.Attribute = "Name"
   StandardConstraint.ImmediateOnly = true

   SearchState.AddOrConstraint(StandardConstraint)

   set LikeConstraint = CreateObject("CSDirect.ConstraintLike")

   LikeConstraint.TypeName = "PAttributes"
   LikeConstraint.Attribute = "CommissionType"
   LikeConstraint.ImmediateOnly = true

   SearchState.AddOrConstraint(LikeConstraint)
%>
```

SetLowerConstraint

Sets the lower-end value in the search range.

Syntax

`SetLowerConstraint(`*`value`*`, `*`inclusive`*`)`

Parameters

`value (required)`
: (String) Value that represents the lower end of the search range.

`inclusive (required)`
: (Boolean) Value that is `True` if the lower constraint may equal the specified value. If `False`, the constraint must be greater than the specified value.

Returns

Void.

Exceptions

There are no error values returned for `SetLowerConstraint`.

Example

The following code instantiates the `SearchState` object, creates a range constraint using the `ConstraintRange` object, sets the lower limit with the `SetLowerConstraint` method, sets the upper limit with the `SetUpperConstraint` method, and adds the range constraint to the `SearchState` object.

```
<%
set SearchManager = CS.GetAssetSearchManager()

   set SearchState = CreateObject("CSDirect.SearchState")
   set RangeConstraint = CreateObject("CSDirect.ConstraintRange")

   RangeConstraint.TypeName = "PAttributes"
   RangeConstraint.Attribute = "CommissionType"
   RangeConstraint.SetLowerConstraint "2.0", false
   RangeConstraint.SetUpperConstraint "4.0", false

   SearchState.AddAndConstraint(RangeConstraint)
%>
```

SetUpperConstraint

Sets the upper-end value in the search range.

Syntax

`SetUpperConstraint(`*`value`*`, `*`inclusive`*`)`

Parameters

`value (required)`
: (String) Value that represents the upper end of the search range

`inclusive (required)`
: (Boolean) Value that is `True` if the upper constraint can equal the specified value. If `False,` the constraint must be less than the specified value.

Returns

Void.

Exceptions

There are no error values returned for `SetUpperConstraint`.

Example

The following code instantiates the `SearchState` object, creates a range constraint using the `ConstraintRange` object, sets the lower limit with the `SetLowerConstraint`

method, sets the upper limit with the `SetUpperConstraint` method, and adds the range constraint to the `SearchState` object.

```
<%
set SearchManager = CS.GetAssetSearchManager()

   set SearchState = CreateObject("CSDirect.SearchState")
   set RangeConstraint = CreateObject("CSDirect.ConstraintRange")

   RangeConstraint.TypeName = "PAttributes"
   RangeConstraint.Attribute = "CommissionType"
   RangeConstraint.SetLowerConstraint "2.0", false
   RangeConstraint.SetUpperConstraint "4.0", false

   SearchState.AddAndConstraint(RangeConstraint)
%>
```

Session Management Methods

SessionManager methods log users in and out of sessions. An active session must be established to use the COM Interface methods. The following methods manage sessions:

- Login
- Logout

Login

Logs a specified user in to the CSEE server.

Syntax

```
void
Login(Username,Password,Server)
```

Parameters

`Username (required)`
: (String) Name of the user to log in.

`Password(required)`
: (String) Password for the specified user name.

`Server(required)`
: (String) Servlet path for the target machine to which you want to connect. For example, `http://galaxy/Servlet`.

Description

This method verifies the user credentials and creates a COM session. If the credentials are valid, `Login` creates a COM session. Before you can use any other methods, you must have an active COM session. The user name, password, and server name inputs are strings.

Returns

The `Login` method returns one of the following Boolean values:

`True`
: Success—user credentials were verified and accepted, and the user was logged into Content Server.

`False`
: User credentials were invalid and the log-in attempt failed.

`Hresult`
: Exception thrown on error.

Exceptions

Possible error values include:

Error Text
Missing Username parameter
Need Password parameter
Need ServerURL parameter
Invalid ServerURL syntax
Could not communicate with server
Invalid response from server

Example

The following code logs in a Content Server user.

```
<%
set CS = CreateObject ("CSDirect.CS")
if CS.Login (strUsername, strPassword, strURL) then
<!-- instantiate your COM Interface methods here -->
else
   response.write ("login failed")
end if
%>
```

See Also

Logout

Logout

Logs out the specified user.

Syntax

```
void Logout()
```

Parameters

None.

Returns

Void.

Exceptions

Possible error values include:

Error Text
Not logged in.

Example

The following code logs out a Content Server user:

```
<%
set CS = CreateObject ("CSDirect.CS")
if CS.Logout (strUsername, strPassword, strURL) then
else
   response.write ("logout failed")
end if
%>
```

See Also

Login

SitePlan Methods

SitePlan methods enable you to examine the page hierarchy of the site. In conjunction with each other, they return the children and properties of any node in the hierarchy.

The SitePlan group comprises the following methods:

- GetChildren
- GetProperties
- GetSitePlanManager

GetChildren

Returns a collection of child nodes for the specified SitePlan node.

Syntax

```
GetChildren(NodeID, [ChildType], [ChildID], [Code],[OrderBy])
```

Parameters

`NodeID (required)`
(String) ID of the node for which you want to return child nodes.

`ChildType (optional)`
(String) Restricts the list to nodes of a specific child-asset type, which must be `Page`. You can combine this parameter with the `ChildID` parameter to request one specific node.

`ChildID (optional)`
(String) The object ID of the specific child node to return. If you supply a child ID, you must also use the associated `ChildType` parameter. If no `ChildID` parameter is specified, the `GetChildren` method returns all children for the asset.

`Code (optional)`
(String) The name of the association that describes the relationship between the child node and the parent node. Restricts the list of children to children whose pages are designated as either placed or unplaced relationships. Placed pages are those that have been set to a particular level in the tree hierarchy. Unplaced pages are those pages that have yet to be placed, or that are intentionally left unplaced.

Valid values are either `Placed` or `Unplaced`.

`OrderBy (optional)`
(String) Name of the asset column (field) used to sort the results, specified as a string.

The fields to sort the list by, and whether the sort result on those fields is `ascending` or `descending`. 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 `nrank, ascending`, the list is sorted by rank starting at number *1*. If you want the list sorted by descending rank, specify `nrank, descending`.

Description

This method 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 can restrict the list of children nodes by association name (`Code`), `ChildType`, `ChildID`, and rank (`Order`).

If you specify `Page` for the `ChildType` parameter, the list of children is a join of the SitePlanTree and the Page table.

Returns

Collection of property sets that describe the SitePlan child nodes.

Exceptions

Possible error values include:

Error Text
Need ID parameter
Could not communicate with server
Invalid response from server

Example

As required for all SitePlan methods, this example code begins by instantiating the `CS` object, logging in the user, and instantiating the `SitePlan` object. Subsequent code extracts the child pages for the page asset identified as the node ID (`nid`) of the page, restricts those child pages by child type and child ID, orders the child pages by node ID, and restricts the list to include only placed pages:

```
<%
set CS = CreateObject ("CSDirect.CS")
if CS.Login (strUsername, strPassword, strURL) then

   set SitePlanManager = CS.GetSitePlanManager()

   set AssetChildrenList =
SitePlanManager.GetChildren("968685129229",
"Page","968695082886","Placed","nid")
%>

There are <% =AssetChildrenList.Count %>  Siteplan children<p>
<TABLE BORDER=3 CELLPADDING=2 cellspacing="4">
<TR>
   <TH>nid</TH><TH>ncode</TH>
</TR>
<% for each asset in AssetChildrenList
%>
<TR>
<TD><% = asset.GetProperty("nid") %>  </TD>
```

```
<TD><% = asset.GetProperty("ncode") %>  </TD>
</TR>
<%
next

set AssetProperties =
SitePlanManager.GetProperties("968685128069")
%>
<BR><BR>
<TABLE BORDER=3 CELLPADDING=2 cellspacing="4">
<TR>
   <TH>nrank</TH><TH>ncode</TH>
</TR>
<TR>
<TD><% = AssetProperties.GetProperty("nrank") %>  </TD>
<TD><% = AssetProperties.GetProperty("ncode") %>  </TD>
</TR>

<%
else
   response.write ("login failed")
end if
%>
```

GetProperties

Retrieves the properties for the specified node ID.

Syntax

```
GetProperties(NodeID)
```

Parameters

`NodeID (Required)`

(String) ID number of the parent siteplan node. Content Server generates the node ID.

Description

Retrieves the properties for the specified node ID. To get the NodeID, first call the Load method to retrieve the asset, and then call the GetChildren method to extract the node ID number.

Returns

PropertySet that describes the siteplan node properties

Exceptions

Possible error values include:

Error Text
Need ID parameter
Could not communicate with server
Invalid response from server

Example

As required for all SitePlan methods, this example code begins by instantiating the `CS` object, logging in the user, and instantiating the `SitePlan` object. Subsequent code extracts the child pages for the page asset identified as the node ID (`nid`) of the page, restricts those child pages by child type and child ID, restricts the list to include only placed pages, and retrieves and displays properties in an HTML table:

```
<%
set CS = CreateObject ("CSDirect.CS")
if CS.Login (strUsername, strPassword, strURL) then

   set SitePlanManager = CS.GetSitePlanManager()

   set AssetChildrenList =
SitePlanManager.GetChildren("968685129229", "Page",
"968695082886", "Placed", "")
%>

There are <% =AssetChildrenList.Count %>  Siteplan children<p>
<TABLE BORDER=3 CELLPADDING=2 cellspacing="4">
<TR>
   <TH>nid</TH><TH>ncode</TH>
</TR>
<% for each asset in AssetChildrenList
%>
<TR>
<TD><% = asset.GetProperty("nid") %>  </TD>
<TD><% = asset.GetProperty("ncode") %>  </TD>
</TR>
<%
next

set AssetProperties =
SitePlanManager.GetProperties("968695082737")
%>
<BR><BR>
<TABLE BORDER=3 CELLPADDING=2 cellspacing="4">
<TR>
   <TH>nrank</TH><TH>ncode</TH>
</TR>
<TR>
<TD><% = AssetProperties.GetProperty("nrank") %>  </TD>
<TD><% = AssetProperties.GetProperty("ncode") %>  </TD>
```

```
</TR>

<%
else
   response.write ("login failed")
end if
%>
```

GetSitePlanManager

Retrieves an instance of the `SitePlanManager` object.

Syntax

```
GetSitePlanManager()
```

Parameters

None.

Description

Retrieves an instance of the `SitePlanManager` object. The `SitePlanManager` is used to retrieve site plan nodes and properties. Because it contains methods that are inherited by the other SitePlanManager methods, you must instantiate this object before using related SitePlan node functions.

Inputs

No other inputs except strings and numbers. No other COM objects pass information to these strings.

Returns

Instance of the `SitePlanManager` object.

Exceptions

Possible error values include:

Error Text
Not logged in.

Example

This example code instantiates the `CS` object, logs in the user, and instantiates the `SitePlan` object:

```
<%
set CS = CreateObject ("CSDirect.CS")
if CS.Login (strUsername, strPassword, strURL) then
```

```
   set SitePlanManager = CS.GetSitePlanManager()

<!--instantiate your COM Interface methods here -->

<%
else
   response.write ("login failed")
end if
%>
```

See Also

Methods used in conjunction with the `SitePlanManager` object:

GetChildren

GetProperties

Chapter 3

Objects

This chapter describes the input and output objects used with the COM Interface methods.

CS Object

The CS object contains the methods that enable instances of all other objects (Asset, AssetSearch, SitePlan, and IPSManager). The CS object supplies common methods to other methods. Because the rest of the COM methods inherit from this object, it must be instantiated first in each client program you write.

CS

Supplies the methods that manage user sessions, including creating Content Server sessions, and retrieving assets.

Description

Contains methods that manage the user session, including creating Content Server sessions, and retrieving assets. It also enables instances of the Asset, AssetSearch, SitePlan, and IPSManager objects.

Example

Before you can call any of the other COM Interface methods, you must create the `CS` object. The following code instantiates the `CS` object.

```
<%
set CS = CreateObject ("CSDirect.CS")
%>
```

Support Objects

The COM Interface also relies on several support objects, which are required to use its methods. These objects can be inputs to other methods or outputs from methods. For a given object, you can either call methods that access its properties or call other methods that accept the object as input.

Objects are either created using supplied COM Interface methods or directly instantiated in your ASP code. Objects are used to build searchstates and as inputs to ObjectCollections.

The relationship of these support objects to COM Interface methods is shown below.

Table 2: Objects as Input and Output Arguments to Methods

Object	Method Type	Input to	Output from
Asset	AssetManager		Load GetSiteParent
PropertySet	AssetManager		List GetChildren

Object	Method Type	Input to	Output from
	AssetSearchManager		GetAttributeValues GetAttributeValues GetMultipleValues
	SitePlan		GetChildren GetProperties
	Asset		GetBinaryProperties GetTextProperties GetListProperties
	IPSManager		Search
ObjectCollection	AssetManager	List	
	AssetSearchManager	GetAttributeValues GetAttributeValues GetMultipleValues GetAttributeValues	
Blob	IPSManager		GetBlob GetMungoBlob
SearchState	AssetSearchManager	GetAttributeValues GetAttributeValues GetMultipleValues GetAttributeValues	
ConstraintLike ConstraintNested ConstraintRange ConstraintRichText ConstraintStandard	SearchState	AddAndConstraint AddOrConstraint	
AssetSortItem ObjectCollection Support Objects		ObjectCollection	

AssetMatchItem

The `AssetMatchItem` object passes lists of assetid/assettype pairs to the asset search functions.

Properties

The following properties can be accessed directly from ASP page using VBScript:

`assetid`
: (String) Object ID of the object to match.

`assettype`
: (String) Name of a list of asset types included in building the asset count. Comma-separated list of flex asset types to match. For example, standard flex asset types like `products`, `ProductGroups`, `PDF`, or `DrillHierarchy`, or some flex-asset type that you have created. The list has one column called `assettype`. If null, then all assets in the system are considered.

Example

The following code creates an `AssetMatchItem` object, adds asset ID and asset type, creates an ObjectCollection object, and adds the `AssetMatchItem` object to the object collection.

```
<%
set MatchItem1 = CreateObject("CSDirect.AssetMatchItem")

   MatchItem1.assetid = "993403853236"
   MatchItem1.assettype = "Products"

   set MatchItems = CreateObject("CSDirect.ObjectCollection")

   MatchItems.AddObject(MatchItem1)
%>
```

AssetSortItem

The `AssetSortItem` object describes how attributes are to be sorted for the `AssetSearch` methods.

Properties

The following properties can be accessed directly from VBScript

`attributename`
: (String)

`attributetypename`
: (String)

`attributesortdirection`
: (String) May be `ascending` or `descending`.

Example

The following code creates an `AssetSortItem` object, adds attribute name and attribute type name, creates an `ObjectCollection` object, and adds the `AssetSortItem` object to the object collection.

```
<%
set SortItem1 = CreateObject("CSDirect.AssetSortItem")

   SortItem1.attributename = "MinimumBalance"
   SortItem1.attributetypename = "PAttributes"

   set SortItems = CreateObject("CSDirect.ObjectCollection")

   SortItems.AddObject(SortItem1)
%>
%>
```

ConstraintLike

The `ConstraintLike` object represents a *like* term, which is a wildcard search that can be added to a `SearchState` object.

Properties

The following properties may be accessed directly from an Active Server page using VBScript:

`Attribute (required)`
: (String) The name of the attribute to constrain.

`Bucket`
: (String) Bucket name for the constraint.

`TypeName (required)`
: (String) Name of the attribute type, either product attribute or content attribute. This property can be one of the following valid attribute types: `PAttribute` or `CAttribute`. `PAttribute` indicates the product flex asset type. `CAttribute` specifies the content flex asset type.

`ImmediateOnly (Boolean)`
: (String) Specify `true` to limit the search to values directly associated with the specified attribute, versus values inherited from a parent.

`CaseInsensitive`
: (Boolean) `True` value indicates that the case of the attribute will be considered during the `ConstraintLike` search. `False` indicates that the case of the attribute was not considered.

Example

The following code instantiates the `SearchState` object, creates a like constraint using the `ConstraintLike` object, and adds the like constraint to the `SearchState` object.

```
<%
set SearchManager = CS.GetAssetSearchManager()
```

```
   set SearchState = CreateObject("CSDirect.SearchState")
   set LikeConstraint = CreateObject("CSDirect.ConstraintLike")

   LikeConstraint.TypeName = "PAttributes"
   LikeConstraint.Attribute = "CommissionType"
   LikeConstraint.ImmediateOnly = true
   SearchState.AddOrConstraint(LikeConstraint)
%>
```

ConstraintNested

Enables nested searchstates to be added as searchstate criteria.

Properties

`Name`
(String) Name of the constraint.

`Bucket`
(String) Bucket name for the constraint.

`State`
(SearchState object) Object to add as a nested constraint.

Example

The following code instantiates the `SearchState` object, creates a nested constraint using the `ConstraintNested` object, and creates a standard constraint. Next, it adds the standard constraint to the nested constraint, creates a range constraint, and adds the range constraint to the nested constraint. Finally, the nested constraint is added to the `SearchState` object.

```
<%
set CS = CreateObject ("CSDirect.CS")
if CS.Login (strUsername, strPassword, strURL) then

   set SearchManager = CS.GetAssetSearchManager()

   set NestedConstraint =
CreateObject("CSDirect.ConstraintNested")

   set SearchState1 = CreateObject("CSDirect.SearchState")
   set SearchState2 = CreateObject("CSDirect.SearchState")
   set StandardConstraint =
CreateObject("CSDirect.ConstraintStandard")

   StandardConstraint.TypeName = "PAttributes"
   StandardConstraint.Attribute = "Name"
    StandardConstraint.Bucket = "outer"

   NestedConstraint.Bucket = "nested"
   NestedConstraint.SearchState = SearchState1

    set SearchState2 = CreateObject("CSDirect.SearchState")
```

```
   set RangeConstraint = CreateObject("CSDirect.ConstraintRange")

   RangeConstraint.TypeName = "PAttributes"
   RangeConstraint.Attribute = "CommissionType"
   RangeConstraint.SetLowerConstraint "2.0", false
   RangeConstraint.SetUpperConstraint "4.0", false

   SearchState2.AddAndConstraint(RangeConstraint)

   NestedConstraint.SearchState = SearchState2

    SearchState1.AddAndConstraint(NestedConstraint)

   set SortItem1 = CreateObject("CSDirect.AssetSortItem")

   SortItem1.attributename = "CommissionType"
   SortItem1.attributetypename = "PAttributes"

   set SortItems = CreateObject("CSDirect.ObjectCollection")

   SortItems.AddObject(SortItem1)

   numassets = SearchManager.GetAssetCount("", "", SearchState3,
"Products", "")
%>
There are <% =numassets %> assets

<%
   set PropSet = SearchManager.GetMultipleValues("myattr",
SortItems, "", SearchState, "Products", False, "")
   set PropNames = PropSet.GetPropertyNames()

   for each propname in PropNames
%>
   <TABLE BORDER=3 CELLPADDING=2 cellspacing="4">
   <TR>
      <TH><%=propname%> </TH>
<%
   set propcollection = PropSet.GetProperty(propname)
   for each props in propcollection
%>
   <TR>
   <TD><% = props.GetProperty("value") %>  </TD>
   </TR>
<%   next
%>
   </TR>
<%next%>

</TABLE> <br><br>
<%
else
```

```
   response.write ("login failed")
end if
%>
```

ConstraintRange

Represents a range of values to match. The configured constraint can then be added to a `SearchState` object.

Properties

The following properties may be accessed directly from an Active Server page using VBScript:

`Attribute (required)`
: (String) The name of the attribute to constrain.

`Bucket`
: (String) Bucket name for the constraint.

`TypeName (required)`
: (String) Name of the attribute type, either product attribute or content attribute. This property can be one of the following valid attribute types: `PAttribute` or `CAttribute`. `PAttribute` indicates the product flex asset type. `CAttribute` specifies the content flex asset type.

`CaseInsensitive`
: (Boolean) `True` value indicates that the case of the attribute will be considered during the `ConstraintStandard` search. `False` indicates that the case of the attribute was not considered.

Example

The following code instantiates the `SearchState` object, creates a range constraint using the `ConstraintRange` object, and adds the range constraint to the `SearchState` object.

```
<%
set SearchManager = CS.GetAssetSearchManager()

   set SearchState = CreateObject("CSDirect.SearchState")
   set RangeConstraint = CreateObject("CSDirect.ConstraintRange")

   RangeConstraint.TypeName = "PAttributes"
   RangeConstraint.Attribute = "CommissionType"
   RangeConstraint.SetLowerConstraint "2.0", false
   RangeConstraint.SetUpperConstraint "4.0", false

   SearchState.AddAndConstraint(RangeConstraint)
%>
```

ConstraintRichText

Adds an index name and rich-text expression to the list of rich-text criteria for items.

Properties

The following properties can be accessed directly from VBScript

`Attribute`
(String) The name of the attribute to constrain.

`Bucket`
Bucket name for the constraint.

`Confidence`
(Float) The minimum confidence level for the match.

`MaxCount`
The maximum number of answers desired for the match.

`Parser`
(String) Input parameter. The search-engine-dependent rich text parser to use.

`TypeName`
(String) Name of the attribute type, either product attribute or content attribute. This property can be one of the following valid attribute types: `PAttribute` or `CAttribute`. `PAttribute` indicates the product flex asset type. `CAttribute` specifies the content flex asset type.

`Value`
(String) Rich text value to match for this criteria.

Description

If the attribute name is already mentioned as part of a rich-text constraint in the searchstate, then the existing constraint will be removed first. This constraint can then be added to a `SearchState` object.

Example

The following code instantiates the `SearchState` object, creates a rich-text constraint using the `ConstraintRichText` object, and adds the rich-text constraint to the `SearchState` object.

```
<%
set SearchManager = CS.GetAssetSearchManager()

   set SearchState = CreateObject("CSDirect.SearchState")
   set RichTextConstraint =
CreateObject("CSDirect.ConstraintRichText")

   RichTextConstraint.TypeName = "PAttributes"
   RichTextConstraint.Attribute = "FundFamily"
   RichTextConstraint.Confidence = 0.1

   RichTextConstraint.Value = "Penrod"

   SearchState.AddAndConstraint(RichTextConstraint)
```

```
%>
```

ConstraintStandard

The `ConstraintStandard` object represents a simple constraint that can be added to a `SearchState` object.

Properties

The following properties may be accessed directly from an Active Server Page using VBScript:

`Attribute (required)`
(String) The name of the attribute to constrain.

`Bucket`
(String) Bucket name for the constraint.

`TypeName (required)`
(String) Name of the attribute type, either product attribute or content attribute. This property can be one of the following valid attribute types: `PAttribute` or `CAttribute`. `PAttribute` indicates the product flex asset type. `CAttribute` specifies the content flex asset type.

`ImmediateOnly`
(Boolean) Specify `True` to limit the search to values directly associated with the specified attribute, excluding values inherited from a parent.

`CaseInsensitive`
(Boolean) `True` value indicates that the case of the attribute will be considered during the `ConstraintStandard` search. `False` indicates that the case of the attribute was not considered.

Example

The following code instantiates the `SearchState` object, creates a standard constraint using the `ConstraintStandard` object, and adds the standard constraint to the `SearchState` object.

```
<%
set SearchManager = CS.GetAssetSearchManager()

   set SearchState = CreateObject("CSDirect.SearchState")
   set StandardConstraint =
CreateObject("CSDirect.ConstraintStandard")

   StandardConstraint.TypeName = "PAttributes"
   StandardConstraint.Attribute = "FundFamily"

   SearchState.AddAndConstraint(StandardConstraint)
```

FieldItem

Object used to pass field name/value pairs to the `AssetManager.List` method.

Properties

The following properties can be directly accessed directly from VBScript:

`Name`
(String) Name of the Content Server database column.

`Value`
(String) Value of the Content Server database field that corresponds to `Name`.

Example

Before you can pass name/value pairs to the `FieldItem` object you must create that object. The following code creates the `FieldItem` object:

```
<%
set Field1 = CreateObject("CSDirect.FieldItem")
   Field1.name = "id"
   Field1.value = "968685128734"

   set FieldList = CreateObject("CSDirect.ObjectCollection")
   FieldList.AddObject(Field1)
%>
```

ObjectCollection

Represents a collection of objects that is used as input to different methods. To create an ObjectCollection object, use the `CreateObject` method.

Example

The following code creates an `ObjectCollection` object.

```
<%
   set Objects = CreateObject("CSDirect.ObjectCollection")
%>
```

See Also

AddObject

GetAssetCount

GetAssetSearchManager

GetAttributeValues

GetMultipleValues

List

SearchState

Object used to pass a collection of search constraints.

See Also

`The SearchState` is an input object to the following `AssetSearchManager` methods:

- GetAttributeValues
- GetMultipleValues
- GetAttributeValues

Example

Before you can pass SearchConstraints to the `SearchState` object you must create that object. You instantiate a `SearchState` object directly in ASP by entering the following COM code:

```
Createobject(“CSDirect.SearchState)
```

Index

M

N

O

P

S