Content Server Enterprise Edition

Version: 5.5

Java API Reference

Document Revision Date: Oct. 30, 2003

FatWire SOFTWARE

Java API Reference
Document Revision Date: Oct. 30, 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

Seed

Content Server Enterprise Edition provides a large set of Java classes and methods. This *Java API Reference* describes these classes. A complete set of Javadocs also summarize these classes. In general, you will find more detailed explanations and examples in the *Java API Reference* than in the Javadocs.

This chapter summarizes the two ways in which you can invoke these Java classes and methods. It also details the soul method of the `Seed` class.

Invoking Java Methods from CSEE

You can invoke the Java classes and methods described in this book using either of the following two mechanisms:

- You can invoke these classes and methods from a traditional Java source code file. After compiling this file, you can invoke the resulting Java executable from a CSEE element (usually an XML element). You invoke the Java executable by creating a custom tag or by calling the special `CALLJAVA` XML tag. Either way, the Java source code you create must use the Seed interface, which is the topic of this chapter. The `Seed` interface has only one method, `Execute`.
- You can use these classes and methods within a JSP CSEE element. Note that in many cases, JSP tags might provide the same features as the Java API. See the *CSEE Developer's Tag Reference* for details.

Execute

This method is invoked as a result of the CALLJAVA or custom tag.

Syntax

```
public String
Execute(FTValList vIn,
        FTValList vOut)
```

Parameters

vIn

Input parameter. List of name/value pairs that appear as ARGUMENT tags in the CALLJAVA tag, or as attributes set in the custom tag.

vOut

Output parameter. List of name/value pairs that will appear as variables in the XML element upon return from the CALLJAVA tag. This is ignored when the method is called via a custom tag.

Description

When an XML element calls the CALLJAVA tag or calls a custom tag, Content Server invokes the Seed.Execute method. The way this method behaves depends on whether it was triggered by CALLJAVA or by a custom tag.

Invoking the Seed Interface with CALLJAVA

When the Seed interface is invoked directly with CALLJAVA:

- The class implementing the Seed interface is instantiated every time it is called. This allows the implementing class to be written in a non-reentrant way. This can impact performance negatively, since the class is instantiated every time.
- Values are passed back to the calling Content Server context by setting the value in the vOut FTValList. Elements of the FTValList are then turned into variables upon return from the CALLJAVA tag.
- There is no way to obtain a reference to the calling ICS context, so there is no way to access any of the Content Server servlets.
- As a direct consequence of the previous point, there is no way to pass any complex values between the class and Content Server. In particular, Content Server lists cannot be accessed from the class, and lists created by the class cannot be returned to the Content Server context.

Invoking the Seed Interface with a Custom Tag

When the Seed interface is invoked with a custom tag:

- The class implementing the Seed interface is instantiated only once. This requires that the class be completely re-entrant.
- Values that may be put into the vOut FTValList are not converted to variables. Variables are set using the Content Server servlet.
- The following three parameters are added to the vIn FTValList by Content Server when the custom tag is invoked:

  - `_ICS`: This is an ICS value that can be cast to an ICS object. This ICS object provides the Content Server context, which allows access to the servlets provided by Content Server.
  - `_TAGNAME`: the custom tag name used to invoke the class.
  - `_CDATA`: all the text found between the begin tag and the end custom tag.
- ICS methods are used to pass values between Content Server and the class. The types of values handled include variables, session variables, properties, counters, and lists.
- The class and all classes it invokes must be fully re-entrant.

Returns

A string that will be inserted into the output stream at the position of the `CALLJAVA` tag. Null or empty strings are valid. If invoked via a custom tag, this string will be appended to any other output that may be streamed back by calls to Content Server servlets, for example, by `StreamEvalBytes()`.

Example

The following example illustrates the use of `Seed` called from a custom tag:

```
import java.util.*;
import java.net.*;
import java.io.*;
import java.text.*;

import COM.FutureTense.XML.Template.Seed;
import COM.FutureTense.Interfaces.FTValList;
import COM.FutureTense.Interfaces.FTVAL;
import COM.FutureTense.Interfaces.ICS;

/**
 **  This sample class just gets the Content Server context and
 **  returns the tagname by which it was called in the variable
 **  tagname.
 **/

public class Example implements Seed
{

  public Example()
  {
      // As a check that the class is getting created..
          //
          System.out.println(“Example created.”);
  }

  public String Execute(FTValList vIn, FTValList vOut)
  {
    ICS cs = null;
    String output = “”;

    try
    {
```

```
        FTVAL obj = vIn.getVal("_ICS");
        cs = (ICS) obj.GetObj();
        String sTag = vIn.getValString("_TAGNAME");
        cs.SetVar("tagname", sTag);
      }
       catch
      {
        System.out.println("Example exception: " + e.getMessage());
      }
      return output;
    }
  }
```

Chapter 2

CacheManager

The CacheManager object maintains both the Content Server and CS-Satellite caches.

> **Note**
>
> The connection timeout to connect to CS-Satellite is set to one minute and cannot be configured.

List of Methods

The CacheManager class contains the following methods:

Table 1: List of CacheManager Methods

Method	Summary
`flushCSEngine`	Clears items from the Content Server cache.
`flushSSEngines`	Clears items from the CS-Satellite cache.
`RecordItem`	Records an item as a cache dependency for a page.
`refreshCSEngine`	Refreshes the items in the Content Server cache.
`refreshSSEngines`	Refreshes the CS-Satellite cache.
`setPagesByArg`	Creates a page inventory list based on a name/value pair.
`setPagesByDate`	Creates a page inventory list based on the date and time an item is published.
`setPagesByID`	Creates a page inventory list based on the items' ID.

CacheManager Constructor

Instantiates a CacheManager object.

Syntax

```
public CacheManager(ICS ics)
   throws InvalidParameterException
```

Parameters

`ics`
The ICS context.

Description

The `CacheManager` constructor instantiates a `CacheManager` object.

The `CacheManager` object allows automated cache refreshing, cache clearing, and cache regeneration based on a list of newly published content. `CacheManager` manages both the Content Server and CS-Satellite caches.

Throws

`InvalidParameterException`
When the number of CS-Satellite machines does not match the number of user names and passwords.

flushCSEngine

Clears items from the Content Server cache.

Syntax

```
public ftStatusCode
flushCSEngine(ICS ics,
              int mode)
```

Parameters

`ics`

The ICS context.

`mode`

Tells CacheManager whether to flush the Content Server cache, the CS-Satellite cache, or both caches. Specify one of the following values:

- `COM.FutureTense.Cache.CacheHelper._cs`—to flush the Content Server cache.
- `COM.FutureTense.Cache.CacheHelper._ss`—to flush the CS-Satellite cache.
- `COM.FutureTense.Cache.CacheHelper._both`—to flush both caches.

Description

The `flushCSEngine` method clears pages from the Content Server cache based on a list of pages logged using the `RecordItem` method.

Returns

An `ftStatusCode` with detailed information about the success or failure of the requested operation.

flushSSEngines

Clears items from the CS-Satellite cache.

Syntax

```
public ftStatusCode
flushSSEngines(ICS ics)
```

Parameters

`ics`
The ICS context.

Description

The `flushSSEngines` method clears items from the CS-Satellite cache, based on a list of pages logged using the `RecordItem` method.

Returns

An `ftStatusCode` with detailed information about the success or failure of the requested operation.

RecordItem

Records an item as a compositional dependency for a page. A cache dependency is an item which, when changed, invalidates the copy of a page that is stored in the cache.

Recording an item as a compositional dependency allows you to create a list of pages containing that dependency item.

This method has two variants, both of which record an item as a cache dependency. The difference is:

- `RecordItem (Variant 1)` lets you set the date and time that the item was updated.
- `RecordItem (Variant 2)` automatically sets the date and time that the item was updated to the current date and time.

RecordItem (Variant 1)

Records an item as a compositional dependency for a page. Use this variant if you want to set the date that the item was updated.

Syntax

```
public static boolean
RecordItem(ICS ics,
           String itemID,
           String dateModified)
```

Parameters

`ics`
: The ICS context.

`itemID`
: A string that uniquely identifies the item you want to record. For example, `Article-9876835` where `9876835` is a unique ID for an asset of type `Article`.

`dateModified`
: The SQL date format for the date and time when the item was last modified. If this parameter is left null, the current date and time are used.

Description

The `RecordItem` method records an item as a compositional dependency for a page. Use this variant if you want to set the date that the item was updated.

Returns

Returns `true` if the item has been recorded in the page inventory list. Returns `false` if the item has not been recorded or if the page inventory is disabled.

RecordItem (Variant 2)

Records an item as a compositional dependency for a page. Use this variant if you want to use the current date as the item's updated date.

Syntax

```
public static boolean
RecordItem(ICS ics,
           String itemID)
```

Parameters

`ics`
: The ICS context.

`itemID`
: A string that uniquely identifies the item you want to record. For example, `Article-9876835`, where `9876835` is a unique ID for an asset of type Article.

Description

The `RecordItem` method records an item as a compositional dependency for a page. Use this variant if you want to use the current date as the item's updated date.

Returns

Returns `true` if the item has been recorded in the page inventory list. Returns `false` if the item has not been recorded or if the page inventory is disabled.

refreshCSEngine

Rebuilds the Content Server cache.

Syntax

```
public ftStatusCode
refreshCSEngine(ICS ics,
                int mode)
  throws InvalidParameterException
```

Parameters

`ics`

The ICS context.

`mode`

Tells CacheManager whether to flush the Content Server cache or the CS-Satellite cache. Use one of the following values:

- `COM.FutureTense.Cache.CacheHelper._cs`—to flush the Content Server cache.
- `COM.FutureTense.Cache.CacheHelper._ss`—to flush the CS-Satellite cache.

To flush both caches, call the `refreshCSEngine` method twice.

Description

The `refreshCSEngine` method rebuilds the Content Server cache with updated content, based on the page inventory list.

Returns

An `ftStatusCode` containing detailed information about the success or failure of the requested operation.

Throws

`InvalidParameterException`

If the mode is set to `COM.FutureTense.Cache.CacheHelper._both`.

refreshSSEngines

Rebuilds the CS-Satellite cache.

Syntax

```
public ftStatusCode
refreshSSEngines(ICS ics)
```

Parameters

`ics`
The ICS context.

Description

The `refreshSSEngines` method rebuilds the CS-Satellite cache with updated content, based on the page inventory list.

The CS-Satellite URLs, user names, and passwords that allow the CacheManager to log in to your CS-Satellite hosts are contained in the following properties:

- `cs.satellitehosts`
- `cs.satelliteusers`
- `cs.satellitepasswords`

These properties are located in the `futuretense.ini` file.

Returns

An `ftStatusCode` with detailed information about the success or failure of the refresh operation.

setPagesByArg

Creates a page inventory list based on a name/value pair.

Syntax

```
public int
setPagesByArg(ICS ics,
              String name,
              String value)
```

Parameters

`ics`
: The ICS context.

`name`
: A variable name. You can optionally specify `pagename`.

`value`
: The value associated with `name`.

Description

The `setPagesByArg` method creates a page inventory list based on a name/value pair. All pages that have the specified `name`/`value` pair in the query string will be set.

Returns

Returns the number of pages added to the page inventory list.

setPagesByDate

Creates a page inventory list based on the date and time an item is published.

Syntax

```
public int
setPagesByDate(ICS ics,
               String sinceTime,
               boolean bPageAge)
```

Parameters

`ics`
: The ICS context.

`sinceTime`
: A date and time in JDBC format—usually the date and time when you start to publish new content.

`bPageAge`
: A value of `true` uses the age of the page to determine what gets deleted from the cache. A value of `false` uses the age of the item to determine what gets deleted from the cache.

Description

The `setPagesByDate` method creates a page inventory list based on the date and time when an item was published.

If `bPageAge` is set to `false`, items published after the date and time specified in the `sinceTime` parameter are deemed updated content. Any cached page that contains content that should be updated, and that has a created date earlier than the date specified in `sinceTime` is added to the page inventory list.

If `bPageAge` is set to `true`, pages are added to the page inventory list if the date they were created is before the date specified in `sinceTime`.

Returns

Returns the number of pages added to the page inventory list.

setPagesByID

Creates a page inventory list based on an array of IDs.

Syntax

```
public int
setPagesByID(ICS ics,
             String[] ids)
```

Parameters

`ics`
The ICS context.

`ids`
An array of item IDs.

Description

The `setPagesByID` method creates a list of pages that have the IDs specified in the `ids` parameter.

Returns

Returns the number of pages added to the page inventory list.

streamResults

Controls streaming of status codes.

Syntax

```
public void
streamResults(boolean b);
```

Parameters

`b`

Set to `true` if you want CacheManager to stream results. Set to `false` if you want CacheManager to suppress streaming messages.

Description

The `streamResults` method tells CacheManager whether or not to stream results to the browser after each page is either flushed or refreshed. Streaming *prevents* HTTP timeout errors.

CacheManager's default behavior is to *not* stream status codes to the browser. In other words, the default behavior is the same as calling `streamResults(false)`.

You may toggle between `streamResults(true)` and `streamResults(false)` between invoking other CacheManager methods.

Chapter 3

FormPoster

The `FormPoster` class provides methods for constructing and posting multipart mime-encoded messages, the same kind of message format that a Web browser can use to post the results of an HTML form.

The `FormPoster` class is the basis for the XMLPost utility, which enables you to load XML-encoded data into content catalogs managed by Content Server. If you do not want to use the XMLPost utility, you can use the FormPoster class to write your own utility.

The `FormPoster` class extends the `java.lang.Object`

List of Fields

Constant	Meaning	Code
`ERRORSUCCESS`	Success	`0`
`ERRORNOCONSERVER`	Cannot connect to server error	`-1`
`ERRORNETDISCONNECT`	Network disconnected error	`-2`
`ERRORNOURLPARAM`	Bad URL input parameters error	`-3`
`ERRORTIMEOUT`	Network timeout forced error	`-4`
`ERRORBADRESPONSE`	Server response did not contain handshake code error	`-5`
`ERRORNOTAUTHENTICATED`	Server certificate does not match what is expected error	`-6`
`ERROROUTOFMEMORY`	Out of memory reading a server response error	`-7`
`ERRORGENERAL`	General error	`-8`
`ERRORFILEIO`	File I/O error	`-9`

Constant	Meaning	Code
ERRORILLEGALDATAFORACTION	File data used with Get method error.	-10

List of Other Fields

HTTPVERSION10	HTTP 1.0 flag
HTTPVERSION11	HTTP 1.1 flag
Post	POST action
Get	GET action

Constructor

```
public FormPoster()
```

List of Methods

addFile	addFileData	addTextValue
addURL	cleanResponse	cleanRawResponse
findAllCookies	findCookie	getCookieArray
getCookies	getHTTPStatusCode	getHTTPStatusText
getLastError	getLastErrorStr	getLastErrorText
getOutputFileSpec	getRawResponse	getResponse
getServletCookies	login	post
postProcessMIMEHeader	registerMIMENotifier	reset
setAction	setAuthenticationParameters	setBodyData
setBodyStream	setCharacterEncoding	setCookies
setDump	setEncodeURL	setHeaderInformation
setHTTPVersion	setLog	setOutputDirectory
setOutputStream	setPort	setProxy
setProxyPort	setSecureProtocol	setTimeout
setURL		

addFile

The `addFile` method adds a file (upload file path specification) to the MIME message under construction, as in `INPUT TYPE=FILE`. This method has two variants:

- `addFile (Variant 1)` adds a file (upload file path specification) to the MIME message under construction and allows you to specify the MIME type.
- `addFile (Variant 2)` adds a file (upload file path specification) to the MIME message under construction.

---

addFile (Variant 1)

Adds a file (upload file path specification) to the MIME message under construction, as in `INPUT TYPE=FILE`.

Syntax

```
public boolean
addFile(String sParamName,
        String sValue,
        String sFileName,
        String sMIMEType)
```

Parameters

`sParamName`
: The parameter name.

`sValue`
: Value of the parameter (usually the file name).

`sFileName`
: Path of the file to add.

`sMIMEType`
: String containing the MIME type of the file.

Returns

A boolean value.

Example

```
  FormPoster fp = new FormPoster();
  FTValList vlCookies;
  fp.setURL("http://myserver/servlet/CatalogManager");

// Add a new row to MyTable
  ftStatusCode ftStatus = new ftStatusCode();
  status = fp.login(true, ftStatus, "myusername", "mypassword");
  vlCookies = fp.findAllCookies();
  Status = fp.addFile("url", "test.html",
                      "/export/home/temp/test.html", "text/html");
  Status = fp.addTextValue("tblname", "MyTable");
  Status = fp.addURL("myparameter", "http://server/main.html");
```

```
  Status = fp.addTextValue("id", "1323");
  Status = fp.addTextValue("ftcmd", "addrow");
  fp.setCookies(vlCookies);
  Status = fp.post();

// Fetch the response without the http headers
  CleanResponse = fp.cleanRawResponse(true);
  fp.login(false, ftStatus, null, null);
```

See Also

addTextValue()
addURL()

---

addFile (Variant 2)

Adds a file (upload file path specification) to the MIME message under construction, as in `INPUT TYPE=FILE`.

Syntax

```
public boolean
addFile(String sParamName,
        String sValue,
        String sFileName)
```

Parameters

`sParamName`
: The parameter name.

`sValue`
: Value of the parameter (usually the file name).

`sFileName`
: Path of the file to add.

Returns

A boolean value.

addFileData

The `addFileData` method adds file (upload) data. Use this method for a multipart post, for example `INPUT TYPE=FILE.`

This method has four variants:

- `addFileData (Variant 1)` adds file (upload) data using a byte array.
- `addFileData (Variant 2)` adds file (upload) data using a byte array and allows you to specify the data's MIME type.
- `addFileData (Variant 3)` adds file (upload) data using an input stream and allows MIME type.
- `addFileData (Variant 4)` adds file (upload) data using an input stream.

---

addFileData (Variant 1)

Adds file (upload) data using a byte array. Use this method for multipart posting operations, for example, `INPUT TYPE=FILE`.

Syntax

```
public boolean
addFileData(String sParamName,
            String sFileName,
            byte[] byData)
```

Parameters

`sParamName`
The parameter name.

`sFileName`
The name of the file on the target system.

`byData`
The byte array containing the file data.

Returns

Returns `true` if successful.

---

addFileData (Variant 2)

Adds file (upload) data using a byte array. Use this method for multipart posting operations; for example, `INPUT TYPE=FILE`.

Syntax

```
public boolean
addFileData(String sParamName,
            String sFileName,
            byte[] byData,
            String sMIMEType)
```

Parameters

`sParamName`
The parameter name.

`sFileName`
The name of the file on the target system.

`byData`
The byte array containing the file data.

`sMIMEType`
The MIME type of the data.

Returns

Returns `true` if successful.

---

addFileData (Variant 3)

Adds file (upload) data using an input stream. Use this method for multipart posting operations; for example, `INPUT TYPE=FILE`.

Syntax

```
public boolean
addFileData(String sParamName,
            String sFileName,
            InputStream is,
            String sMIMEType)
```

Parameters

`sParamName`
The parameter name.

`sFileName`
The name of the file on the target system.

`is`
The object containing the data you want to read.

`sMIMEType`
The MIME type of the data.

Returns

Returns `true` if successful.

---

addFileData (Variant 4)

Adds file (upload) data using an input stream. Use this method for multipart posting operations; for example, `INPUT TYPE=FILE`.

Syntax

```
public boolean
```

```
addFileData(String sParamName,
            String sFileName,
            InputStream is)
```

Parameters

`sParamName`
: The parameter name.

`sFileName`
: The name of the file on the target system.

`is`
: The object containing the data you want to read.

Returns

Returns `true` if successful.

addTextValue

Adds a name/value pair to the mime message under construction.

Syntax

```
public boolean
addTextValue(String sParamName,
             String sValue)
```

Parameters

`sParamName`
Name of the parameter.

`sValue`
Value string of the parameter

Example

```
FormPoster fp = new FormPoster();
FTValList vlCookies;
fp.setURL("http://myserver/servlet/CatalogManager");

// Add a new row to MyTable
FTStatusCode ftStatus = new FTStatusCode();
status = fp.login(true, ftStatus, "myusername", "mypassword");
vlCookies = fp.findAllCookies();
Status = fp.addFile("url", "test.html",
                    "/export/home/temp/test.html", "text/html");
Status = fp.addTextValue("tblname", "MyTable");
Status = fp.addURL("myparameter", "http://server/main.html");
Status = fp.addTextValue("id", "1323");
Status = fp.addTextValue("ftcmd", "addrow");
fp.setCookies(vlCookies);
Status = fp.post();

// Fetch the response without the http headers
CleanResponse = fp.cleanRawResponse(true);
fp.login(false, ftStatus, null, null);
```

See Also

addFile, addURL, setCharacterEncoding

addURL

Adds a URL (`http://` or `https://`) to data for multipart posting operations; as in `INPUT TYPE=FILE`.

Syntax

```
public boolean
addURL(String sParamName,
       String sURL)
```

Parameters

`sParamName`
the parameter name

`sURL`
URL to fetch

Returns

Boolean value.

Example

```
    FormPoster fp = new FormPoster();
    FTValList vlCookies;
    fp.setURL("http://myserver/servlet/CatalogManager");

    // Add a new row to MyTable
    FTStatusCode ftStatus = new FTStatusCode();
    status = fp.login(true, ftStatus, "myusername", "mypassword");
    vlCookies = fp.findAllCookies();
    Status = fp.addFile("url", "test.html",
                   "/export/home/temp/test.html", "text/html");
    Status = fp.addTextValue("tblname", "MyTable");
    Status = fp.addURL("myparameter", "http://server/main.html");
    Status = fp.addTextValue("id", "1323");
    Status = fp.addTextValue("ftcmd", "addrow");
    fp.setCookies(vlCookies);
    Status = fp.post();

// Fetch the response without the http headers
    CleanResponse = fp.cleanRawResponse(true);
    fp.login(false, ftStatus, null, null);
```

See Also

addFile, addTextValue

cleanResponse

The `cleanResponse` method retrieves the response string from the server and removes the HTTP header. This method is similar to `cleanRawResponse` except the data is returned as a string rather than a byte array.

This method has two variants:

- `cleanResponse (Variant 1)` retrieves the response string from the server and removes the HTTP header.
- `cleanResponse (Variant 2)` retrieves the response string from the server and removes the HTTP header and allows you to remove the trailing status code.

---

cleanResponse (Variant 1)

Retrieves the response string from the server and removes the HTTP header. Optionally removes the trailing status code.

Syntax

```
public String cleanResponse()
```

Description

The `cleanResponse` method retrieves the response string from the server and removes the HTTP header. This method is similar to `cleanRawResponse` except the data is returned as a string rather than a byte array.

Example

```
    FormPoster fp = new FormPoster();
    FTValList vlCookies;
    fp.setURL("http://myserver/servlet/CatalogManager");
    //---
    // Add a new row to MyTable
    FTStatusCode ftStatus = new FTStatusCode();
    status = fp.login(true, ftStatus, "myusername", "mypassword");
    vlCookies = fp.findAllCookies();
        Status = fp.addFile("url", "test.html",
                            "/export/home/temp/test.html", "text/
html");
        Status = fp.addTextValue("tblname", "MyTable");
        Status = fp.addURL("myparameter", "http://server/
main.html");
        Status = fp.addTextValue("id", "1323");
        Status = fp.addTextValue("ftcmd", "addrow");
        fp.setCookies(vlCookies);
        Status = fp.post();
        //---
        // Fetch the response without the http headers
        CleanResponse = fp.cleanRawResponse(true);
        fp.login(false, ftStatus, null, null);
```

See Also

getResponse, getRawResponse, cleanRawResponse

---

cleanResponse (Variant 2)

Retrieves the response string from the server and removes the HTTP header. Optionally removes the trailing status code.

Syntax

```
public String
cleanResponse(boolean bRemoveStatus)
```

Parameters

`bRemoveStatus`
Set this parameter to `true` to remove status; set it to `false` to keep the status code.

Description

The `cleanResponse` method retrieves the response string from the server and removes the HTTP header. `cleanResponse` optionally removes the trailing status code.

This method is similar to cleanRawResponse except the data is returned as a string rather than as a byte array.

Example

```
    FormPoster fp = new FormPoster();
    FTValList vlCookies;
    fp.setURL("http://myserver/servlet/CatalogManager");
    //---
    // Add a new row to MyTable
    FTStatusCode ftStatus = new FTStatusCode();
    status = fp.login(true, ftStatus, "myusername", "mypassword");
    vlCookies = fp.findAllCookies();
        Status = fp.addFile("url", "test.html",
                            "/export/home/temp/test.html", "text/
html");
        Status = fp.addTextValue("tblname", "MyTable");
        Status = fp.addURL("myparameter", "http://server/
main.html");
        Status = fp.addTextValue("id", "1323");
        Status = fp.addTextValue("ftcmd", "addrow");
        fp.setCookies(vlCookies);
        Status = fp.post();
        //---
        // Fetch the response without the http headers
        CleanResponse = fp.cleanRawResponse(true);
        fp.login(false, ftStatus, null, null);
```

See Also

getResponse, getRawResponse, cleanRawResponse

cleanRawResponse

The `cleanRawResponse` method fetches the raw bytes from the output stream and removes the HTTP header.

This method is similar to `getResponse()` except the data is returned as an array of bytes rather than a string.

This method has two variants:

- `cleanRawResponse (Variant 1)` fetches the raw bytes from the output stream and removes the HTTP header.
- `cleanRawResponse (Variant 2)` fetches the raw bytes from the output stream, removes the HTTP header, and optionally removes the trailing status code.

---

cleanRawResponse (Variant 1)

Fetches the raw bytes from the output stream and removes the HTTP header.

Syntax

```
public byte[]
cleanRawResponse()
```

Description

The `cleanRawResponse` method fetches the raw bytes from the output stream and removes the HTTP header.

This method is similar to `getResponse()` except the data is returned as an array of bytes rather than as a string.

Returns

```
byte[] response
```

Example

```
FormPoster fp = new FormPoster();
FTValList vlCookies;
fp.setURL("http://myserver/servlet/CatalogManager");
//---
// Add a new row to MyTable
FTStatusCode ftStatus = new FTStatusCode();
status = fp.login(true, ftStatus, "myusername", "mypassword");
vlCookies = fp.findAllCookies();
    Status = fp.addFile("url", "test.html",
                "/export/home/temp/test.html", "text/html");
    Status = fp.addTextValue("tblname", "MyTable");
    Status = fp.addURL("myparameter",
                        "http://server/main.html");
    Status = fp.addTextValue("id", "1323");
    Status = fp.addTextValue("ftcmd", "addrow");
    fp.setCookies(vlCookies);
```

```
        Status = fp.post();
        //---
        // Fetch the response without the http headers
        CleanResponse = fp.cleanRawResponse(true);
        fp.login(false, ftStatus, null, null);
```

See Also

getResponse(), getRawResponse(), cleanRawResponse()

---

cleanRawResponse (Variant 2)

Fetches the raw bytes from the output stream and removes the HTTP header. Optionally removes the trailing status code.

Syntax

```
public byte[]
cleanRawResponse(boolean bRemoveStatus)
```

Parameters

`bRemoveStatus`

Set this parameter to `true` to remove status; set it to `false` to keep the status code.

Description

The `cleanRawResponse` method fetches the raw bytes from the output stream and removes the HTTP header. `cleanRawResponse` optionally removes the trailing status code.

This method is similar to `getResponse()` except the data is returned as an array of bytes rather than as a string.

Returns

byte[] response

Example

```
    FormPoster fp = new FormPoster();
    FTValList vlCookies;
    fp.setURL("http://myserver/servlet/CatalogManager");
    //---
    // Add a new row to MyTable
    FTStatusCode ftStatus = new FTStatusCode();
    status = fp.login(true, ftStatus, "myusername", "mypassword");
    vlCookies = fp.findAllCookies();
        Status = fp.addFile("url", "test.html", "/export/home/
temp/test.html", "text/html");
        Status = fp.addTextValue("tblname", "MyTable");
        Status = fp.addURL("myparameter", "http://server/
main.html");
        Status = fp.addTextValue("id", "1323");
        Status = fp.addTextValue("ftcmd", "addrow");
```

```
fp.setCookies(vlCookies);
Status = fp.post();
//---
// Fetch the response without the http headers
CleanResponse = fp.cleanRawResponse(true);
fp.login(false, ftStatus, null, null);
```

See Also

getResponse(), getRawResponse(), cleanResponse()

findAllCookies

The `findAllCookies` method finds the cookies in an HTTP response.

This method has two variants:

- `findAllCookies (Variant 1)` finds all of the cookies in an HTTP header.
- `findAllCookies (Variant 2)`

---

findAllCookies (Variant 1)

Retrieves all cookies from the HTTP response.

Syntax

```
public static FTValList
findAllCookies()
```

Returns

`FTValList` object containing all cookie names and values.

Example

```
FormPoster fp = new FormPoster();
FTValList vlCookies;
fp.setURL("http://myserver/servlet/CatalogManager");
//---
// Add a new row to MyTable
FTStatusCode ftStatus = new FTStatusCode();
status = fp.login(true, ftStatus, "myusername", "mypassword");
vlCookies = fp.findAllCookies();
    Status = fp.addFile("url", "test.html",
          "/export/home/temp/test.html", "text/html");
    Status = fp.addTextValue("tblname", "MyTable");
    Status = fp.addURL("myparameter",
                       "http://server/main.html");
    Status = fp.addTextValue("id", "1323");
    Status = fp.addTextValue("ftcmd", "addrow");
    fp.setCookies(vlCookies);
    Status = fp.post();
    //---
    // Fetch the response without the http headers
    CleanResponse = fp.cleanRawResponse(true);
    fp.login(false, ftStatus, null, null);
```

See Also

`findCookie`

findAllCookies (Variant 2)

Retrieves all cookies from the HTTP response.

Syntax

```
public static FTValList
findAllCookies(String sData)
```

Parameters

`sData`
The HTTP response.

Returns

`FTValList` object containing all cookie names and values.

findCookie

Finds a specific cookie in the HTTP response, where the cookie name starts with the specified prefix.

Syntax

```
public static String
findCookie(String data,
           String prefix)
```

Parameters

`data`
HTTP response string.

`prefix`
Prefix string for the cookie.

Returns

The cookie string.

Example

```
  FormPoster fp = new FormPoster();
  FTValList vlCookies;
  fp.setURL("http://myserver/servlet/CatalogManager");

// Add a new row to MyTable
  FTStatusCode ftStatus = new FTStatusCode();
  status = fp.login(true, ftStatus, "myusername", "mypassword");
  String sCookie = fp.findCookie("mycookieprefix",
                                 fp.getResponse());
  Status = fp.addFile("url", "test.html",
                    "/export/home/temp/test.html", "text/html");
  Status = fp.addTextValue("tblname", "MyTable");
  Status = fp.addURL("myparameter", "http://server/main.html");
  Status = fp.addTextValue("id", "1323");
  Status = fp.addTextValue("ftcmd", "addrow");
  Status = fp.post();

// Fetch the response without the http headers
  CleanResponse = fp.cleanRawResponse(true);
  fp.login(false, ftStatus, null, null);
```

See Also

findAllCookies()

getCookieArray

Finds all cookie values in the HTTP response and returns them in an array.

Syntax

```
public static COM.FutureTense.Util.CookieWrapper[]
getCookieArray(String sData)
```

Parameters

`sData`
: The HTTP response.

Returns

Returns the cookie values in an array.

getCookies

Finds all cookie values in the HTTP response and returns them in an `FTValList`.

Syntax

```
public FTValList
getCookies(String sData)
```

Parameters

`sData`
The HTTP response.

Returns

Returns an `FTValList` containing the cookie values.

getHTTPStatusCode

Retrieves the HTTP status code from the response.

Syntax

```
public int
getHTTPStatusCode()
```

Returns

The HTTP status code.

getHTTPStatusText

Retrieves the HTTP status text from the response.

Syntax

```
public String
getHTTPStatusText()
```

Returns

A string containing the status text.

getLastError

Retrieves the last error code set.

Syntax

```
public int
getLastError()
```

errno

Possible values of errno include:

Value	Description
-10	Illegal data for action. The `get` action does not support file data, so a post operation when using the `get` action will fail with this error if file data parameters have been added.

Example

```
  FormPoster fp = new FormPoster();
  FTValList vlCookies;
  fp.setURL("http://myserver/servlet/CatalogManager");

// Add a new row to MyTable
  FTStatusCode ftStatus = new FTStatusCode();
  status = fp.login(true, ftStatus, "myusername", "mypassword");
  vlCookies = fp.findAllCookies();
  Status = fp.addFile("url", "test.html",
                   "/export/home/temp/test.html", "text/html");
  Status = fp.addTextValue("tblname", "MyTable");
  Status = fp.addURL("myparameter", "http://server/main.html");
  Status = fp.addTextValue("id", "1323");
  Status = fp.addTextValue("ftcmd", "addrow");
  fp.setCookies(vlCookies);
  Status = fp.post();
  int nLastErrorCode = fp.getLastError();

// Fetch the response without the http headers
  CleanResponse = fp.cleanRawResponse(true);
  fp.login(false, ftStatus, null, null);
```

See Also

getLastErrorStr()

getLastErrorStr

Retrieves any coded message from the status code object.

Syntax

```
public String
getLastErrorStr()
```

Returns

String containing any coded message from the status code object.

Example

```
FormPoster fp = new FormPoster();
FTValList vlCookies;
fp.setURL("http://myserver/servlet/CatalogManager");
//---
// Add a new row to MyTable
FTStatusCode ftStatus = new FTStatusCode();
status = fp.login(true, ftStatus, "myusername", "mypassword");
vlCookies = fp.findAllCookies();
    Status = fp.addFile("url", "test.html",
                  "/export/home/temp/test.html", "text/html");
    Status = fp.addTextValue("tblname", "MyTable");
    Status = fp.addURL("myparameter",
                       "http://server/main.html");
    Status = fp.addTextValue("id", "1323");
    Status = fp.addTextValue("ftcmd", "addrow");
    fp.setCookies(vlCookies);
    Status = fp.post();
    int nLastErrorCode = fp.getLastError();
    String sLastError = fp.getLastErrorStr();

    // Fetch the response without the http headers
    CleanResponse = fp.cleanRawResponse(true);
    fp.login(false, ftStatus, null, null);
```

See Also

getLastError()

getLastErrorText

Retrieves the text representation of the error returned by `getLastError()`.

Syntax

```
public String getLastErrorText()
```

Returns

Returns the text representation of the error.

getOutputFileSpec

Fetches the specified file that contains the output from a posting operation.

Syntax

```
public String getOutputFileSpec()
```

Description

The `getOutputFileSpec` method fetches the specified file that contains the output from a posting operation. This call is applicable only if `setOutputDirectory()` has been called before post.

Returns

A string.

Example

```
FormPoster fp = new FormPoster();
FTValList vlCookies;
fp.setURL("http://myserver/servlet/CatalogManager");
//---
// Add a new row to MyTable
FTStatusCode ftStatus = new FTStatusCode();
status = fp.login(true, ftStatus, "myusername", "mypassword");
vlCookies = fp.findAllCookies();
    Status = fp.addFile("url", "test.html",
         "/export/home/temp/test.html", "text/html");
    Status = fp.addTextValue("tblname", "MyTable");
    Status = fp.addURL("myparameter",
                       "http://server/main.html");
    Status = fp.addTextValue("id", "1323");
    Status = fp.addTextValue("ftcmd", "addrow");
    fp.setCookies(vlCookies);

    //---
    // Write the post results to a file
    fp.setOutputDirectory("/export/home/mydirectory");
    Status = fp.post();
    int nLastErrorCode = fp.getLastError();
    String sLastError = fp.getLastErrorStr();
    //---
    // Fetch the output file name
    String sOutputFile = fp.getOutputFileSpec();
    Fp.login(false, null, null);
```

See Also

`setOutputDirectory()`

getRawResponse

Fetches the raw bytes from the output stream.

Syntax

```
public byte[] getRawResponse()
```

Description

The `getRawResponse` method fetches raw bytes from the output stream. `getRawResponse` is similar to `getResponse`, except the result is returned as a byte array rather than a stream.

Returns

A byte[] response.

Example

```
FormPoster fp = new FormPoster();
FTValList vlCookies;
fp.setURL("http://myserver/servlet/CatalogManager");
//---
// Add a new row to MyTable
FTStatusCode ftStatus = new FTStatusCode();
status = fp.login(true, ftStatus, "myusername", "mypassword");
vlCookies = fp.findAllCookies();
    Status = fp.addFile("url", "test.html",
                 "/export/home/temp/test.html", "text/html");
    Status = fp.addTextValue("tblname", "MyTable");
    Status = fp.addURL("myparameter",
                       "http://server/main.html");
    Status = fp.addTextValue("id", "1323");
    Status = fp.addTextValue("ftcmd", "addrow");
    fp.setCookies(vlCookies);
    Status = fp.post();
    int nLastErrorCode = fp.getLastError();
    String sLastError = fp.getLastErrorStr();
    //---
    // Fetch the response
    byte[] byResponse = fp.getRawResponse();
    fp.login(false, ftStatus, null, null);
```

See Also

cleanRawResponse()
getResponse()
cleanResponse()

getResponse

Gets the string response from the server after calling `post()` to post a request.

Syntax

```
String getResponse()
```

Description

The `getResponse` method gets the string response from the server after calling `post()` to post a request. `getResponse()` is similar to `getRawResponse`, except the output is returned as a string rather than as a byte array.

Returns

A string response.

Example

```
FormPoster fp = new FormPoster();
FTValList vlCookies;
fp.setURL("http://myserver/servlet/CatalogManager");
//---
// Add a new row to MyTable
FTStatusCode ftStatus = new FTStatusCode();
status = fp.login(true, ftStatus, "myusername", "mypassword");
vlCookies = fp.findAllCookies();
    Status = fp.addFile("url", "test.html",
                  "/export/home/temp/test.html", "text/html");
    Status = fp.addTextValue("tblname", "MyTable");
    Status = fp.addURL("myparameter",
                       "http://server/main.html");
    Status = fp.addTextValue("id", "1323");
    Status = fp.addTextValue("ftcmd", "addrow");
    fp.setCookies(vlCookies);
    Status = fp.post();
    int nLastErrorCode = fp.getLastError();
    String sLastError = fp.getLastErrorStr();
    //---
    // Fetch the response
    String sResponse = fp.getResponse();
    fp.login(false, ftStatus, null, null);
```

See Also

`cleanRawResponse()`
`getRawResponse()`
`cleanResponse()`

getServletCookies

Gets cookies set by the CookieServer servlet.

Syntax

```
public javax.servlet.http.Cookie[]
getServletCookies()
```

Returns

An array containing the cookie data.

login

Logs a user in or out.

Syntax

```
public void
login(boolean login,
      ftStatusCode status,
      String user,
      String pass)
```

Parameters

`login`
Specify `true` to log in, `false` to log out.

`status`
Optional if specified return status will be set.

`username`
Username to log in.

`password`
Password associated with `username`.

Description

The `login` method logs in or logs out a user. Prior to calling `login`, you must first set the URL by calling `setURL`.

Errno

`-3500`
If the `FormPoster` can not communicate with the target server during login.

Example

```
    FormPoster fp = new FormPoster();
    FTValList vlCookies;
    fp.setURL("http://myserver/servlet/CatalogManager");

// Add a new row to MyTable
    FTStatusCode ftStatus = new FTStatusCode();
    status = fp.login(true, ftStatus, "myusername", "mypassword");
    vlCookies = fp.findAllCookies();
    Status = fp.addFile("url", "test.html",
                        "/export/home/temp/test.html", "text/html");
    Status = fp.addTextValue("tblname", "MyTable");
    Status = fp.addURL("myparameter",  "http://server/main.html");
    Status = fp.addTextValue("id", "1323");
    Status = fp.addTextValue("ftcmd", "addrow");
    fp.setCookies(vlCookies);
    Status = fp.post();
    int nLastErrorCode = fp.getLastError();
    String sLastError = fp.getLastErrorStr();
// Fetch the response
    byte[] byResponse = fp.getRawResponse();
    fp.login(false, ftStatus, null, null);
```

post

Sends a constructed MIME message via an HTTP POST operation.

Syntax

```
public boolean
post()
```

Returns

A boolean value.

Example

```
FormPoster fp = new FormPoster();
FTValList vlCookies;
fp.setURL("http://myserver/servlet/CatalogManager");

// Add a new row to MyTable
FTStatusCode ftStatus = new FTStatusCode();
status = fp.login(true, ftStatus, "myusername", "mypassword");
vlCookies = fp.findAllCookies();
Status = fp.addFile("url", "test.html",
                    "/export/home/temp/test.html", "text/html");
Status = fp.addTextValue("tblname", "MyTable");
Status = fp.addURL("myparameter", "http://server/main.html");
Status = fp.addTextValue("id", "1323");
Status = fp.addTextValue("ftcmd", "addrow");
fp.setCookies(vlCookies);
Status = fp.post();
int nLastErrorCode = fp.getLastError();
String sLastError = fp.getLastErrorStr();

// Fetch the response
byte[] byResponse = fp.getRawResponse();
fp.login(false, ftStatus, null, null);
```

See Also

addTextValue, addURL, addFile, addFileData

postProcessMIMEHeader

Post-processes a MIME header after the postconnector reads it.

Syntax

```
public FTValList
postProcessMIMEHeader(String sHeader)
```

Parameters

`sHeader`
The MIME header.

Returns

An `FTValList` containing the processed MIME header.

registerMIMENotifier

Registers an object that supports the `IMIMENotifier` interface so that it will be called with MIME header information.

This method is only applicable when the output from a posting operation is written to a disk or to the output stream.

Syntax

```
public void
registerMIMENotifier(COM.FutureTense.Interfaces.IMIMENotifier mn)
```

Parameters

`mn`

The `IMIMENotifier` object.

reset

Resets for a new posting operation.

Syntax

```
public void
reset()
```

Example

```
FormPoster fp = new FormPoster();
FTValList vlCookies;
fp.setURL("http://myserver/servlet/CatalogManager");

// Add a new row to MyTable
FTStatusCode ftStatus = new FTStatusCode();
status = fp.login(true, ftStatus, "myusername", "mypassword");
vlCookies = fp.findAllCookies();
Status = fp.addFile("url", "test.html",
                    "/export/home/temp/test.html", "text/html");
Status = fp.addTextValue("tblname", "MyTable");
Status = fp.addURL("myparameter", "http://server/main.html");
Status = fp.addTextValue("id", "1323");
Status = fp.addTextValue("ftcmd", "addrow");
fp.setCookies(vlCookies);
Status = fp.post();
int nLastErrorCode = fp.getLastError();
String sLastError = fp.getLastErrorStr();

// Fetch the response
byte[] byResponse = fp.getRawResponse();
fp.reset();
Status = fp.addFile("url", "body.html",
                    "/export/home/temp/body.html", "text/html");
Status = fp.addTextValue("tblname", "MyTable");
Status = fp.addURL("myparameter", "http://server/index.html");
Status = fp.addTextValue("id", "4567");
Status = fp.addTextValue("ftcmd", "addrow");
Status = fp.post();
fp.login(false, ftStatus, null, null);
```

setAction

Sets the action for this request.

Syntax

```
public void
setAction(String sAction)
   throws Exception
```

Parameters

`sAction`

The action to issue on the `post()`call. Valid values are `FormPoster.Get` or `FormPoster.Post.`

Description

The `setAction` method sets the action for this request. Valid values are `FormPoster.Get` or `FormPoster.Post.`

Throws

Exception if an invalid action is specified.

Example

Use the formposter to fetch a HTML page:

```
FormPoster fp = new FormPoster();
fp.setURL("http://myserver/home.html");
fp.setAction(FormPoster.Get);
status = fp.post();
String sHTMLPage = fp.getResponse();
```

See Also

post

setAuthenticationParameters

Sets the certificate parameters used to authenticate the server.

Syntax

```
public void
setAutenticationParameters(String sEntity,
                           String sIssuer)
```

Parameters

`sEntity`
Name of the entity.

`sIssuer`
Name of the certificate issuer.

Description

The `setAuthenticationParameters` method sets the certificate parameters used to authenticate the server. These parameters will not be used if you are not using a secure protocol (https).

Example

```
    FormPoster fp = new FormPoster();
    FTValList vlCookies;
    fp.setURL("https://myserver/servlet/CatalogManager");

// Allow to connect only if the server can be authenticated
    fp.setAuthenticationParameters("myserver", "Verisign");

// Add a new row to MyTable
    FTStatusCode ftStatus = new FTStatusCode();
    status = fp.login(true, ftStatus, "myusername", "mypassword");
    vlCookies = fp.findAllCookies();
    Status = fp.addFile("url", "test.html",
              "/export/home/temp/test.html", "text/html");
    Status = fp.addTextValue("tblname", "MyTable");
    Status = fp.addURL("myparameter", "http://server/main.html");
    Status = fp.addTextValue("id", "1323");
    Status = fp.addTextValue("ftcmd", "addrow");
    fp.setCookies(vlCookies);
    Status = fp.post();
    int nLastErrorCode = fp.getLastError();
    String sLastError = fp.getLastErrorStr();

// Fetch the response
    byte[] byResponse = fp.getRawResponse();
    fp.login(false, ftStatus, null, null);
```

setBodyData

Passes a pre-built MIME body for posting.

Syntax

```
public void
setBodyData(byte[] data)
```

Parameters

`data`
: A byte array containing the data.

setBodyStream

Passes a pre-built MIME body for posting.

Syntax

```
public void
setBodyStream(Object is,
              int length)
```

Parameters

`is`
The MIME body.

`length`
The length of the MIME body.

setCharacterEncoding

Sets the character coding of string values in subsequent calls to `FormPoster.addTextValue`.

Syntax

```
public void
setCharacterEncoding(String encoding)
```

Parameters

`encoding`

Specify the character encoding. Typical encoding values include `"UTF-8"` or `"Latin-1"`.

Description

The `setCharacterEncoding` method sets the character encoding when subsequent calls to `addTextValue` are made.

The character encoding set by `setCharacterEncoding` only affects data passed through POST; that is, `setCharacterEncoding` does not affect any data passed through GET.

See Also

`addTextValue`

setCookies

The `setCookies` method sets a cookie.

This method has two variants:

- `setCookies (Variant 1)` accepts an `FTValList`.
- `setCookies (Variant 2)` accepts an array.

---

setCookies (Variant 1)

Sets a cookie.

Syntax

```
public void
setCookies(FTValList vlCookies)
```

Parameters

`vlCookies`
: `FTValList` containing cookie name/value pairs.

Example

```
    FormPoster fp = new FormPoster();
    FTValList vlCookies;
    fp.setURL("http://myserver/servlet/CatalogManager");

    // Add a new row to MyTable
    FTStatusCode ftStatus = new FTStatusCode();
    status = fp.login(true, ftStatus, "myusername", "mypassword");
    vlCookies = fp.findAllCookies();
    Status = fp.addFile("url", "test.html",
             "/export/home/temp/test.html", "text/html");
    Status = fp.addTextValue("tblname", "MyTable");
    Status = fp.addURL("myparameter",
                          "http://server/main.html");
    Status = fp.addTextValue("id", "1323");
    Status = fp.addTextValue("ftcmd", "addrow");
    fp.setCookies(vlCookies);
    Status = fp.post();
    int nLastErrorCode = fp.getLastError();
    String sLastError = fp.getLastErrorStr();

// Fetch the response
    byte[] byResponse = fp.getRawResponse();
    fp.reset();
    Status = fp.addFile("url", "body.html",
            "/export/home/temp/body.html", "text/html");
    Status = fp.addTextValue("tblname", "MyTable");
    Status = fp.addURL("myparameter", "http://server/index.html");
    Status = fp.addTextValue("id", "4567");
```

```
Status = fp.addTextValue("ftcmd", "addrow");
Status = fp.post();
fp.login(false, ftStatus, null, null);
```

See Also

findAllCookies

---

setCookies (Variant 2)

Sets a cookie.

Syntax

```
public void
setCookies(javax.servlet.http.Cookie[] c)
```

Parameters

`c`

An array of cookie data. Use `null` to clear.

setDump

Controls status logging information.

Syntax

```
public void
setDump(boolean dumpOutput)
```

Parameters

`dumpOutput`
A value of `true` dumps output messages. The default is `false`.

Example

```
FormPoster fp = new FormPoster();

// Turn on logging
fp.setDump(true);
FTValList vlCookies;
fp.setURL("http://myserver/servlet/CatalogManager");

// Add a new row to MyTable
FTStatusCode ftStatus = new FTStatusCode();
status = fp.login(true, ftStatus, "myusername", "mypassword");
vlCookies = fp.findAllCookies();
Status = fp.addFile("url", "test.html",
    "/export/home/temp/test.html", "text/html");
Status = fp.addTextValue("tblname", "MyTable");
Status = fp.addURL("myparameter", "http://server/main.html");
Status = fp.addTextValue("id", "1323");
Status = fp.addTextValue("ftcmd", "addrow");
fp.setCookies(vlCookies);
Status = fp.post();
int nLastErrorCode = fp.getLastError();
String sLastError = fp.getLastErrorStr();

// Fetch the response
byte[] byResponse = fp.getRawResponse();
fp.reset();
Status = fp.addFile("url", "body.html",
    "/export/home/temp/body.html", "text/html");
Status = fp.addTextValue("tblname", "MyTable");
Status = fp.addURL("myparameter", "http://server/index.html");
Status = fp.addTextValue("id", "4567");
Status = fp.addTextValue("ftcmd", "addrow");
Status = fp.post();
fp.login(false, ftStatus, null, null);
```

See Also

`setLog`

setEncodeURL

Sets the target URL to which the information will ultimately be transmitted via the HTTP GET mechanism.

Syntax

```
public boolean
setEncodeURL(String URL)
```

Parameters

`URL`

The URL to which the message is to be transmitted; the URL should include the payload.

Description

Call `setEncodeURL` to establish the taget URL to which information will ultimately be transmitted when the `post` method executes. If you use `setEncodeURL` to establish the URL, the `post` method will transmit the information via the HTTP GET method. To transmit a message via the HTTP POST method, establish the target URL with `setURL` instead.

Because the HTTP GET method will ultimately be used, you must pass name/value pairs within `URL`. The `setEncodeURL` method ensures that the values are appropriately encoded. For example, if a space appears in the value portion of `URL`, `setEncodeURL` will translate it to `%20`.

Returns

Returns `true` if `URL` has valid URL format; otherwise, returns `false`.

Example

The following registers a URL (`target.FatWire.com`) and two two name/value pairs:

```
boolean result;
result = FP.setEncodeURL("http://target.FatWire.com?a=12&b=7");
```

See Also

`post`, `setEncodeURL`, `setURL`

setHeaderInformation

Sets the HTTP header. This method supports user-defined header records.

Syntax

```
public void
setHeaderInformation(FTValList vlHeaderInfo)
```

Parameters

`vlHeaderInfo`
An `FTValList` containing the header information.

setHTTPVersion

Sets the HTTP protocol version to use.

Syntax

```
public void
setHTTPVersion(int nVersion)
   throws java.lang.Exception
```

Parameters

`nVersion`

Value for the HTTP protocol. Valid values are as follows:

- `FormPoster.HTTPVERSION10`
- `FormPoster.HTTPVERSION11`

Description

The `setHTTPVersion` method sets the HTTP protocol version. By default the FormPoster object transmits data using HTTP version 1.0.

Throws

Exception if the HTTP version is invalid.

Example

```
    FormPoster fp = new FormPoster();

// Use HTTP 1.1
    fp.setHTTPVersion(FormPoster.HTTPVERSION11);
    FTValList vlCookies;
    fp.setURL("http://myserver/servlet/CatalogManager");

// Add a new row to MyTable
    FTStatusCode ftStatus = new FTStatusCode();
    status = fp.login(true, ftStatus, "myusername", "mypassword");
    vlCookies = fp.findAllCookies();
    Status = fp.addFile("url", "test.html",
                    "/export/home/temp/test.html", "text/html");
    Status = fp.addTextValue("tblname", "MyTable");
    Status = fp.addURL("myparameter", "http://server/main.html");
    Status = fp.addTextValue("id", "1323");
    Status = fp.addTextValue("ftcmd", "addrow");
    fp.setCookies(vlCookies);
    Status = fp.post();
    int nLastErrorCode = fp.getLastError();
    String sLastError = fp.getLastErrorStr();

// Fetch the response
    byte[] byResponse = fp.getRawResponse();
    fp.login(false, ftStatus, null, null);
```

setLog

The `setLog` method supplies a logging object for special log types. Use of this method is not required. If you do not use setLog, output streams to a default location based on the environment in which you invoke the method.

This method has two variants:

- `setLog (Variant 1)` accepts a logfile object.
- `setLog (Variant 2)` accepts a string.

---

setLog (Variant 1)

Supplies a logging object for special log types.

Syntax

```
public void
setLog(LogFile logFile)
```

Parameters

`logFile`
An object which can construct a logging output.

Description

The `setLog` method supplies a logging object for special log types. Use of this method is not required. If you do not use `setLog`, output streams to a default location based on the environment in which you invoke the method.

Example

```
FormPoster fp = new FormPoster();

// Turn on logging
fp.setDump(true);
fp.setLog("/export/home/mydir/");
FTValList vlCookies;
fp.setURL("http://myserver/servlet/CatalogManager");

// Add a new row to MyTable
FTStatusCode ftStatus = new FTStatusCode();
status = fp.login(true, ftStatus, "myusername", "mypassword");
vlCookies = fp.findAllCookies();
Status = fp.addFile("url", "test.html",
                    "/export/home/temp/test.html", "text/html");
Status = fp.addTextValue("tblname", "MyTable");
Status = fp.addURL("myparameter", "http://server/main.html");
Status = fp.addTextValue("id", "1323");
Status = fp.addTextValue("ftcmd", "addrow");
fp.setCookies(vlCookies);
Status = fp.post();
int nLastErrorCode = fp.getLastError();
```

```
String sLastError = fp.getLastErrorStr();

// Fetch the response
byte[] byResponse = fp.getRawResponse();
fp.reset();
Status = fp.addFile("url", "body.html",
                    "/export/home/temp/body.html", "text/html");
Status = fp.addTextValue("tblname", "MyTable");
Status = fp.addURL("myparameter", "http://server/index.html");
Status = fp.addTextValue("id", "4567");
Status = fp.addTextValue("ftcmd", "addrow");
Status = fp.post();
fp.login(false, ftStatus, null, null);
```

See Also

setDump

---

setLog (Variant 2)

Supplies a logging object for special log types.

Syntax

```
public void
setLog(String logdir)
```

Parameters

`logdir`
Path to log directory and '`wwwresponse.log`' will be appended to create a logfile.

Description

The `setLog` method supplies a logging object for special log types. Use of this method is not required. If you do not use `setLog`, output streams to a default location based on the environment in which you invoke the method.

Example

```
FormPoster fp = new FormPoster();

// Turn on logging
fp.setDump(true);
fp.setLog("/export/home/mydir/");
FTValList vlCookies;
fp.setURL("http://myserver/servlet/CatalogManager");

// Add a new row to MyTable
FTStatusCode ftStatus = new FTStatusCode();
status = fp.login(true, ftStatus, "myusername", "mypassword");
vlCookies = fp.findAllCookies();
Status = fp.addFile("url", "test.html",
```

```
    "/export/home/temp/test.html", "text/html");
Status = fp.addTextValue("tblname", "MyTable");
Status = fp.addURL("myparameter", "http://server/main.html");
Status = fp.addTextValue("id", "1323");
Status = fp.addTextValue("ftcmd", "addrow");
fp.setCookies(vlCookies);
Status = fp.post();
int nLastErrorCode = fp.getLastError();
String sLastError = fp.getLastErrorStr();

// Fetch the response
byte[] byResponse = fp.getRawResponse();
fp.reset();
Status = fp.addFile("url", "body.html",
                    "/export/home/temp/body.html", "text/html");
Status = fp.addTextValue("tblname", "MyTable");
Status = fp.addURL("myparameter", "http://server/index.html");
Status = fp.addTextValue("id", "4567");
Status = fp.addTextValue("ftcmd", "addrow");
Status = fp.post();
fp.login(false, ftStatus, null, null);
```

See Also

setDump

setOutputDirectory

Sets the directory where the post results output file should be written.

Syntax

```
public void
setOutputDirectory(sDir outputdir)
```

Parameters:

`outputdir`
The directory path. If null, no output file is written.

Description

The `setOutputDirectory` method sets the directory where the post results output file should be written. `getOutputFileSpec` must be called after the post to determine the name of the file where the output has been written.

Example

```
FormPoster fp = new FormPoster();
FTValList vlCookies;
fp.setURL("http://myserver/servlet/CatalogManager");

// Add a new row to MyTable
FTStatusCode ftStatus = new FTStatusCode();
status = fp.login(true, ftStatus, "myusername", "mypassword");
vlCookies = fp.findAllCookies();
Status = fp.addFile("url", "test.html",
"/export/home/temp/test.html", "text/html");
Status = fp.addTextValue("tblname", "MyTable");
Status = fp.addURL("myparameter", "http://server/main.html");
Status = fp.addTextValue("id", "1323");
Status = fp.addTextValue("ftcmd", "addrow");
fp.setCookies(vlCookies);

// Write the post results to a file
fp.setOutputDirectory("/export/home/mydirectory");
Status = fp.post();
int nLastErrorCode = fp.getLastError();
String sLastError = fp.getLastErrorStr();

// Fetch the output file name
String sOutputFile = fp.getOutputFileSpec();
Fp.login(false, null, null);
```

See Also

`getOutputFileSpec`

setOutputStream

Sets an output stream to be used for capturing the response. This method is mutually exclusive with `setOutputDirectory`.

Syntax

```
public void
setOutputStream(OutputStream os)
```

Parameters

`os`

The output stream (not closed).

setPort

Sets the port to be used to connect to the remote HTTP server.

Syntax

```
public void
setPort(int nPort)
```

Parameters

`nPort`

If set after the URL is set, this parameter will override the default port that was determined from the URL protocol.

Example

```
FormPoster fp = new FormPoster();
FTValList vlCookies;
fp.setURL("http://myserver/servlet/CatalogManager");

// Use port 500 for communication
fp.setPort(500);

// Add a new row to MyTable
FTStatusCode ftStatus = new FTStatusCode();
status = fp.login(true, ftStatus, "myusername", "mypassword");
vlCookies = fp.findAllCookies();
Status = fp.addFile("url", "test.html",
             "/export/home/temp/test.html", "text/html");
Status = fp.addTextValue("tblname", "MyTable");
Status = fp.addURL("myparameter", "http://server/main.html");
Status = fp.addTextValue("id", "1323");
Status = fp.addTextValue("ftcmd", "addrow");
fp.setCookies(vlCookies);
Status = fp.post();
int nLastErrorCode = fp.getLastError();
String sLastError = fp.getLastErrorStr();

// Fetch the response
byte[] byResponse = fp.getRawResponse();
fp.reset();
Status = fp.addFile("url", "body.html",
               "/export/home/temp/body.html", "text/html");
Status = fp.addTextValue("tblname", "MyTable");
Status = fp.addURL("myparameter", "http://server/index.html");
Status = fp.addTextValue("id", "4567");
Status = fp.addTextValue("ftcmd", "addrow");
Status = fp.post();
fp.login(false, ftStatus, null, null);
```

See Also

`setURL`

setProxy

Sets the proxy host when a proxy server is needed to post data through a firewall.

Syntax

```
public void
setProxy(String sHost)
```

Parameters

`sHost`
Host name of the proxy server.

Example

```
FormPoster fp = new FormPoster();
FTValList vlCookies;
fp.setURL("http://myserver/servlet/CatalogManager");

// Use a proxy server
fp.setProxy("myproxy");

// Add a new row to MyTable
FTStatusCode ftStatus = new FTStatusCode();
status = fp.login(true, ftStatus, "myusername", "mypassword");
vlCookies = fp.findAllCookies();
Status = fp.addFile("url", "test.html",
"/export/home/temp/test.html", "text/html");
Status = fp.addTextValue("tblname", "MyTable");
Status = fp.addURL("myparameter", "http://server/main.html");
Status = fp.addTextValue("id", "1323");
Status = fp.addTextValue("ftcmd", "addrow");
fp.setCookies(vlCookies);
Status = fp.post();
int nLastErrorCode = fp.getLastError();
String sLastError = fp.getLastErrorStr();

// Fetch the response
byte[] byResponse = fp.getRawResponse();
fp.reset();
Status = fp.addFile("url", "body.html", "/export/home/temp/
body.html", "text/html");
Status = fp.addTextValue("tblname", "MyTable");
Status = fp.addURL("myparameter", "http://server/index.html");
Status = fp.addTextValue("id", "4567");
Status = fp.addTextValue("ftcmd", "addrow");
Status = fp.post();
fp.login(false, ftStatus, null, null);
```

See Also

`setProxyPort()`

setProxyPort

Specifies a proxy server port when a proxy server is needed to post data through a firewall.

Syntax

```
public void
setProxyPort(int nPort)
```

Parameters

`nPort`

Port number of the proxy server.

Example

```
FormPoster fp = new FormPoster();
FTValList vlCookies;
fp.setURL("http://myserver/servlet/CatalogManager");

// Use the proxy server listening on port 200
fp.setProxy("myproxy");
fp.setProxyPort(200);

// Add a new row to MyTable
FTStatusCode ftStatus = new FTStatusCode();
status = fp.login(true, ftStatus, "myusername", "mypassword");
vlCookies = fp.findAllCookies();
Status = fp.addFile("url", "test.html",
                "/export/home/temp/test.html", "text/html");
Status = fp.addTextValue("tblname", "MyTable");
Status = fp.addURL("myparameter", "http://server/main.html");
Status = fp.addTextValue("id", "1323");
Status = fp.addTextValue("ftcmd", "addrow");
fp.setCookies(vlCookies);
Status = fp.post();
int nLastErrorCode = fp.getLastError();
String sLastError = fp.getLastErrorStr();

// Fetch the response
byte[] byResponse = fp.getRawResponse();
fp.reset();
Status = fp.addFile("url", "body.html",
                "/export/home/temp/body.html", "text/html");
Status = fp.addTextValue("tblname", "MyTable");
Status = fp.addURL("myparameter", "http://server/index.html");
Status = fp.addTextValue("id", "4567");
Status = fp.addTextValue("ftcmd", "addrow");
Status = fp.post();
fp.login(false, ftStatus, null, null);
```

See Also

setProxy

setSecureProtocol

Allows you to communicate in secure (HTTPS) mode.

Syntax

```
public void
setSecureProtocol(boolean bSecure)
```

Parameters

`bSecure`

If set after the URL is set, this parameter will override the default which was determined from the URL protocol.

Example

```
FormPoster fp = new FormPoster();
FTValList vlCookies;
fp.setURL("http://myserver/servlet/CatalogManager");

// Use https for comminication
fp.setSecureProtocol(true);

// Add a new row to MyTable
FTStatusCode ftStatus = new FTStatusCode();
status = fp.login(true, ftStatus, "myusername", "mypassword");
vlCookies = fp.findAllCookies();
Status = fp.addFile("url", "test.html",
 "/export/home/temp/test.html", "text/html");
Status = fp.addTextValue("tblname", "MyTable");
Status = fp.addURL("myparameter", "http://server/main.html");
Status = fp.addTextValue("id", "1323");
Status = fp.addTextValue("ftcmd", "addrow");
fp.setCookies(vlCookies);
Status = fp.post();
int nLastErrorCode = fp.getLastError();
String sLastError = fp.getLastErrorStr();

// Fetch the response
byte[] byResponse = fp.getRawResponse();
fp.reset();
Status = fp.addFile("url", "body.html",
"/export/home/temp/body.html", "text/html");
Status = fp.addTextValue("tblname", "MyTable");
Status = fp.addURL("myparameter", "http://server/index.html");
Status = fp.addTextValue("id", "4567");
Status = fp.addTextValue("ftcmd", "addrow");
Status = fp.post();
fp.login(false, ftStatus, null, null);
```

See Also

setURL

setTimeout

Sets a network timeout.

Syntax

```
public void
setTimeout(int nTimeout)
```

Parameters

`nTimeout`
: Timeout in milliseconds.

Description

The `setTimeout` method sets a network timeout. Specify the value in milliseconds. 0 means no timeout.

Example

```
FormPoster fp = new FormPoster();
FTValList vlCookies;
fp.setURL("http://myserver/servlet/CatalogManager");

// Add a new row to MyTable
FTStatusCode ftStatus = new FTStatusCode();
status = fp.login(true, ftStatus, "myusername", "mypassword");
vlCookies = fp.findAllCookies();
Status = fp.addFile("url", "test.html",
                 "/export/home/temp/test.html", "text/html");
Status = fp.addTextValue("tblname", "MyTable");
Status = fp.addURL("myparameter", "http://server/main.html");
Status = fp.addTextValue("id", "1323");
Status = fp.addTextValue("ftcmd", "addrow");
fp.setCookies(vlCookies);

// Don’t wait forever for a response
fp.setTimeout(200);
Status = fp.post();
int nLastErrorCode = fp.getLastError();
String sLastError = fp.getLastErrorStr();

// Fetch the response
byte[] byResponse = fp.getRawResponse();
fp.reset();
Status = fp.addFile("url", "body.html",
                   "/export/home/temp/body.html", "text/html");
Status = fp.addTextValue("tblname", "MyTable");
Status = fp.addURL("myparameter", "http://server/index.html");
Status = fp.addTextValue("id", "4567");
Status = fp.addTextValue("ftcmd", "addrow");
Status = fp.post();
fp.login(false, ftStatus, null, null);
```

setURL

Sets the target URL to which the information will ultimately be transmitted via the HTTP POST mechanism.

Syntax

```
public boolean
setURL(String URL)
```

Parameters

`URL`

The URL to which the message is to be transmitted.

Description

Call `setURL` to establish the taget URL to which information will ultimately be transmitted when `post` executes. If you call `setURL` to establish the URL, post will transmit the information via the HTTP POST method. To transmit a message via the HTTP GET method, establish the target URL with `setEncodeURL` instead.

Returns

Returns `true` if `URL` has valid URL format; otherwise, returns `false`.

Example

```
FormPoster fp = new FormPoster();
FTValList vlCookies;
fp.setURL("http://myserver/servlet/CatalogManager");

// Add a new row to MyTable
FTStatusCode ftStatus = new FTStatusCode();
status = fp.login(true, ftStatus, "myusername", "mypassword");
Status = fp.addFile("url", "test.html",
                    "/export/home/temp/test.html", "text/html");
Status = fp.addTextValue("tblname", "MyTable");
Status = fp.addURL("myparameter", "http://server/main.html");
Status = fp.addTextValue("id", "1323\");
Status = fp.addTextValue("ftcmd", "addrow");
Status = fp.post();
int nLastErrorCode = fp.getLastError();
String sLastError = fp.getLastErrorStr();

// Fetch the response
byte[] byResponse = fp.getRawResponse();
fp.reset();
Status = fp.addFile("url", "body.html",
"/export/home/temp/body.html", "text/html");
Status = fp.addTextValue("tblname", "MyTable");
Status = fp.addURL("myparameter", "http://server/index.html");
Status = fp.addTextValue("id", "4567");
Status = fp.addTextValue("ftcmd", "addrow");
Status = fp.post();
fp.login(false, ftStatus, null, null);
```

See Also

setEncodeURL, setSecureProtocol, setPort

Chapter 4

FTVAL

FTVAL extends Object. FTVAL is not an interface, but is a class used to abstract various data types to allow them to be passed between components of the Content Server environment as items in an FTValList.

An FTVAL object can only be created indirectly. The usual procedure is to use the various methods in FTValList to add elements to the list. These methods will internally create the appropriate FTVAL objects and add them to the FTValList. There are also a number of methods that build and return FTValList objects, and some methods that return FTVAL objects.

The FTValList.getVal method is the usual way to get an FTVAL object itself.

List of Methods

Table 2: List of FTVAL Methods

Method	Summary
getBlob	Gets the value of a blob.
getFile	Gets a file name that holds the data for this FTVAL.
GetInt	Returns the integer value associated with this FTVAL object.
GetObj	Returns the data from the specified object.
getStream	Returns an input stream.
getString	Returns the string associated with this object.
isEmpty	Determines whether the data in this FTVAL object is empty.
length	Returns the length of the value associated with this FTVAL object.
toString	Returns a string representation of an FTVAL object.

getBlob

Gets the value of a blob.

Syntax

```
public final byte[]
getBlob()
```

Returns

If the data held by this FTVAL is a blob (binary large object), `getBlob` returns the value of the blob inside the byte array. For example, if the FTVAL holds a PDF, `getBlob` writes the entire contents of the PDF into the return value. If the data held by this FTVAL is not a blob, `getBlob` returns null.

getFile

Gets a file name that holds the data for this FTVAL.

Syntax

```
public String
getFile()
  throws Exception
```

Returns

This method returns the pathname of the file from which you can retrieve the data associated with this FTVAL. This variable might be represented as a file, using a `url` column, or as a binary or character data type in the database.

Throws

`Exception`

Sometimes, `getFile` has to create a temporary file. An exception is thrown when you do not have permission to create this temporary file or there is insufficent disk space to do so.

GetInt

Returns the integer value associated with this FTVAL object.

Syntax

```
public int
GetInt()
```

Returns

An integer corresponding to the value stored in the FTVAL object. If the FTVAL type is not I4, this method returns 0.

Example

Suppose that MyVal holds an integer value. In this case, the following code fragment returns that integer value into x:

```
int x = MyVal.GetInt();
```

See Also

GetObj, GetSize

GetObj

Returns the data from the specified object.

Syntax

```
public Object
GetObj()
```

Returns

An object corresponding to the object stored in this FTVAL. If the FTVAL type is not appropriate for an object value, as is the case where the type is I4, this method returns a null.

See Also

GetInt, GetSize

GetSize

Deprecated *as of version 5.0*.

Returns the integer size.

Syntax

```
public int
GetSize()
```

Returns

An integer representing the size in bytes of the object stored in the FTVAL object. The size for FTVAL type I4 is zero. The size may not always be known to the FTVAL object, in which case this method returns 0.

See Also

GetInt(), GetObj()

getStream

Returns an input stream

Syntax

```
public final InputStream
getStream()
  throws Exception
```

Returns

An input stream.

Description

If the data in the `FTVAL` object is a blob, `getStream` returns either a `ByteArrayInputStream` or a `FileInputStream` to the serialized data.

If the data in the `FTVAL` object is anything other than a blob, `getStream` returns a `ByteArrayInputStream` to the serialized data.

getString

Returns the string associated with this object.

Syntax

```
public String
GetString()
```

Returns

If this value holds binary data, `getString` returns `null`. If this object holds nonbinary data, `getString` returns the string representation of the data.

See Also

GetInt, GetSize

GetType (Deprecated)

Deprecated *as of version 5.0*.

Returns the integer type.

Syntax

```
public int
GetType()
```

Description

The `GetType` method returns the integer type of the specified integer.

An `FTVAL` has a default type of `UNKNOWN`. Strings would generally have the type `LPSTR`, which is guaranteed if the `FTVAL` is created by the `FTValList` method `setValString`().

Any integers encapsulated in an `FTVAL` would have the type I4, which is guaranteed if it was created by the `FTValList` method `setValInt()`.

`BLOB` would be used for binary data.

Returns

An integer whose value represents the type. Valid type values are listed above.

See Also

`GetInt, GetObj, GetSize`

isEmpty

Determines whether the data in this FTVAL object is empty.

Syntax

```
public final boolean
isEmpty()
```

Returns

Returns `true` if the data in this FTVAL object is empty. Returns `false` if this object has some associated data. If the data type is numeric, this method always returns `false`.

Description

Call `isEmpty` to determine whether the data in this FTVAL object is empty. This method is particularly useful to determine if an upload field is empty.

length

Returns the length of the value associated with this FTVAL object.

Syntax

```
public final long
length()
```

Returns

The length of the value associated with this FTVAL object.

Description

If the value held by this FTVAL object is a blob, length returns the number of bytes in the blob. For example, if the blob is a 60,000 byte JPEG file, then length returns 60,000.

If the value held by this FTVAL object is not a blob, `length` internally transforms the value to a `String and` then returns the number of characters in the String. For example, suppose the FTVAL object holds the LONG value 713. Since “713” consumes three character positions, `length` returns 3.

toString

The `toString` method returns a string representation of an `FTVAL` object. This method overrides the `toString` method in the `java.lang.Object` class.

This method has two variants:

- `toString (Variant 1)` returns a string representation of an `FTVAL` object.
- `toString (Variant 2)` returns a string representation of an `FTVAL` object and allows you to specify the encoding for the conversion.

---

toString (Variant 1)

Returns a string representation of an `FTVAL` object. This method overrides the `toString` method in the `java.lang.Object` class.

Syntax

```
public String
toString()
```

Returns

Returns a string representation of an `FTVAL` object if appropriate; otherwise returns `null`.

---

toString (Variant 2)

Returns a string representation of an `FTVAL` object. This method overrides the `toString` method in the `java.lang.Object` class.

Syntax

```
public String
toString(String encoding)
```

Parameters

`encoding`
The encoding to be used in the conversion.

Returns

Returns a string representation of an `FTVAL` object if appropriate; otherwise returns `null`.

Chapter 5

FTValList

An FTValList is a list of name/value pairs. The name is the key, and it must be unique in the list. The value is represented using the FTVAL class.

Use FTValList to pass arguments to Content Server subsystems like the CatalogManager and TreeManager, to pass named values into Seed classes being called using the CALLJAVA tag, and to return a set of variables to the element containing the CALLJAVA tag. The ARGUMENT tag is used in the CALLJAVA tag to add name/value pairs to the FTValList; those name/value pairs are then passed to the Execute method of the class being called.

FTValList Constructors

An FTValList will grow as necessary to accommodate new elements. However, it is most efficient if it starts with the capacity to hold a small number of elements, rather than growing from a size of zero. Use the following constructor to specify an initial capacity:

```
FTValList(int x);
```

where x is the number of name/value pairs that the list can hold.

If you use the default constructor FTValList(), the default size is 32 name/value pairs.

List of Methods

`FTValList` supports the following nondeprecated methods:

Table 3: List of FTValList Methods

Method	Summary
`alphaSortedKeys`	Returns an enumeration of keys, sorted alphabetically.
`copy`	Creates a copy of this list.
`count`	Returns a count of the number of name/value pairs in this list.
`equals`	Determines if two lists are equal.
`get`	Returns value, regardless of data type.
`getVal`	Gets the value associated with the specified key.
`getValBLOB`	Gets the blob of type `byte[]` that corresponds to the specified key name.
`getValBLOBSize`	Gets the size of the blob (Binary Large Object) that is associated with the specified key.
`getValInt`	Gets the integer value associated with the specified key.
`getValString`	Gets the string value that is associated with the specified key.
`keys`	Returns an enumeration of key names.
`keysVector`	Returns a vector of keys.
`parse`	Converts a string of form "`a=b&c=d&...`" to an `FTValList`.
`put`	Associates a value with a specified key.
`remove`	Removes one item from this list.
`removeAll`	Removes all items from this list.
`setVal`	Adds an `FTVal` object to this list.
`setValBLOB`	Adds a blob to this list, using the associated key name.
`setValBLOBFile`	Adds a blob file to this list, using the associated key name.
`setValInt`	Adds an integer value to this list.
`setValString`	Adds a String value to this list.
`sortedKeys`	Returns an enumeration of key names, sorted longest to shortest.

alphaSortedKeys

Returns an enumeration of keys, sorted alphanumerically. This method is thread safe.

Syntax

```
public Enumeration
alphaSortedKeys()
```

Returns

Returns an enumeration of keys, sorted alphanumerically.

copy

Creates a new `FTValList` that is a copy of this list.

Syntax

```
public FTValList
copy()
```

Description

The `Copy` method creates a new `FTValList` that is a copy of the specified list. The copy is shallow: the items themselves remain for use by the original `FTValList`.

Returns

New `FTValList`.

count

Gets the number of items (name/value pairs) in this `FTValList`.

Syntax

```
public int
count()
```

Returns

Number of elements in the `FTValList`.

Example

This example creates a `FTValList` and counts the number of name/value pairs:

```
cs.ClearErrno();
FTValList ft = Utilities.getParams("a=b&c=d");
if (cs.GetErrno() == 0)
{
     int n = ft.count();
}
```

equals

Overrides the `equals` method in the `java.util.Hashtable` class.

Syntax

```
public boolean
equals(Object o)
```

Parameters

o
: Specify an object.

Returns

Returns `true` if `o` is equivalent to this list; otherwise, returns `false`.

get

Returns the value associated with the specified key, regardless of type. Overrides the `get` method in the `java.util.Hashtable` class.

Syntax

```
public Object
get(Object key)
```

Parameters

`key`

String containing the name of the key.

Returns

Returns the value of the specified key.

getNextKey (Deprecated)

Deprecated. *As of version 4.0, replaced by* `keys()`.

Syntax

```
public String
getNextKey()
```

Returns

Returns the key name of the next key.

getVal

Gets an FTVAL (value) that corresponds to the specified key name.

Syntax

```
public FTVAL
getVal(String key)
```

Parameters

`key`
String containing the name of the key.

Description

The `getVal` method returns the FTVAL corresponding to `key`. Use the methods of the FTVAL class to evaluate the returned FTVAL.

Returns

The FTVAL object associated with this key. It returns null if the key is not found in the `FTValList`.

getValBLOB

Gets the blob (Binary Large Object), of type `byte[]`, that corresponds to the specified key name.

Syntax

```
public byte[]
getValBLOB(String key)
```

Parameters

`key`
: String containing the name of the key.

Returns

Value of type `byte[]`; null if the key is not found in the `FTValList` or the `FTVAL` type is not `FTVAL.BLOB`.

getValBLOBSize

Gets the size of the blob (Binary Large Object) that is associated with the specified key.

Syntax

```
public int
getValBLOBSize(String key)
```

Parameters

`key`

String containing the name of a key.

Returns

The size of the blob; in other words, the number of bytes in the binary data. Returns zero (0) if the key is not found or if the key is not associated with a blob.

getValInt

Gets the integer value associated with the specified key.

Syntax

```
public int
getValInt(String key)
```

Parameters

`key`
String containing the name of the key.

Returns

Value if found. Zero (0) if not found.

getValString

Gets the string value that is associated with the specified key.

Syntax

```
public String
getValString(String key)
```

Parameters

`key`
The name of the key.

Returns

The `String` value associated with the key. If the associated `FTVAL` is of type `FTVAL.I4`, the integer is converted to a string and returned. If the key is not found, or the `FTVAL` is not a string or an integer, then null is returned.

Example

The following code constructs a tiny list and then calls `getValString` to retrieve the value associated with the sole key:

```
String aKey = "DMajor";
String aValue = "Sonata for Two UNIX Boxes";
FTValList MyTinyList = new FTValList(8);
MyTinyList.setValString(aKey, aValue);

// Now retrieve the value associated with DMajor
String RetrievedValue = MyTinyList.getValString(aKey);
```

After running this code, `RetrievedValue` should hold:

```
Sonata for Two UNIX Boxes
```

keys

Returns an enumeration of key names.

Syntax

```
public Enumeration
keys()
```

Description

The keys method returns an enumeration of key names. This method is thread safe.

Returns

Enumeration of key names.

Example

The following code generates a comma-separated list of the keys in `vIn`:

```
String bf = new StringBuffer();
Enumeration theKeys = vIn.keys();
int i = 0;
while ( theKeys.hasMoreElements() )
{
     String name = theKeys.nextElement();
     if ( i > 0 )
          bf.append( "," );
     bf.append( name );
     i++;
}
```

keysVector

Returns a vector of keys. This method is thread safe.

Syntax

```
public Vector
keysVector()
```

Returns

Returns a vector containing all the keys currently in this list.

parse

Converts a string of form "`a=b&c=d&...`" to an `FTValList`.

Syntax

```
public int
parse(String inputParam)
```

Parameters

`inputParam`
: Name/value pairs in the following format: "`a=b&c=d&...`"

Returns

An integer representing the number of total items in the list **after** the parse. So, if the list was empty prior to running the `parse` method, the returned value will equal the number of items parsed.

Description

Adds name/value pairs to this list. The input name/value pairs must have the same format used in the payload portion of a URL. If any of the input names (keys) already exist in the list, then `parse` replaces the current value with the value designated within `inputParam`.

put

Associates a value with a specified key. Overrides the `put` method in the `java.util.Hashtable` class.

Syntax

```
public Object
put(Object key,
    Object value)
```

Parameters

`key`
The key to associate with the value.

`value`
The value to associate with the key.

Returns

Returns previous value associated with the key, or `null` (if there was no previous value).

remove

Removes one item from this list. This method overrides the `remove` method in the `java.util.Hashtable` class.

Syntax

```
public Object
remove(Object key)
```

Parameters

`key`
: They key of the object to remove.

Returns

The previous value, or `null` on error.

removeAll

Removes all items from this list.

Syntax

```
public void
removeAll()
```

resetPosition (Deprecated)

Deprecated *as of version 4.0, use* `keys instead.`

Syntax

```
public int
resetPosition()
```

Returns

Returns 1 if successful.

setVal

The `setVal` method adds an `FTVal` object to an `FTValList`.

This method has two variants:

- `setVal (Variant 1)` adds an `FTVal` object to an `FTValList`.
- `setVal (Variant 2)` creates an `FTVAL` object, using the type and data object, to add to the `FTValList` with the associated key name.

---

setVal (Variant 1)

Adds the `FTVAL` object to the `FTValList` with the associated key name.

Syntax

```
public int
setVal(String key,
       FTVAL val)
```

Parameters

`key`
: Name of the key to add to the FTValList.

`val`
: Value to assign to the value part of the name/value pair.

Returns

Zero (0) on success, and -1 on failure.

Example

This example uses `setVal` to build a filtered `FTValList`:

```
// Copy FTValList to v2, but eliminate elements of type FTVAL.I4
FTValList v2 = new FTValList();
Enumeration keys = v.keys();
while ( keys.hasMoreElements() )
{
     String keystr = (String)keys.nextElement();
     FTVAL fval = v.getVal( keystr );
     if ( fval.GetType() != FTVAL.I4 )
          v2.setVal( keystr, fval );
}
```

---

setVal (Variant 2)

Creates an `FTVAL` object, using the type and data object, to add to the `FTValList` with the associated key name.

Syntax

```
public int
setVal(String key,
       int type,
       Object data)
```

Parameters

`key`
Name of the key to add to the FTValList.

`type`
Type of the value. A list of valid types follows:

```
FTVAL.I4
FTVAL.DOUBLE
FTVAL.UNKNOWN
FTVAL.LPSTR
FTVAL.BLOB
FTVAL.DATE
FTVAL.STRBYTES
FTVAL.BLOBBYTES
FTVAL.NASTHING
```

`data`
Value to assign to the value part of the name/value pair.

Returns

Zero (0) on success, -1 on failure.

setValBLOB

Adds a blob (Binary Large Object) to the `FTValList`, using the associated key name.

Syntax

```
public int
setValBLOB(String key,
           byte[] b,
           int sz)
```

Parameters

`key`
: Name of the key to add to the `FTValList`.

`b`
: Array of bytes containing the binary data.

`sz`
: Size: the number of bytes in the array of data.

Returns

Zero (0) on success, -1 on failure.

setValBLOBFile

Adds a blob (Binary Large Object) file to the `FTValList`, using the associated key name.

Syntax

```
public int
setValBLOBFile(String key,
               String file)
```

Parameters

`key`
: Name of the key to add to the `FTValList`.

`file`
: The name of the file to add.

Returns

Zero (0) on success, -1 on failure.

setValInt

Adds an integer value to an `FTValList`, by using the associated key name.

Syntax

```
public int
setValInt(String key,
          int i)
```

Parameters

`key`
: Name of the key to add to the `FTValList`.

`i`
: Integer value

Returns

Zero (0) on success -1 on failure.

Example

This example adds an integer value to a `FTValList`:

```
FTValList fList = new FTValList();
fList.setValInt( "age" , 3 );
```

setValString

Adds a String value to the `FTValList`, using the associated key name.

Syntax

```
public int
setValString(String key,
             String val)
```

Parameters

`key`
: Name of the key to add to the FTValList.

`val`
: Value to assign using the key.

Returns

Zero (0) on success, -1 on failure.

Example

This example adds a `String` value to a `FTValList`:

```
FTValList fList = new FTValList();
fList.setValString("errormsg" , "File not found.");
```

sortedKeys

Returns an enumeration of key names, sorted longest to shortest. This method is thread safe.

Syntax

```
public Enumeration
sortedKeys()
```

Returns

Returns an enumeration of key names, sorted longest to shortest.

Chapter 6

ICS

The `ICS` (Interface to Content Server) interface provides access to Content Server functionality during the evaluation of a page. All variables, session variables, counters, lists, and resultsets established before using the `ICS` interface are available. Any modifications or additions to variables, counters, lists, and resultsets using the `ICS` interface will be reflected upon return from the `CALLJAVA` XML tag. This section contains a list of methods, followed by a description of each method.

List of Methods

The following lists categorize all the methods of the ICS class.

Conditions

ioErrorThrown	isCacheable	IsElement
isSystemSecure	IsTracked	UserIsMember

Counters

GetCounter	RemoveCounter	SetCounter

Database Operations

CallSQL	CatalogDef	CatalogManager
CommitBatchedCommands	dbDebug	FlushCatalog
GetCatalogType	literal	RestoreProperty

RollbackBatchedCommands | SelectTo | SQL
SQLExp

Debug Flags

dbDebug | eventDebug | pgCacheDebug
rsCacheDebug | sessionDebug | synchDebug
systemDebug | systemSession | timeDebug
xmlDebug

Errors

ClearErrno | getComplexError | GetErrno
setComplexError | SetErrno

Events and Email

AppEvent | DestroyEvent | DisableEvent
EmailEvent | EnableEvent | eventDebug
QueryEvents | SendMail

Execution

CallElement | CallSQL | DeployJSPData
DeployJSPFile | DisableCache | FlushCatalog
GetBin | GetCgi | getIServlet
GetSynchronizedHash | InsertPage | ReadPage

Lists

CopyList | GetList | RegisterList
RenameList

Miscellaneous Tags

clientIsSS | decode | DeleteSynchronizedHash
encode | genID | getLocaleString
GetSynchronizedHash | LogMsg | Mirror

pageCriteriaKeys pageURL runTag
StreamEvalBytes ThrowException TreeManager

Objects

GetObj PushObj SetObj
PopObj

Property Files

GetProperty LoadProperty RestoreProperty

Regular Variables

GetVar GetVars PopVars
PushVars RemoveVar ResolveVariables
SetVar

Revision Tracking

CatalogManager IsTracked RTCommit
RTDeleteRevision RTHistory RTLock
RTRelease RTRetrieveRevision RTRollback
RTSetVersions RTTrackTable RTUnlockRecord
RTUntrackTable

Search Engines

IndexAdd IndexCreate IndexDestroy
IndexExists IndexRemove IndexReplace
Search SearchDateToNative

Sessions and Cookies

GetSSVar GetSSVars RemoveSSVar
sessionDebug SessionExists SessionID
SetCookie SetSSVar

Streams

StreamBinary	StreamEvalBytes	StreamHeader
StreamText	FlushStream	

AppEvent

Creates an event that triggers a registered application logic.

Syntax

```
public boolean
AppEvent(String name,
         String guid,
         String times,
         FTValList params)
```

Parameters

`name`

Name to register the event under. Do not use special characters in event name, such as "{", "*", or "&".

`guid`

The name of the servlet that is called when the event is triggered.

`times`

Time at which the event will be triggered. Use the format:

`hh:mm:ss W/DD/MM`

where:

- `hh` (hour): 0 - 23
- `mm` (minute): 0 - 59
- `ss` (second): 0 - 59
- `W` (day of the week): 0 - 6 with 0 = Sunday.
- `DD` (day of the month): 1 - 31
- `MM` (month): 1 - 12

For each of these fields, it is valid to use an asterisk (all legal values) or a list of elements separated by commas. A value can be one or more numbers separated by a minus sign indicating an inclusive range. For example:

```
2, 5 - 7:0:0 1/*/*
```

means the event is triggered at 2 a.m., 5 a.m., 6 a.m. and 7 a.m. every Monday.

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

`params`

A list of name/value pairs that are passed to the servlet. May be null.

Description

The `AppEvent` method creates an event that triggers a registered application logic.

Returns

Boolean value indicating success or failure.

errno

Use `GetErrno` to view the error.

Possible values of `errno` include:

Value	Description
`-200`	Failed to register event.
`-201`	Exception in event. (May appear as `-200`.)

Example

This example registers an event that will invoke the ContentServer application logic. The event will be triggered every day at 1:10 p.m. and invoke ContentServer to process the pagename `API/Main`. Note that the single parameter to this application logic is passed in the `FTValList`.

```
ics.ClearErrno();
FTValList params = new FTValList()
params.setValString( “pagename” , “API/Main” );
String csGUID = “Content Server”;
ics.AppEvent( “AppEvent” , csGUID ,
              “13:10:00 */*/*” , params );
int errs = ics.GetErrno();
```

See Also

`DeployJSPFile, DestroyEvent, EmailEvent, EnableEvent, SendMail`

CallElement

Processes the content of an element.

Syntax

```
public boolean
CallElement(String element,
            FTVAlList vIn)
```

Parameters

`element`
: The element name listed in the ElementCatalog table.

`vIn`
: List of name/value pairs to be passed as input parameters to the page. A name/value pair supplied here will overwrite the value of any pre-existing variable of the same name. If `vIn` is null, the current variable set is used, and may be modified during the page evaluation.

Description

The `CallElement` method processes the content of an element. The element must exist in the `ElementCatalog`.

Returns

Returns `true` for success and `false` for failure.

errno

If an error occurs, `errno` will be set appropriately. Use `GetErrno` to view the error.

Example

The following code calls an element named `Callee`, passing two arguments:

```
ics.ClearErrno();
FTValList args = new FTValList();
args.removeAll();
args.setValString("flower", "orange blossom");
args.setValString("bird", "mockingbird");
ics.CallElement("Experiment/Callee", args);
int errs = ics.GetErrno();
```

CallSQL

Executes a SQL statement stored in the SystemSQL table.

Syntax

```
public IList
CallSQL(String qryname,
        String listname,
        int limit,
        boolean bCache,
        StringBuffer errstr)
```

Parameters

`qryname`
: Specify the value of the `qryname` column that uniquely identifies the row whose SQL statement should be executed.

`listname`
: The name of the list to be created as a result of the query. The resultset will be accessible from XML using this name. If you set `listname` to `null`, `CallSQL` will generate a unique name for the list.

`limit`
: Maximum number of rows returned; setting limit to `-1` means no limit.

`bCache`
: `true` to cache the results; `false` otherwise. **Setting `bCache` to `false` can hurt performance.** Regardless of the value of `bCache`, the existing cache will be used in the return value. `bCache` only prevents new resultsets from being cached.

`errstr`
: For return values; may contain error information.

Description

The `CallSQL` method executes a SQL statement stored in the `SystemSQL` table.

The `deftable` column of the `SystemSQL` table holds a list of the tables in which the resultsets will be cached. The tables specified in `deftable` *must* all exist. If you specify multiple tables in the `deftable` column, the first table listed is the "primary" table, meaning that the resultset will be cached based on that table's caching rules (maximum cache size, timeout, and so on).

Returns

`IList` object containing the results. An empty result set returns an `IList` with no data.

errno

If `IList` is `null`, an `errno` will be set. Use `GetErrno` to find the error.

Example

The following example executes a SQL statement stored in the SystemSQL table. Before running this code, you would have to place a SQL statement named `Query1` in the SystemSQL table:

```
StringBuffer errStr = new StringBuffer();
ics.ClearErrno()
IList results = ics.CallSQL("Query1", null, -1,
```

```
                                    true, errStr);
    int errNum = ics.GetErrno();
```

See Also

literal, SelectTo, SQL, SQLExp

CatalogDef

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

Syntax

```
public IList
CatalogDef(String from,
           String listname,
           StringBuffer errstr)
```

Parameters

`from`
The name of the table.

`listname`
List name to present to XML side for processing. The name can be `null`, in which case a unique name is generated.

`errstr`
For return values; may contain error information.

Description

The `CatalogDef` method queries a table for its column definition and stores the results in a list. The list contains a row for each column in the table. The `KEY` and `FOREIGN` values will be identical for each row in the resultset, since they are table-specific values, rather than column-specific values.

Returns

IList object containing the results. The column names of this IList are as follows:

`KEY`	Name of table's primary key (`unknown` if table is foreign).
`FOREIGN`	If table is foreign, value is `true`; otherwise, `false`.
`COLNAME`	Name of the column in the table.
`COLTYPE`	Valid values are: `text`, `integer`, `double binary`, `date/time`, `upload`, or `unknown`.
`COLSIZE`	Size of the column based on database information.

errno

If `IList` is `null`, an `errno` will be set. Use `GetErrno` to find the error.

CatalogManager

The CatalogManager method to executes a CatalogManager command. There are two variants of this method:

- `CatalogManager (Variant 1)`: Executes a CatalogManager command.
- `CatalogManager (Variant 2)`: Executes a CatalogManager command and accepts a handle for a BatchContext object.

CatalogManager (Variant 1)

Use this method to execute a CatalogManager command. The list of CatalogManager commands follows:

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

Syntax

```
public boolean
CatalogManager(FTValList vIn)
```

Parameters

`vIn`

List of name/value pairs passed to CatalogManager. The `ftcmd` parameter is required and specifies the CatalogManager command to execute. Any other parameters depend on the CatalogManager command. `Null` indicates that the required parameters are passed in existing Content Server variables.

Returns

Returns a boolean value; `true` if CatalogManager was invoked successfully.

errno

Use `GetErrno` to check the results of the specific CatalogManager command.

Example

The following example deletes a row from the `Movies` table and then examines the return value to determine whether the method worked properly:

```
ics.ClearErrno();
FTValList inList = new FTValList();

inList.setValString("ftcmd", "deleterow");
inList.setValString("tablename", "Movies");
inList.setValString("contentname", "Ishtar");
inList.setValString("Delete uploaded file(s)", "yes");

ics.CatalogManager(inList);
int err = ics.GetErrno();
if( err==0 )
{
  ics.StreamEvalBytes( “Movie deleted. <br>” );
}
else
{
  //handle error
}
```

CatalogManager (Variant 2)

Use this method to execute a CatalogManager command. This variant accepts a handle for a BatchContext object.

Syntax

```
public boolean
CatalogManager(FTValList vIn,
               Object oBatchContext)
```

Parameters

`vIn`
: List of name/value pairs passed to CatalogManager. The `ftcmd` parameter is required and specifies the CatalogManager command to execute. Any other parameters depend on the CatalogManager command. `Null` indicates that the required parameters are passed in existing Content Server variables.

`oBatchContext`
: Handle to the `BatchContext` object.

Description

Use this method to execute a CatalogManager command. This variant accepts a handle for a BatchContext object. The follwing is a list of the CatalogManager commands that accept a Batch Context object:

- `addrow`
- `addrows`

- replacerow
- replacerows
- deleterow
- deleterows

Returns

Returns `true` if CatalogManager was invoked successfully.

errno

Use `GetErrno` to check the results of the specific CatalogManager command.

Example

The following example deletes a row from the `Movies` table and then examines the return value to determine whether the method worked properly:

```
ics.ClearErrno();
FTValList inList = new FTValList();

inList.setValString("ftcmd", "deleterow");
inList.setValString("tablename", "Movies");
inList.setValString("contentname", "Ishtar");
inList.setValString("Delete uploaded file(s)", "yes");

ics.CatalogManager(inList);
int err = ics.GetErrno();
if( err==0 )
{
  ics.StreamEvalBytes( "Movie deleted. <br>" );
}
else
{
  //handle error
}
```

---

addrow

Adds a row to a table.

Parameters

An `FTValList` containing the following name/value pairs:

`ftcmd (required)`
: Value must be set to `addrow`.

`tablename (required)`
: Specify the name of the table to which you plan to add a row.

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

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

`columname_file (required for non-binary files)`
: Specify the name of the file you want to upload. The column name in the table must begin with `url` (for example, `urltext`). The argument name must have `_file` appended to the table's column name (for example, `urltext_file`).

`columnname (optional)`
: Value of each column in the row to be added. The `columnname` parameter is a table column name.

Description

The `addrow` command adds a row to a table. The `addnodes` `TreeManager` command can be committed in the same transaction.

Note

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

errno

Possible values of `errno` include:

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

Example

The following example adds a row to the Movies table:

```
ics.ClearErrno();
FTValList inList = new FTValList();

inList.setValString("ftcmd", "addrow");
inList.setValString("tablename", "Movies");
inList.setValString("Reviewer", "John Doe");

success = ics.CatalogManager(inList);
int err = ics.GetErrno();
if( err==0 )
```

```
{
  ics.StreamEvalBytes( "Movie added. <br>" );
}
else
{
  result = ics.GetErrno();
  //handle error
}
```

See Also

addrows, addrowtext

---

addrows

Adds multiple rows to a table.

Parameters

An `FTValList` containing the following name/value pairs:

`ftcmd (Required)`
: Value must be set to `addrows`.

`tablename (Required)`
: Table name to add rows.

`columnname_folder (Optional)`
: A subfolder name to store the uploaded file. The file is then stored under the upload folder and the subfolder. The upload folder is specified in the SystemInfo table's `defdir` column. You can have multiple upload columns.

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

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

Description

The addrows command adds multiple rows to a table.

errno

Possible values of `errno` include:

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

Example

The following example adds two rows to the `Movies` table:

```
ics.ClearErrno();
FTValList inList = new FTValList();

inList.setValString("ftcmd", "addrows");
inList.setValString("tablename", "Movies");
inList.setValString("urlIshtar_movieFolder", "Ishtar");
inList.setValString("urlCasablanca_movieFolder", "Casablanca");

ics.CatalogManager(inList);
int err = ics.GetErrno();
if( err==0 )
{
  ics.StreamEvalBytes( "Movies added. <br>" );
}
else
{
  //handle error
}
```

See Also

`addrow`, `addrows`, `addrowtext`

---

addrowtext

Adds a row to a table.

Parameters

An `FTValList` containing the following name/value pairs:

`ftcmd (required)`
: Value must be set to `addrowtext`.

`tablename (required)`
: Name of the table to which to add text.

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

`columnname_file (optional)`
: Used to specify a name for a file that will be created on the server. If not provided, the system generates a file name and gives it the extension `.txt`.

`columnname (optional)`
: Value of each column in the row to be added. `columnname` is a table column name.

Description

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

errno

Possible values of `errno` include:

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

Example

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

```
ics.ClearErrno();
FTValList inList = new FTValList();

inList.setValString("ftcmd", "addrowtext");
inList.setValString("tablename", "reviews");
inList.setValString("id", "1234");
inList.setValString("title", "Ishtar");
inList.setValString("subhead", "A film that’s lost in the
desert.");

ics.CatalogManager(inList)
int err = ics.GetErrno();
if (err==0)
{
 ics.StreamEvalBytes( "Movie added. <br>" );
}
else
{
 int result = ics.GetErrno();
 //handle error
}
```

See Also

`addrow`, `addrows`

---

addUser

Adds a user.

Parameters

An `FTValList` containing the following name/value pairs:

`ftcmd (required)`
: Value must be set to `addUser`.

`username (required)`
: Name of user to add.

`password (required)`
Password for the user. The password is stored in an encrypted format.

`useracl (required)`
Access Control List. Comma-separated list of groups.

errno

Possible value of `errno`:

Value	Description
-105	Database error.

Example

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

```
FTValList inList = new FTValList();
ics.ClearErrno();

inList.setValString("ftcmd", "addUser");
inList.setValString("username", "Editor");
inList.setValString("password", "edit1234");
inList.setValString("useracl", "ContentEditor");

ics.CatalogManager(inList);
int err = ics.GetErrno();
if( err==0 )
{
 ics.StreamEvalBytes( "User added. <br>" );
}
else
{
 //handle error
}
```

See Also

`delUser`, `modifyUser`

---

commit

Commits a record from a revision-tracked table into the Revision Tracking subsystem.

Parameters

An `FTValList` containing the following name/value pairs:

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

`tablename (required)`
Name of the table.

`asset (required)`
: Value of the primary key for the row to commit.

`checkout (optional)`
: Indicates lock status of record after commit. Set value to `true` to indicate the record will still be locked by user after the commit.

`annotation (optional)`
: Comment describing the new revision.

Description

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

errno

Possible values of `errno` include:

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

Example

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

```
ics.ClearErrno();
FTValList inList = new FTValList();

inList.setValString("ftcmd", "commit");
inList.setValString("tablename", "reviews");
inList.setValString("asset", "95137");
inList.setValString("checkout", "true");
inList.setValString("annotation", "Fixed error.");

ics.CatalogManager(inList);
int err = ics.GetErrno();
if( err==0 )
{
  ics.StreamEvalBytes( "Record committed. <br>" );
}
else
{
  //handle error
}
```

See Also

`lock`, `unlockrecord`

createtable

Creates a table.

Parameters

An `FTValList` containing the following name/value pairs:

`ftcmd (required)`
: Value must be set to `createtable`.

`tablename (required)`
: Name of the table to create.

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

`uploadDir (required)`
: Server-side folder path to contain uploaded files.

`aclList (optional)`
: Comma-separated list of ACLs. When security is enabled, only members of the ACLs can access content in the table.

`colvalueN (required)`
: Database-specific field type for column. `N` is 0 (zero) for the first column, 1 for the second column, etc.

`colnameN (required)`
: Column field name. Each `colname` parameter corresponds to a column in the table. `N` is 0 (zero) for the first column, 1 for the second column, and so forth.

Description

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

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

errno

Possible values of `errno` include:

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

Example

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

```
ics.ClearErrno();
FTValList inList = new FTValList();
```

```
inList.setValString("ftcmd", "createtable");
inList.setValString("tablename", "ExampleTable");
inList.setValString("systable", "no");
inList.setValString("checkout", "true");
inList.setValString("uploaddir", "/export/home/public");
inList.setValString("colname0", "CS.Property.cc.contentkey");
inList.setValString("colvalue0", "varchar(10)");
inList.setValString("colname1", "description");
inList.setValString("colvalue1", "varchar(24)");
inList.setValString("colname2", "quantity");
inList.setValString("colvalue2", "CS.Property.cc.integer");

ics.CatalogManager(inList);
int err = ics.GetErrno();
if( err==0 )
{
 ics.StreamEvalBytes( "Table created. <br>" );
}
else
{
  //handle error
}
```

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

`tablename``Key`

The `K` in `Key` must be capitalized; for example, the following is valid:

```
cc.TestKey = testname
```

Note that `tablename` must also be named `Test`; otherwise the table name will not get created.

See Also

`deletetable`

---

deletelog

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

Parameters

An `FTValList` containing the following name/value pair:

`ftcmd (required)`
: Value must be set to `deletelog`.

Description

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

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

errno

Possible value of `errno`:

Value	Description
-300	File not found.

Example

This example deletes the log via a URL:

```
ics.ClearErrno();
FTValList inList = new FTValList();

inList.setValString("ftcmd", "deletelog");

ics.CatalogManager(inList);
int err = ics.GetErrno();
if( err==0 )
{
  ics.StreamEvalBytes( "Log deleted. <br>" );
}
else
{
//handle error
}
```

See Also

`exportlog`

---

deleterevision

Deletes a revision of a row from a revision-tracked table.

Parameters

An `FTValList` containing the following name/value pairs:

`ftcmd (required)`
: Value must be set to `deleterevision`.

`tablename (required)`
: Name of the tracked table.

`asset (required)`
: Value of the primary key for the row to obtain history.

`version (optional)`
Possible values are:

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

Description

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

errno

Possible values of `errno` include:

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

Example

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

```
ics.ClearErrno();
FTValList inList = new FTValList();

inList.setValString("ftcmd", "deleterevision");
inList.setValString("tablename", "movies");
inList.setValString("asset", "95137");
inList.setValString("version", "5");

ics.CatalogManager(inList);
int err = ics.GetErrno();
if( err==0 )
{
  ics.StreamEvalBytes( "Revision deleted. <br>" );
}
else
{
    //handle error
}
```

See Also

release

deleterow

Deletes a row in a table.

Parameters

An `FTValList` containing the following name/value pairs:

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

`tablename (required)`
Name of the table containing the row to delete.

`Delete uploaded file(s) (optional)`
Value should be set to `"yes"` if upload files associated with the row should be deleted. Default is `"no"`.

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

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

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

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

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

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

Description

The `deleterow` command deletes a row in a table. The `addnodes` TreeManager command can be committed in the same transaction.

There are two options you can use to delete a row with this command:

- If you want to delete using the primary key, you must set the following parameters:
  - `ftcmd`
  - `tablename`
  - `[primarykey column name]`
  - If the `primarykey` is `id`, you must have a variable called `id set`.
- If you want to delete zero or more rows that match some alternate column value, you must set the following parameters:
  - `ftcmd`
  - `tablename`
  - `tablekey`
  - `tablekeyvalue`

Note

This is valid only if the `primarykey` column does not have a variable set.

errno

Possible value of `errno`:

Value	Description
-105	Database error.

Example

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

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

The first example uses the primary key column to delete a row:

```
ics.ClearErrno();
FTValList inList = new FTValList();

inList.setValString("ftcmd", "deleterow");
inList.setValString("tablename", "movies");
inList.setValString("id", "1");

ics.CatalogManager(inList);
int err = ics.GetErrno();
if( err==0 )
{
 ics.StreamEvalBytes( "Row deleted. <br>" );
}
else
{
   //handle error
}
```

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

```
ics.ClearErrno();
FTValList inList = new FTValList();

inList.setValString("ftcmd", "deleterow");
inList.setValString("tablename", "movies");
inList.setValString("tablekey", "title");
inList.setValString("tablekeyvalue", "The Princess Bride");
```

```
ics.CatalogManager(inList);
int err = ics.GetErrno();
if( err==0 )
{
 ics.StreamEvalBytes( "Row deleted. <br>" );
}
else
{
   //handle error
}
```

After both rows have been removed, the resulting table will look as follows:

id (primary key)	rating	title
3	R	The Godfather

See Also

addrow, deleterows

---

deleterows

Deletes multiple rows in a table.

Parameters

An `FTValList` containing the following name/value pairs:

`ftcmd (required)`
: Values must be set to "`deleterows`".

`tablename (required)`
: Name of the table containing the rows to delete.

`primarykeyN (required)`
: Value of row's primary key. The `primarykey` parameter is the primary key's column name. The value of `N` corresponds to the row number—in the resultset—that you want to delete.

`Delete uploaded file(s)(optional)`
: Value should be set to "`yes`" if the upload files associated with the row should be deleted. Default is "`no`".

Description

The deleterows command deletes multiple rows in a table.

errno

Possible value of `errno`:

Value	Description
-105	Database error.

Example

The following example deletes two rows from the `TopMovies` table:

id	Director	Title
133	Charlie Chaplin	Modern Times
137	Buster Keaton	The General
145	Clint Eastwood	A Perfect World
148	Rob Reiner	The Princess Bride

```
ics.ClearErrno();
FTValList inList = new FTValList();

inList.setValString("ftcmd", "deleterows");
inList.setValString("tablename", "TopMovies");
inList.setValString("id0", "137");
inList.setValString("id1", "145");

ics.CatalogManager(inList);
int err = ics.GetErrno();
if( err==0 )
{
  ics.StreamEvalBytes( "Rows deleted. <br>" );
}
else
{
    //handle error
}
```

See Also

`deleterow`

---

deletetable

Deletes a table.

Parameters

An `FTValList` containing the following name/value pairs:

`ftcmd (required)`
: The literal value `deletetable`.

`tablename (required)`
: The name of the table to delete.

Description

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

errno

Possible values of `errno`:

Value	Description
-103	`tablename` does not exist.
-105	Database error.

Example

The following example deletes the table named `moviereviews`.

```
ics.ClearErrno();
FTValList inList = new FTValList();

inList.setValString("ftcmd", "deletetable");
inList.setValString("tablename", "moviereviews");

ics.CatalogManager(inList);
int err = ics.GetErrno();
if( err==0 )
{
  ics.StreamEvalBytes( "Table deleted. <br>" );
}
else
{
    //handle error
}
```

See Also

`createtable`

---

delUser

Deletes a user.

Parameters

An `FTValList` containing the following name/value pairs:

`ftcmd (required)`
: Value must be set to `delUser`.

`username (required)`
: Name of the user to delete.

errno

Possible value of `errno`:

Value	Description
-105	Database error.

Example

The following example deletes the user `joe`.

```
ics.ClearErrno();
FTValList inList = new FTValList();

inList.setValString("ftcmd", "delUser");
inList.setValString("username", "joe");

ics.CatalogManager(inList);
int err = ics.GetErrno();
if( err==0 )
{
  ics.StreamEvalBytes( "User deleted. <br>" );
}
else
{
   //handle error
}
```

See Also

`addUser`

---

editrow

Modifies the fields of a row in a table.

Parameters

An `FTValList` containing the following name/value pairs:

`ftcmd (required)`
: Value must be set to "`editrow`".

`tablename (required)`
: Name of the table whose row will be edited.

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

`columnname (optional)`
: The information in the field you are editing. The *columnname* parameter is the name of the column to be edited.

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

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

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

```
inList.setValString("tablekeyvalue", "Jaws");
```
This parameter is required only if the `tablekey` parameter is used.

Description

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

errno

Possible values of `errno` include:

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

Example

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

```
ics.ClearErrno();
FTValList inList = new FTValList();

inList.setValString("ftcmd", "editrow");
inList.setValString("tablename", "movies");
inList.setValString("id", "2");
inList.setValString("title", "The African Queen");

ics.CatalogManager(inList);
int err = ics.GetErrno();
if( err==0 )
{
 ics.StreamEvalBytes( "Editing successful. <br>" );
}
else
{
   //handle error
}
```

See Also

`addrowtext`, `editrow`, `replacerow`, `updaterow`

editrows

Modifies the fields for multiple rows in a table.

Parameters

An `FTValList` containing the following name/value pairs:

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

`tablename (required)`
Name of the table containing rows to be edited.

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

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

`columnnameN (optional)`
The `columname` parameter is the name of the column that you want to edit. The value of `N` corresponds to the row number—in the resultset—that you want to edit. For example, if you want to edit the value in the `comments` field of row 0 of the table to read "miss it", you would use the following code:

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

Description

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

errno

Possible values of `errno` include:

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

Example

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

```
ics.ClearErrno();
FTValList inList = new FTValList();

inList.setValString("ftcmd", "editrows");
inList.setValString("tablename", "movies");
inList.setValString("id0", "1");
inList.setValString("comments0", "miss it");
```

```
inList.setValString("id1", "2");
inList.setValString("comments1", "see it");

ics.CatalogManager(inList);
int err = ics.GetErrno();
if( err == 0 )
{
  ics.StreamEvalBytes( "Rows edited. <br>");
}
else
{
  //handle error
}
```

See Also

editrow, replacerows, updaterows

---

exportform

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

Parameters

An `FTValList` containing the following name/value pairs:

`ftcmd (required)`
: Value must be set to `exportform`.

`tablename (required)`
: Name of the table that the form uses.

`ftchoice (required)`
: Value must be set to `addrow`, `deleterow`, `updaterow`, or `replacerow`.

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

errno

Possible value of `errno`:

Value	Description
-105	Database error.

Example

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

```
ics.ClearErrno();
```

```
FTValList inList = new FTValList();

inList.setValString("ftcmd", "exportform");
inList.setValString("tablename", "SystemInfo");
inList.setValString("tablekeyvalue", "id");
inList.setValString("ftchoice", "updaterow");

ics.CatalogManager(inList);
int err = ics.GetErrno();
if( err==0 )
{
  ics.StreamEvalBytes( "Rows edited. <br>" );
}
else
{
  //handle error
}
```

See Also

addrow, deleterow, replacerow, selectrow(s), updaterow

---

exportlog

Exports a per-IP address log.

Parameters

An `FTValList` containing the following name/value pair:

`ftcmd (required)`
You must set the associated value to `exportlog`.

Description

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

errno

Possible value of `errno`:

Value	Description
-300	File not found.

Example

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

`http://host:port/servlet/CatalogManager?ftcmd=exportlog`

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

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

See Also

deletelog

---

flushcatalog

Flushes the internal memory cache for a table.

Parameters

An `FTValList` containing the following name/value pairs:

`ftcmd (required)`
: Value must be set to `flushcatalog`.

`tablename (required)`
: Name of the table to flush.

Description

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

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

errno

Possible value of `errno`:

Value	Description
-105	Database error.

Example

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

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

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

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

flushpage

Deletes all the page's cache file.

Parameters

Create an `FTValList` containing the following name/value pairs:

`ftcmd (required)`
: Specify `flushpage`.

`pagename (required)`
: Specify the name of page to be cleared from cache. The method will flush all pages having this name.

`params (optional)`
: As of CSEE 5.0, these optional parameters no longer have any effect.

Example

The following example shows how to flush the page named `arts2` from the disk cache.

```
ics.ClearErrno();
FTValList inList = new FTValList();

inList.setValString("ftcmd", "flushpage");
inList.setValString("pagename", "region/arts2");

ics.CatalogManager(inList);
int err = ics.GetErrno();
if( err==0 )
{
  ics.StreamEvalBytes( "Page flushed. <br>" );
}
else
{
   //handle error
}
```

history

Gets a revision history for rows in a revision-tracked table.

Parameters

An `FTValList` containing the following name/value pairs:

`ftcmd (required)`
: Value must be set to `history`.

`tablename (required)`
: Name of the tracked table.

`asset (required)`
: Value of the primary key for the row to obtain history.

`version (optional)`
Possible values are:

`first` - history of the earliest revision of the record.

`last` - history of the most recent revision of the record.

`#` - history of a specific revision number.

`date` - history of the record up to the revision current at the specified date. The date must be in SQL format (*yyyy-mm-dd hh:mm:ss*) and is expected to be GMT.

`locker (optional)`
Filters rows returned based on a user who has rows locked. If the current user does not have rtaudit privileges, you must specify a value for `LOCKER`, `CREATOR`, or both. If you specify only `LOCKER`, the value must be the current user's name; otherwise, no data will be returned.

`creator (optional)`
Filters rows returned based on the user who created the row. If the current user does not have rtaudit privileges, you must specify a value for `LOCKER`, `CREATOR`, or both. If you specify only `CREATOR`, the value must be the current user's name; otherwise, no data will be returned.

`annotation (optional)`
Filters rows returned based on the annotation of a row.

Description

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

To obtain history for all rows, the current user must have `rtaudit` privileges on the table. If the current user has read privileges on the table, only history on the revisions which the user has created or locked will be returned.

errno

Possible values of `errno` include:

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

Example

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

```
ics.ClearErrno();
FTValList inList = new FTValList();

inList.setValString("ftcmd", "history");
inList.setValString("tablename", "movies");
inList.setValString("asset", "95137");

ics.CatalogManager(inList);
int err = ics.GetErrno();
if( success )
```

```
{
   //things to do if command is successful
}
else
{
   //handle error
}
```

---

lock

Locks a record for the current user in a revision-tracked table.

Parameters

An `FTValList` containing the following name/value pairs:

`ftcmd (required)`
: Value must be set to `lock`.

`tablename (required)`
: Name of the tracked table.

`asset (required)`
: Value of the primary key for the row to lock.

Description

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

errno

Possible values of `errno` include:

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

Example

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

```
ics.ClearErrno();
FTValList inList = new FTValList();

inList.setValString("ftcmd", "lock");
inList.setValString("tablename", "movies");
inList.setValString("asset", "95137");

ics.CatalogManager(inList);
int err = ics.GetErrno();
```

```
if( err==0 )
{
  ics.StreamEvalBytes( "Record locked. <br>" );
}
else
{
    //handle error
}
```

See Also

release

---

login

Logs in, or authorizes, a user within an application.

Parameters

An FTValList containing the following name/value pairs:

ftcmd (required)
: Value must be set to login.

username (required)
: Name of existing user.

password (required)
: Password of user.

system (optional)
: Domain as hostname to validate with.

errno

Possible value of errno:

Value	Description
-105	Database error.

Example

This example logs in user Tom with password of oscar:

```
ics.ClearErrno();
FTValList inList = new FTValList();

inList.setValString("ftcmd", "login");
inList.setValString("username", "Tom");
inList.setValString("password", "oscar");

ics.CatalogManager(inList);
int err = ics.GetErrno();
if( err==0 )
{
```

```
  ics.StreamEvalBytes( "You are logged in. <br>" );
}
else
{
  //handle error
}
```

See Also

logout

---

logout

Logs out a user from within an application.

Parameters

An `FTValList` containing the following name/value pairs:

`ftcmd (required)`
: Value must be set to `logout`.

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

errno

Possible value of `errno`:

Value	Description
-105	Database error.

Example

The following example logs out the user.

```
ics.ClearErrno();
FTValList inList = new FTValList();

inList.setValString("ftcmd", "logout");
inList.setValString("username", "Tom");
inList.setValString("password", "oscar");

ics.CatalogManager(inList);
int err = ics.GetErrno();
if( err==0 )
{
  ics.StreamEvalBytes( "You are logged out. <br>" );
}
else
{
```

```
    //handle error
}
```

See Also

login

---

mirrorabort

Aborts a mirror operation in progress.

Parameters

An `FTValList` containing the following name/value pair:

`ftcmd (required)`
: Value must be set to `mirrorabort`.

errno

Possible values of `errno` include:

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

Value	Description
-615	Could not delete the temporary table.
-616	Server does not support this mirror operation.
-617	Table is not an Object table.
-618	No tree nodes found matching the nid parameter of FTTreeNode.

Example

The following example aborts a mirror operation.

```
ics.ClearErrno();
FTValList inList = new FTValList();

inList.setValString("ftcmd", "mirrorabort");

ics.CatalogManager(inList);
int err = ics.GetErrno();
if( err==0 )
{
  //things to do if mirror abort is successful );
}
else
{
  //handle error
}
```

See Also

mirrorrows

---

mirrorrows

Copies rows from a table in one database to a table in another database.

Parameters

An FTValList containing the following name/value pairs:

ftcmd (required)
: Value must be set to mirrorrows.

tablename (required)
: Name of the source table.

tablekey (required)
: Name of the table column to perform a selection on. If the system does not find the key, it uses the primary key of the specified table as the selection column. Contentname is the default value. See futuretense.ini to change this default.

tablekeyvalue (required)
: Value in the tablekey column.

`mirrorport (optional)`
HTTP target port. Port 80 is used if a value is not specified.

`mirrortarget (required)`
Network name of the mirror target machine.

`mirrorauthusername (optional)`
Content Server username to log into the target machine. Value is required if security is enabled on the mirror target.

`mirrorauthpassword (optional)`
Password associated with value of `mirrorauthusername` to log into the mirror target. Value is required if security is enabled on the mirror target.

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

`cgiprefix (optional)`
Specifies the application server type on the target system. For iPlanet Application Server, set it to:

```
/NASApp/CS/
```

For WebLogic and WebSphere, set it to:

```
/servlet
```

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

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

errno

Possible values of `errno` include:

Value	Description
`0`	Mirror operation succeeded.
`-105`	Database error.
`-601`	Returned from `mirrorabort` when a mirror operation is not currently in progress.
`-602`	Mirror operation in progress.
`-603`	Required variables are missing from the `mirrorrows catalogmanager` command.
`-604`	The data (file) associated with a url column could not be found on the source machine.
`-605`	The schema for a column on the source machine can not be determined.
`-606`	No rows to mirror from the target system.
`-607`	Current mirroring operation was aborted via the `mirrorabort` command.
`-608`	A connection could not be established to the mirror target server.

Value	Description
-609	The network disconnected communication between the source and target machines.
-610	An attempt was made to mirror from a machine where the database columns are mixed case to a system where the database columns are case-sensitive.
-611	Mirror commit operation failed.
-612	Target server returned an invalid configuration connection message.
-613	Obsolete, will not be returned.
-614	Could not create a temporary table on the target server.
-615	Could not delete the temporary table.
-616	Server does not support this mirror operation.
-617	Table is not an Object table.
-618	No tree nodes found matching the `nid` parameter of `FTTreeNode`.

Example

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

```
ics.ClearErrno();
FTValList inList = new FTValList();

inList.setValString("ftcmd", "mirrorrows");
inList.setValString("tablename", "movies");
inList.setValString("tablekey", "id");
inList.setValString("tablekeyvalue", "your_server");
inList.setValString("mirrortarget", "");
inList.setValString("mirrorauthusername", "ContentServer");
inList.setValString("mirrorauthpassword", "FutureTense");

ics.CatalogManager(inList);
int err = ics.GetErrno();
if( err==0 )
{
  ics.StreamEvalBytes( "Mirroring successful. <br>" );
}
else
{
  //handle error
}
```

See Also

`mirrorabort`

modifyUser

Modifies a user.

Parameters

An `FTValList` containing the following name/value pairs:

`ftcmd (required)`
: Value must be set to `modifyUser`.

`username (required)`
: Name of the user to modify.

`password (optional)`
: Password for the user. The password is stored in an encrypted format.

`useracl (optional)`
: Access Control List. Comma-separated list of groups.

errno

Possible value of `errno`:

Value	Description
-105	Database error.

Example

The following example modifies the user `Joe`:

```
FTValList inList = new FTValList();
ics.ClearErrno();

inList.setValString("ftcmd", "modifyuser");
inList.setValString("username", "joe");
inList.setValString("password", "fred");
inList.setValString("useracl", "ContentEditor");

ics.CatalogManager(inList);
int err = ics.GetErrno();
if( err==0 )
{
  ics.StreamEvalBytes( "User modified. <br>" );
}
else
{
  //handle error
}
```

See Also

`addUser`, `delUser`

release

Releases a lock on a row from a revision-tracked table.

Parameters

An `FTValList` containing the following name/value pairs:

`ftcmd (required)`
: Value must be set to `release`.

`tablename (required)`
: Name of the tracked table.

`asset (required)`
: Value of the primary key for the row to lock.

Description

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

errno

Possible values of `errno` include:

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

Example

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

```
ics.ClearErrno();
FTValList inList = new FTValList();

inList.setValString("ftcmd", "release");
inList.setValString("tablename", "movies");
inList.setValString("asset", "95137");

ics.CatalogManager(inList);
int err = ics.GetErrno();
if( err==0 )
{
  ics.StreamEvalBytes( "Record released. <br>" );
}
else
{
  //handle error
```

```
}
```

See Also

lock

---

replacerow

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

Parameters

An `FTValList` containing the following name/value pairs:

`ftcmd (required)`
: Value must be set to `replacerow`.

`tablename (required)`
: Name of the table that contains the row to replace.

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

`columnname (optional)`
: Each column in the row to be replaced. `columnname` is a table column name.

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

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

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

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

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

Description

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

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

errno

Possible values of `errno` include:

Value	Description
-103	No such table.

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

Example

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

id	title	rating
133	Modern Times	G
137	The General	G
145	A Perfect World	PG
148	The Princess Bride	PG
166	Titanic	PG 13

```
ics.ClearErrno();
FTValList inList = new FTValList();

inList.setValString("ftcmd", "replacerow");
inList.setValString("tablename", "movies");
inList.setValString("id", "166");
inList.setValString("title", "The Great Escape");
inList.setValString("rating", "PG");

ics.CatalogManager(inList);
int err = ics.GetErrno();
if( err==0 )
{
  ics.StreamEvalBytes( "Row replaced. <br>" );
}
else
{
  //handle error
}
```

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

id	title	rating
133	Modern Times	G
137	The General	G
145	A Perfect World	PG
148	The Princess Bride	PG

id	title	rating
166	The Great Escape	PG

See Also

editrow, updaterow, updaterow2

---

replacerows

Replaces multiple rows in a table.

Parameters

An FTValList containing the following name/value pairs:

ftcmd (required)
: Value must be set to replacerows.

tablename (required)
: Name of the table that contains the rows to replace.

primarykeyN (optional)
: Value of row's primary key. The primarykey parameter is the primary key's column name. The value of N corresponds to the row number—in the resultset—that you want to replace.

columnnameN (optional)
: Each column in the row to be replaced. columnname is a table column name. The value of N corresponds to the row number—in the resultset—that you want to replace.

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

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

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

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

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

Description

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

If the row specified by the primarykeyN value does not exist, then the row will be added to the table.

errno

Possible values of `errno` include:

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

Example

Consider a table named movies whose two rows contain the following information:

id	Title	Rating
24	Something About Mary	R
25	The Godmother	R

The following example adds a third row and updates the second row of the preceding table:

```
movies:ics.ClearErrno();
FTValList inList = new FTValList();

inList.setValString("ftcmd", "replacerows");
inList.setValString("tablename", "movies");
inList.setValString("id0", "26");
inList.setValString("title0", "American Pie");
inList.setValString("rating0", "R");
inList.setValString("id1", "25");
inList.setValString("title1", "The Godfather");

ics.CatalogManager(inList);
int err = ics.GetErrno();
if( err==0 )
{
  ics.StreamEvalBytes( "Rows replaced. <br>" );
}
else
{
  //handle error
}
```

The modified table contains the following information; notice that the `Rating` field for the modified title is left blank because it was not explicitly set in the `updaterows` command:

id	Title	Rating
24	Something About Mary	R
25	The Godfather	
26	American Pie	R

See Also

editrows, updaterows

---

rollback

Reverts a row in a revision-tracked table to a previous revision.

Parameters

An `FTValList` containing the following name/value pairs:

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

`tablename (required)`
Name of the tracked table.

`asset (required)`
Value of the primary key for the row to roll back.

`version (optional)`
Possible values are:

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

Description

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

errno

Possible values of `errno` include:

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

Example

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

```
ics.ClearErrno();
FTValList inList = new FTValList();

inList.setValString("ftcmd", "rollback");
inList.setValString("tablename", "movies");
inList.setValString("asset", "95137");
inList.setValString("version", "5");

ics.CatalogManager(inList);
int err = ics.GetErrno();
if( err==0 )
{
  ics.StreamEvalBytes( "Rollback successful. <br>" );
}
else
{
  //handle error
}
```

See Also

`history`

---

selectrow(s)

Executes a query against a given table and displays records from a table.

Parameters

An `FTValList` containing the following name/value pairs:

`ftcmd (required)`
: Value must be set to `selectrow(s)`.

`tablename (required)`
: Name of the table to query.

`tablekeyN (optional)`
: Name of the table column to perform a selection on, where `N` is a counter starting at 0 and ending at the number of rows -1. If the system does not find the key, it uses the primary key of the specified table as the selection column. The default primary key is `contentname`. Change this default in `futuretense.ini.`

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

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

  This parameter can contain the% wildcard.

Description

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

Example

This example selects the row whose `moviename` is `godzilla`.

```
ics.ClearErrno();
FTValList inList = new FTValList();

inList.setValString("ftcmd", "selectrow(s)");
inList.setValString("tablename", "Reviews");
inList.setValString("tablekey1", "title");
inList.setValString("tablekeyvalue", "Godzilla");

ics.CatalogManager(inList);
int err = ics.GetErrno();
if( err==0 )
{
  //things to do when selectrow(s) is successful
}
else
{
  //handle error
}
```

---

setversions

Sets the number of revisions stored for each row in a revision-tracked table.

Parameters

An `FTValList` containing the following name/value pairs:

`ftcmd (required)`
: Value must be set to `setversions`.

`tablename (required)`
: Name of the tracked table.

`numversions (required)`
: Number of revisions to be stored for each row in a tracked table.

Description

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

errno

Possible values of `errno` include:

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

Example

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

```
ics.ClearErrno();
FTValList inList = new FTValList();

inList.setValString("ftcmd", "setversions");
inList.setValString("tablename", "movies");
inList.setValString("numversions", "8");

ics.CatalogManager(inList);
int err = ics.GetErrno();
if( err==0 )
{
  //things to do when setversions is successful
}
else
{
  //handle error
}
```

See Also

`tracktable`

---

tracktable

Enables revision tracking operations for a table.

Parameters

An `FTValList` containing the following name/value pairs:

`ftcmd (required)`
: Value must be set to `tracktable`.

`tablename (required)`
: Name of the table to track.

`numversions (optional)`
: Maximum number of versions to store for each row. When the maximum is reached, the oldest version is removed. If the value of `numversions` is not specified or is set to 0, there is no limit to the number of revisions.

`storage (optional)`
: Location to store revisions associated with a row in the table.

Description

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

errno

Possible values of `errno` include:

Value	Description
`-103`	No such table.
`-104`	No table definition.
`-105`	Database error.
`-1000`	Revision tracking error. This error may be appearing because the user attempting to track the table does not have `rtadmin` privileges.

Example

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

```
ics.ClearErrno();
FTValList inList = new FTValList();

inList.setValString("ftcmd", "tracktable");
inList.setValString("tablename", "Reviews");
inList.setValString("numversions", "5");
inList.setValString("storage", "c:/storage");

ics.CatalogManager(inList);
int err = ics.GetErrno();
if( err == 0 )
{
  //things to do when tracktable is successful
}
else
{
  //handle error
}
```

See Also

`untracktable`

---

untracktable

Stops tracking a table.

Parameters

An `FTValList` containing the following name/value pairs:

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

`tablename (required)`
Name of the table to stop tracking.

Description

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

errno

Possible values of `errno` include:

Value	Description
`-103`	No such table.
`-104`	No table definition.
`-105`	Database error.
`-1000`	Revision tracking error. This error may appear because the user attempting to stop tracking the table does not have `rtadmin` privileges.

Example

This example stops tracking on the `reviews` table.

```
ics.ClearErrno();
FTValList inList = new FTValList();

inList.setValString("ftcmd", "untracktable");
inList.setValString("tablename", "Reviews");

ics.CatalogManager(inList);
int err = ics.GetErrno();

if( err==0 )
{
  //things to do when untracktable is successful
}
else
{
  //handle error
}
```

See Also

`tracktable`

unlockrecord

Unlocks a locked row.

Parameters

An `FTValList` containing the following name/value pairs:

`ftcmd (required)`
: Value must be set to `unlockrecord`.

`tablename (required)`
: Name of the tracked table.

`asset (required)`
: Value of the primary key for the row to unlock.

Description

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

errno

Possible values of `errno` include:

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

Example

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

```
ics.ClearErrno();
FTValList inList = new FTValList();

inList.setValString("ftcmd", "unlockrecord");
inList.setValString("tablename", "movies");
inList.setValString("asset", "95137");

ics.CatalogManager(inList);
int err = ics.GetErrno();

if( err==0 )
{
  //things to do when unlockrecord is successful
}
else
{
  //handle error
}
```

See Also

lock

---

updaterow

Updates field values for a row in a table.

Parameters

An `FTValList` containing the following name/value pairs:

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

`tablename (required)`
Name of the table containing the row to be updated.

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

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

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

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

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

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

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

Description

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

errno

Possible values of `errno` include:

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

Example

The following example updates the comments in the `Movies` table for a given movie:

```
ics.ClearErrno();
FTValList inList = new FTValList();

inList.setValString("ftcmd", "updaterow");
inList.setValString("tablename", "movies");
inList.setValString("id", "1234");
inList.setValString("comments", "three stars");

ics.CatalogManager(inList);
int err = ics.GetErrno();
if( err==0 )
{
  //things to do when updaterow is successful
}
else
{
  //handle error
}
```

See Also

`editrow`, `replacerow`, `updaterow2`

---

updaterow2

Updates or clears values in columns for a row in a table

Parameters

An `FTValList` containing the following name/value pairs:

`ftcmd (required)`
: Value must be set to `updaterow2`.

`tablename (required)`
: Name of the table containing the row to be updated.

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

`columnname (optional)`
: Each column in the row to be modified. `columnname` is a table column name.

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

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

`tablekeyvalue (optional)`

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

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

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

`updaterowNullColumns (optional)`

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

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

Description

Like the `updaterow` command, `updaterow2` updates values in columns for a row in a table. However, where you cannot clear columns with `updaterow`, `updaterow2` allows you to clear columns if there is no value for the specified column (for example, if there is no related field in the form, or if the field on the form was left blank).

errno

Possible values of `errno` include:

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

Example

The following example updates a row in the `movies` table:

```
ics.ClearErrno();
FTValList inList = new FTValList();

inList.setValString("ftcmd", "updaterow2");
inList.setValString("tablename", "movies");
inList.setValString("id", "1234");
inList.setValString("comments", "three stars")

ics.CatalogManager(inList);
int err = ics.GetErrno();
if( err==0 )
{
  //things to do when updaterow2 is successful
}
else
{
  result = ics.GetErrno();
  //handle error
}
```

See Also

editrow, editrows, replacerow

---

updaterows

Modifies field values for multiple rows in a table.

Parameters

An `FTValList` containing the following name/value pairs:

`ftcmd (required)`
: Value must be set to `updaterows`.

`tablename (required)`
: Name of the table containing rows to be updated.

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

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

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

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

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

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

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

Description

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

errno

Possible values of `errno` include:

Value	Description
-103	No such table.

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

Example

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

```
ics.ClearErrno();
FTValList inList = new FTValList();

inList.setValString("ftcmd", "updaterows");
inList.setValString("tablename", "movies");
inList.setValString("id0", "1");
inList.setValString("title0", "godzilla");
inList.setValString("comments0", "miss it");
inList.setValString("id1", "2");
inList.setValString("title1", "titanic");
inList.setValString("comments1", "see it");

ics.CatalogManager(inList);
int err = ics.GetErrno();
if( err==0 )
{
  //things to do when updaterows is successful
}
else
{
  //handle error
}
```

See Also

`editrows`, `replacerows`

---

updaterows2

Modifies field values or clears columns for multiple rows in a table.

Parameters

`ftcmd (required)`
: Value must be set to "`updaterows2`".

`tablename (required)`
: Name of the table containing rows to be updated.

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

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

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

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

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

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

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

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

Description

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

errno

Possible values of `errno` include:

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

Example

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

```
ics.ClearErrno();
FTValList inList = new FTValList();

inList.setValString("ftcmd", "updaterows2");
inList.setValString("tablename", "movies");
inList.setValString("id0", "2");
```

```
inList.setValString("title0", "godzilla");
inList.setValString("comments0", "miss it");
inList.setValString("id1", "8774");
inList.setValString("title"1, "titanic");
inList.setValString("comments1", "see it");

ics.CatalogManager(inList);
int err = ics.GetErrno();

if( err==0 )
{
  //things to do when updaterows2 is successful
}
else
{
  //handle error
}
```

See Also

replacerow, replacerows

ClearErrno

Clears the current error state.

Syntax

```
public void
ClearErrno()
```

Description

The `ClearErrno` method clears the current error state, setting the value of error state to 0. You should call this method immediately before invoking a method that may alter the `errno`.

Note that the error state is more than just the value of the variable `errno`, so `ClearErrno` should be used to properly reset the error state. Using `SetVar("errno",0)` is discouraged, since it does not completely reset the error state.

Example

The following code clears the value of `errno` prior to issuing a new call:

```
ics.ClearErrno();
ics.CatalogManager(inList);
String errno = ics.GetErrno();
```

See Also

`GetErrno, SetErrno`

clientIsSS

This method discovers whether the current request is coming from a browser or from CS-Satellite. There are two variants:

- clientIsSS (Variant 1)
- clientIsSS (Variant 2)

---

clientIsSS (Variant 1)

Given an `ics` object, determines whether the current request is coming from a browser or from CS-Satellite.

Syntax

```
public static boolean
clientIsSS(ICS ics)
```

Parameters

`ics`
: The current context.

Returns

Returns `true` if the client rquest is coming from CS-Satellite. Returns `false` otherwise.

---

clientIsSS (Variant 2)

Given an `ics` object, determines whether the current request is coming from a browser or from CS-Satellite.

Syntax

```
public static boolean
clientIsSS(HttpServletRequest req)
```

Parameters

`req`
: Specify an `HttpServletRequest` object.

Returns

Returns `true` if the client rquest is coming from CS-Satellite. Returns `false` otherwise.

CommitBatchedCommands

Commits CatalogManager and TreeManager commands queued to the batch context.

Syntax

```
public boolean
CommitBatchedCommands(Object oBatchContext)
```

Parameters

`oBatchContext`
The Batch Context object.

Description

The CommitBatchedCommands method commits CatalogManager and TreeManager commands queued to the batch context. The follwing table lists the CatalogManager and TreeManager commands that accept a Batch Context object:

CatalogManager Commands	TreeManager Commands
`addrow`	`addchild`
`addrows`	`addchildren`
`replacerow`	`deletechild`
`replacerows`	`deletechildren`
`deleterow`	
`deleterows`	

Returns

Returns `true` for success and `false` for failure.

Example

The following code creates a transaction consisting of two operations:

```
// Start a batch
Object batchObject = ics.StartBatchContext();


// do a CatalogManager operation
FTValList inList  = new FTValList();
inList.setValString("ftcmd",     "addrow");
inList.setValString("tablename", "TestTable");
inList.setValString("id","100");
// Make sure that you pass the batchObject to CatalogManager
boolean retVal1 = ics.CatalogManager(inList, batchObject);
```

```
// do another CatalogManager operation
inList  = new FTValList();
inList.setValString("ftcmd",     "deleterow");
inList.setValString("tablename", "TestTable");
inList.setValString("id","101");
// Again make sure that you pass the SAME batchObject
boolean retVal2 = ics.CatalogManager(inList, batchObject);

// if both the operations were a success, commit all the changes
// otherwise, roll all of them back
if ( retVal1 && retVal2)
   ics.CommitBatchedCommands(batchObject);
else
   ics.RollbackBatchedCommands(batchObject);
```

See Also

`RollbackBatchedCommands`, `StreamEvalBytes`

CopyList

Copies a list object.

Syntax

```
public boolean
CopyList(String list,
         String newname)
```

Parameters

`list`
The name of the list to copy.

`newname`
The name of the list to be created.

Description

The `CopyList` method copies a list object.

Copying to a list that already exists will destroy the existing list. Cached resultsets corresponding to the list, however, are not destroyed, and are accessible to other threads. To delete cached resultsets, use `FlushCatalog()`.

Returns

Returns `true` for success and `false` for failure.

Example

The following code copies the existing list `oldlist` to a new list named `aNewList`:

```
String oldListName = oldlist.getName();
if ( !ics.CopyList( oldListName, “aNewList” ) )
{
  // error in copying list
}
```

See Also

`GetList (Variant 2), FlushCatalog, RegisterList, RenameList`

dbDebug

Determines whether database debugging is enabled.

Syntax

```
public boolean
dbDebug()
```

Returns

Returns `true` if debugging is enabled; `false` otherwise.

See Also

`eventDebug`, `rsCacheDebug`, `sessionDebug`, `synchDebug`, `systemDebug`, `timeDebug`, `xmlDebug`

decode

Splits the attribute portion of a URL string into a set of attributes.

Syntax

```
public void
decode(String sAttributes,
       Map map)
```

Parameters

`sAttributes`
Attribute string in the following format: `k1=v1&k2=v2&....`

`map`
Set of key/value pairs from the string.

See Also

`encode`

DeleteSynchronizedHash

Deletes a synchronized hash.

Syntax

```
public boolean
DeleteSynchronizedHash(String name)
```

Parameters

`name`
Hash name.

Returns

Boolean success (`false` means no such hash).

errno

The `errno` is cleared, or set if there is an error.

Example

The following example deletes a synchronized hash:

```
boolean delHash = ics.DeleteSynchronizedHash("myHash");
if (!delHash)
{
   //this hash does not exist
}
```

See Also

`GetSynchronizedHash`

DeployJSPData

Deploys the JSP data contained in a buffer.

Syntax

```
public IJSPObject
deployJSPData(String sData,
              String sZone,
              StringBuffer sbError)
```

Parameters

`sData`
: The JSP data.

`sZone`
: The name of the JSP zone to store the data in. This is a relative path within the server doc root. The default value is `/`.

`sbError`
: For return values. This may contain error information.

Returns

`IJSPObject` on success, `null` on error.

DeployJSPFile

Deploys a JSP file.

Syntax

```
public IJSPObject
deployJSPData(String sPath,
              String sZone,
              StringBuffer sbError)
```

Parameters

`sPath`
: The path to the JSP file.

`sZone`
: The name of the JSP zone to store the data in. This is a relative path within the server doc root. The default value is `/`.

`sbError`
: For return values. This may contain error information.

Returns

`IJSPObject` on success, `null` on error.

DestroyEvent

Deletes an event.

Syntax

```
public boolean
DestroyEvent(String name)
```

Parameters

`name`
Name of event to delete.

Returns

Boolean value indicating success or failure.

errno

Use `GetErrno()` to view the error.

Possible value of `errno`:

Value	Description
-203	manage event error

Example

The following example deletes an event named `AppEvent`:

```
ics.ClearErrno();
ics.DestroyEvent("AppEvent");
ics.GetErrno();
```

See Also

`AppEvent, DisableEvent, EmailEvent, EnableEvent`

DisableCache

Disables caching on the ContentServer and EvalServer servlets for the current page evaluation.

Syntax

```
public void
DisableCache()
```

Parameters

None.

Description

The `DisableCache` method disables caching on both the EvalServer and the Content Server servlets for the current page evaluation. This does not affect any other thread and does not affect export.

Use this method when an error is detected in the page that should prevent it from being cached.

Returns

None.

Example

The following example disables caching when an error is detected in the page:

```
int err = ics.GetErrno();
if (err != 0)
  {ics.DisableCache();}
```

DisableEvent

Disables an event.

Syntax

```
public boolean
DisableEvent(String name)
```

Parameters

`name`
: Name of the registered event

Description

The `DisableEvent` method disables an event. While disabled, the event still exists but is never triggered.

Returns

Boolean value indicating success or failure.

errno

Use `GetErrno()` to view the error.

Possible value of `errno`:

Value	Description
-203	manage event error

Example

The following example disables an event:

```
ics.ClearErrno();
ics.DisableEvent("AppEvent");
ics.GetErrno();
```

See Also

`AppEvent, DeployJSPFile, EmailEvent, EnableEvent`

diskFileName (Deprecated)

For CSEE 5.0, the `diskFileName` methods have been disabled. They now simply return `null` under all circumstances.

EmailEvent

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

Syntax

```
public boolean
EmailEvent(String name,
           String tolist,
           String times,
           String filespec)
```

Parameters

`name`

Name to register the event under. Do not use special characters in the event name, (for example, `"{"`, `"*"`, and `"&"`).

`tolist`

A comma-separated list of recipients to receive the message.

`times`

Time at which the email will be triggered. Use the format:

```
hh:mm:ss W/DD/MM
```

where:

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

For each of these fields, it is valid to use an asterisk (all legal values) or a list of elements separated by commas. A value can be one or more numbers separated by a minus sign indicating an inclusive range. For example:

```
2, 5 - 7:0:0 5/*/*
```

means the event is triggered at 2 a.m., 5 a.m., 6 a.m. and 7 a.m. every Friday.

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

`filespec`

Complete file specification of the file whose content will be used as the body of the e-mail message.

Description

The `EmailEvent` method creates an event that sends SMTP mail to one or more recipients at a specific time. To send mail, the Content Server properties file must contain valid values for the `cs.emailhost` and `cs.emailreturnto` properties. To receive mail, the ContentServer properties file must contain valid values for the `cs.emailhost`, `cs.emailaccount`, and `cs.emailpassword` properties.

Returns

Boolean value indicating success or failure.

errno

Use `GetErrno()` to view the error.

Possible value of `errno`:

Value	Description
-200	register failed

See Also

`AppEvent, DeployJSPFile, EnableEvent, SendMail`

EnableEvent

Enables a disabled registered event.

Syntax

```
public boolean
EnableEvent(String name)
```

Parameters

`name`
Name of registered event

Description

The `EnableEvent` method enables a disabled registered event by name. An event is enabled automatically when it is created using `AppEvent` or `EmailEvent`.

Returns

Boolean value indicating success or failure.

errno

Use `GetErrno()` to view the error.

Possible value of `errno`:

Value	Description
-203	manage event error

Example

The following example enables a disabled event:

```
ics.ClearErrno();
ics.EnableEvent("AppEvent");
ics.GetErrno();
```

See Also

`AppEvent, DeployJSPFile, DisableEvent, EmailEvent`

encode

Builds an encoded URL string from a set of attributes.

Syntax

```
public String
encode(String base,
       Map map,
       boolean bSession)
```

Parameters

`base`
: The base of the URL, for example, `"/servlet/ContentServer"`.

`map`
: Set of key/value pairs for each attribute, for example, `pagename/test01`.

`bSession`
: A value of `true` includes the session id in the URL when cookies are turned off on the client's browser. Note that if the session id is included in the URL, the page that URL refers to will not be cached.

Description

The `encode` method is analogous to the `encode` XML and JSP tags. It takes a URL base and a set of key/value pairs and creates a URL like the following:

```
/servlet/ContentServer?pagename=test01
```

`encode` also handles encoding for I18N, and optionally inserts a session id into the URL if cookies are turned off on the client's browser. Note that if the session id is included in the URL, the page that URL refers to will not be cached.

Returns

A String; the encoded URL.

Example

The following examples encode a URL:

Example 1:

```
Map map=newHashMap();
map.put("pagename", "test01");
String url=ics.encode("/servlet/ContentServer", map,true);
```

Example 2:

```
String sUrl =
ics.GetProperty("cs.eventhost")+ics.GetProperty("ft.cgipath")
+ics.GetProperty("ft.approot")+"ContentServer";

Map mapPageCallResargs = new HashMap();
mapPageCallResargs.clear();
mapPageCallResargs.put("pagename", "books/bestsellers");
mapPageCallResargs.put("articleid", "102342");
mapPageCallResargs.put("author", "jones");
```

```
String sEncodedUrl = cs.encode(sUrl, mapCallUrl, true);
```

See Also

decode

eventDebug

Determines whether event debugging is enabled.

Syntax

```
public boolean
eventDebug()
```

Returns

Returns `true` if debugging is enabled, `false` otherwise.

See Also

`rsCacheDebug`, `sessionDebug`, `synchDebug`, `systemDebug`, `timeDebug`, `xmlDebug`

FlushCatalog

Flushes any cached information for a named table.

Syntax

```
public boolean
FlushCatalog(String catname)
```

Parameters

`catname`
The name of the table to flush.

Returns

Returns `true` on success, `false` on failure.

errno

Use `GetErrno()` to view the error.

Example

```
ics.ClearErrno();
ics.FlushCatalog("myCatalog")
ics.GetErrno();
```

FlushStream

Flushes the output stream of pending data. Any data in the cache is untouched.

Syntax

```
public void
FlushStream()
```

genID

Generates a unique ID.

Syntax

```
public String
genID(boolean bClusterSafe)
```

Parameters

`bClusterSafe`
A value of `true` specifies that the ID will be cluster safe.

Returns

Returns the string containing the ID, or `null` on error.

GetBin

Returns the value of a binary posted value.

Syntax

```
public byte[]
GetBin(String name)
```

Parameters

`name`
: The name of the variable.

Returns

Returns the value of the variable on success, null on error.

GetCatalogType

Determines the type of table.

Syntax

```
public int
GetCatalogType(String catalog)
```

Parameters

`catalog`
The name of the table to examine.

Returns

Integer Constant	Meaning
`ICS.NoSuchCatalog`	Table doesn't exist.
`ICS.SystemCatalog`	Table is a system table.
`ICS.ContentCatalog`	Table is a content table.
`ICS.ForeignCatalog`	Table is a foreign table
`ICS.TreeCatalog`	Table is a tree table
`ICS.ObjectCatalog`	Table is an object table.
`ICS.TempCatalog`	Table is a temporary table.
`ICS.TempTreeCatalog`	Table is a temporary tree table.

GetCgi

Returns the value of a specified variable.

Syntax

```
public FTVAL
GetCgi(String name)
```

Parameters

`name`
: The name of a variable.

Description

`GetCGI` is similar to `GetVar`. Both methods return the value associated with the specified variable name. The difference is that `GetCGI` returns the value into an `FTVAL` object but `GetVar` returns the value into a `String` object. For this reason, you should call the `GetCGI` method when a variable holds a binary value.

Despite the name, the `GetCGI` method has nothing to do with CGI programs.

Returns

`FTVAL` value of the variable or `null` if the value of the variable is not defined.

See Also

`GetVar`

getComplexError

Gets error details in the form of a `COM.FutureTense.Util.ftErrors` object.

Syntax

```
public ftErrors
getComplexError()
```

Returns

A `COM.FutureTense.Util.ftErrors` object that contains the following information:

```
public int m_nErrorCode
public int m_nErrorReason
public String[] m_aErrorDetails
```

GetCounter

Gets the value of a counter.

Syntax

```
public int
GetCounter(String counter)
  throws Exception
```

Parameters

`counter`
Name of counter to get.

Returns

Value of counter.

Throws

`Exception`
If there is no counter with this name, the `GetCounter()` method returns `null`. This will cause `Integer.parseInt()` to throw a `NumberFormatException`.

Example

The following example gets the value of a counter:

```
try
{
 int myCounter = ics.GetCounter("testCounter");
}
 catch(Exception e)
 {
  //handle exception
 }
```

See Also

`RemoveCounter, SetCounter`

GetErrno

Gets the current error code as an integer.

Syntax

```
public int
GetErrno()
```

Returns

Current `errno`. Numbers less than 0 are errors.

Example

The following example calls a method (in this case, `ics.CallSQL`), then uses `GetErrno` to determine if the method reported an error.

```
ics.ClearErrno();
results = ics.CallSQL("test2", null, -1, false, errStr);
int errNum = ics.GetErrno;
```

See Also

`ClearErrno, SetErrno`

getIServlet

Returns `IServlet` functionality.

Syntax

```
public IServlet
getIServlet()
```

Returns

Returns `IServlet` functionality.

GetList

The `GetList` method retrieves the `IList` object for the given list name. This method does not create a list; it retrieves an existing list from among those known to Content Server. There are two variants of this method:

- `GetList (Variant 1)`, which retrieves a specified `IList`
- `GetList (Variant 2)`, which retrieves a specified `IList` and can set the `IList`'s cursor to the first position in the list

---

GetList (Variant 1)

Retrieves the `IList` object for the given list name.

Syntax

```
public IList
GetList(String listname)
```

Parameters

`listname`
The name of the list.

Description

The `GetList` method retrieves the `IList` object for the given list name. This method does not create a list, it retrieves an existing list from among those known to ContentServer.

Returns

The requested `IList`, or null if it does not exist.

Example

The following code gets a list whose name is passed in the variable `theListName`:

```
String sListName = ics.GetVar("theListName");
IList theList = ics.GetList(sListName);
```

See Also

`CopyList`

GetList (Variant 2)

Retrieves the `IList` object for the given list name.

Syntax

```
public IList
GetList(String listname,
        Boolean reset)
```

Parameters

`listname`
: The name of the list.

`reset`
: A value of `true` sets the cursor to the first position in the list. A value of `false` maintains the current position in the list.

Description

The `GetList` method retrieves the `IList` object for the given list name. This method does not create a list, it retrieves an existing list from among those known to ContentServer.

Returns

The requested `IList`, or null if it does not exist.

Example

The following code gets a list whose name is passed in the variable `theListName`:

```
String sListName = ics.GetVar("theListName");
IList theList = ics.GetList(sListName, true);
```

See Also

`CopyList, RegisterList, RenameList`

getLocaleString

Returns a locale-specific string.

Syntax

```
public String
getLocaleString(String sName,
                String sLocale);
```

Parameters

`sName`
A key in the SystemLocaleString database table.

`sLocale`
Locale of the string (for example, `en_US` or `fr_CA`).

Returns

The locale-specific message associated with the specified `sName`.

Description

Call `getLocaleString` to access a locale-specific string from the SystemLocaleString database table. This database table holds all the strings that CSEE displays. An overly simplified version of the table looks as follows:

Table 4: Simplified View of SystemLocaleString Table

id	key	locale	Message
100000	continue	en_US	Continue
100001	continue	fr_FR	Continuez
100002	stop	en_US	Stop now.
100003	stop	fr_FR	Arrêt maintenant.

Example

Using the preceding table, you would access the `stop` message appropriate for the French in France locale through the following code:

```
String sName = "stop";
String sLocale = "EN-US";
String Message = ics.getLocaleString(sName, sLocale);
```

GetObj

Retrieves an object that was saved using `SetObj()` or `PushObj()`.

Syntax

```
public Object
GetObj(String name)
```

Parameters

`name`
: Name of the object.

Description

The `GetObj` method retrieves an object that was saved using `SetObj`. Objects managed using `SetObj` and `GetObj` are available within the context of the current page evaluation. They are not persistent across page requests and they cannot be shared by separate page execution threads.

Returns

Object saved. Returns `null` on error.

Example

The following code gets an object:

```
ics.ClearErrno();
clearResult();
MyObject myObj = new MyObject(9,10);
boolean flag = ics.SetObj("myObjName", myObj);
Object strObj = ics.Getobj("myObjName");
errNum = ics.GetErrno();
```

See Also

`PushObj, SetObj`

GetProperty

The `GetProperty` method gets a property value from the currently loaded Content Server property files.

The `GetProperty` method provides the same functionality as the `<CSVAR NAME="CS.Property.PROPERTY_NAME"/>` command in XML.

There are two variants of this method:

- `GetProperty (Variant 1)`: Gets a property value based on the property name you specify.
- `GetProperty (Variant 2)`: Gets a property value based on the property name you specify, allows you to specify the property files you want to search, and allows you to append the current property file.

GetProperty (Variant 1)

Gets a property value from the currently loaded Content Server property files.

Syntax

```
public String
GetProperty(String name)
```

Parameters

`name`
: The name of the property to get.

Description

The `GetProperty` method gets a property value from the currently loaded Content Server property files.

This method provides the same functionality as the following XML command:

```
<CSVAR NAME="CS.Property.PROPERTY_NAME"/>
```

Returns

The value of the property. This value can be `null` or an empty string.

Example

The following code gets the value of the `cs.emailhost` property:

```
String sEmailHost = cs.GetProperty( "cs.emailhost" );
if ( !Utilities.goodString( sEmailHost ) )
{
  // no Email Host set
}
```

See Also

`LoadProperty, RestoreProperty`

GetProperty (Variant 2)

Gets a property value from the specified property files.

Syntax

```
public String
GetProperty(String name,
            String files,
            boolean append)
```

Parameters

`name`
The name of the property to get.

`files`
A semicolon-separated list of property files.

`append`
A value of `true` appends the specified property files to current property file.

Description

The `GetProperty` method gets a property value from the currently loaded Content Server property files.

This method provides the same functionality as the following XML tag:

```
<CSVAR NAME="CS.Property.PROPERTY_NAME"/>
```

Returns

The value of the property. This value can be `null` or an empty string.

Example

The following code gets the value of the `cs.emailhost` property:

```
String sUsername = cs.GetProperty("cs.emailhost",
                                  "futuretense.ini", true);
```

GetSSVar

Gets the value of a session variable.

Syntax-

```
public String
GetSSVar(String name)
```

Parameters

`name`

The name of the session variable.

Returns

The value of the session variable. If the session variable does not exist, returns `null`.

Example

```
// Create a session variable named ssVarUser
String ssvName = "ssVarUser";
ics.SetSSVar( ssvName, "Joe User" );

// Get the value of this session variable
String sv = ics.GetSSVar( ssvName );
// sv should now hold the value "Joe User"

// Delete this session variable
ics.RemoveSSVar( ssvName );

// Try to retrieve the value of the deleted session.
sv = ics.GetSSVar( ssvName );
// sv should now be null
```

See Also

`GetSSVars, RemoveSSVar, SetSSVar`

GetSSVars

Returns an enumeration of all session variables.

Syntax

```
public Enumeration
GetSSVars()
```

Returns

Enumeration of all session variable names.

Example

The following code prints all session variables:

```
Enumeration e = ics.GetSSVars();
while ( e.hasMoreElements() )
{
  out.println(e.nextElement());
}
```

See Also

GetSSVar, RemoveSSVar, SetSSVar

GetSynchronizedHash

Locates or creates a named synchronized hash.

Syntax

```
public ISyncHash
GetSynchronizedHash(String name,
                    boolean bCreate,
                    int timeout,
                    int size,
                    boolean bAbsolute,
                    boolean bClusterSync)
```

Parameters

`name`
: Hashname

`bCreate`
: If `true` and hash is not found, the hash is created using the following parameters. The existing hash is always returned if found.

`timeout`
: Time, in minutes.

`size`
: Size of entries.

`bAbsolute`
: Timeout vs. idle

`bClusterSync`
: Synchronize over cluster

Description

The `GetSynchronizedHash` method locates or creates a named synchronized hash. The hash name is global across Content Server applications.

Returns

Object handle or `null`. The `errno` is cleared, or set if there is an error. Creating a hash with a size less than 1 will return an error.

Example

The following example creates a synchronized hash:

```
ics.ClearErrno();
ics.GetSynchronizedHash("myHash", true, 5, 10, true, false);
int hashErr = ics.GetErrno();
```

See Also

`DeleteSynchronizedHash`

GetVar

Returns the value of a Content Server variable.

Syntax

```
public String
GetVar(String name)
```

Parameters

`name`
The name of the variable.

Returns

The value of the variable. If the variable does not exist, returns `null`.

Example

The following line retrieves the value of the ContentServer variable `username.` The retrieved value is stored in a variable named `sUserName.`

```
String sUserName = cs.GetVar( "username" );
```

See Also

`GetCgi, GetVars, RemoveVar, SetVar`

GetVars

Returns an enumeration of all variable names known to Content Server.

Syntax

```
public Enumeration
GetVars()
```

Returns

Enumeration of all variables names.

Example

The following code gets the value of every variable known to Content Server:

```
Enumeration e = ics.GetVars();
while ( e.hasMoreElements() )
{
  String sVarName = e.nextElement();
  String sVarValue = ics.GetVar( sVarName );
}
```

See Also

GetVar, RemoveVar, SetVar

IndexAdd

Adds an item to the search index.

Syntax

```
public boolean
IndexAdd(String sIndex,
         String sEntryName,
         String sEntryDetail,
         String sWordList,
         String sWordListDelimiters,
         String sFileList,
         String sDate,
         FTValList vlTextArguments,
         FTValList vlFileArguments,
         FTValList vlDateArguments,
         String sEngine,
         String sCharset,
         StringBuffer sbError)
```

Parameters

`sIndex`
: The name of the search index. If `null`, the default index is specified in the Content Server properties `av.defaultindex` or `verity.defaultindex` as appropriate.

`sEntryName`
: Name of the index entry to add. This value is returned in the search results. The `sEntryName` value must be unique within an index. Choose `sEntryName` values carefully so that the caller of the search can reference the associated article(s).

`sEntryDetail`
: The detail string associated with entry. May be `null`. This value is returned in the search results.

`sWordList`
: Words to add to the index. These are space-separated by default. Note that `vITextArguments` is the preferred way to add words to the index.

`sWordListDelimiters`
: String containing delimiter characters to use to separate words in the value of `vITextArguments`. White space will be used as delimiter characters if `sWordListDelimiters` is not specified.

`sFileList`
: Comma-separated list of files to add. Note that `vIFileArguments` is the preferred way to add files.

`sDate`
: Date string to add to the index entry. If no `sDate` is specified, the `IndexAdd` time is returned in the search results. Format is in Java SQL.

`vlTextArguments`
: The text argument names and values. May be `null`.

`vlFileArguments`
: The file argument names and values. May be `null`.

`vlDateArguments`
: The date argument names and values. May be `null`.

`sEngine`
: Search engine name. If `null`, the value of the Content Server property, `cs.searchengine`, is used.

`sCharset`
: Constant value representing the character set that the index uses. For the AltaVista search engine, this value can be 0, 1, or 2 (ISO_LATIN1, UTF8, or ASCII8). If `sCharSet` is `null`, Content Server uses the value of `av.charset` in the properties file.

`sbError`
: For return values; may contain error information.

Returns

Returns `true` on success, `false` on failure.

errno

Use `GetErrno()`, and see `Appendix A, "Error Conditions,"` for the possible `errno` values.

IndexCreate

Creates a search index.

Syntax

```
public boolean
IndexCreate(String sIndex,
            FTValList vlFieldArguments,
            String sEngine,
            String sCharset,
            StringBuffer errstr)
```

Parameters

`sIndex`
Full path name to the index to create. May be null.

`vlFieldArguments`
A list of field names and type values defining the subfields of a Verity search index. For Verity, index subfields must be defined when the index is created. The valid field types are `DATE` and `TEXT`. For AltaVista, this argument must be null.

`sEngine`
Search engine name. If null, the value of the ContentServer property, `cs.searchengine`, is used.

`sCharset`
Constant value representing the character set the index uses.

- For the AltaVista search engine, this value can be 0, 1, or 2 (ASCII7, ASCII8 (Not supported), or UTF8). If `sCharSet` is `null`, ContentServer uses the value of `av.charset` in the properties file.
- For the Verity search engine, this value specifies the name of the subdirectory of the common directory where the locale is defined. If `sCharSet` is `null`, Content Server uses the value of `verity.charset` in the properties file.

`errstr`
For return values; may contain error information.

Returns

Returns `true` on success, `false` on failure.

errno

Use `GetErrno()`, and see `Appendix A, "Error Conditions,"` for the possible `errno` values.

IndexDestroy

Deletes a search index.

Syntax

```
public boolean
IndexDestroy(String sIndex,
             String sEngine,
             String sCharset,
             StringBuffer sbError)
```

Parameters

`sIndex`
: Full pathname of search index to destroy.

`sEngine`
: Search engine name. If null, the value of the Content Server property, `cs.searchengine`, is used.

`sCharset`
: Constant value representing the character set the index uses.

  For the AltaVista search engine, this value can be 0, 1, or 2 (ASCII7, ASCII8 (Not supported), or UTF8). If `sCharSet` is `null`, Content Server uses the value of `av.charset` in the properties file.

  For the Verity search engine, this value specifies the name of the subdirectory of the common directory where the locale is defined. If `sCharSet` is `null`, Content Server uses the value of `verity.charset` in the properties file.

`sbError`
: For return values; may contain error information.

Returns

Returns `true` on success, `false` on failure.

errno

Use `GetErrno()`, and see `Appendix A, "Error Conditions,"` for the possible `errno` values.

IndexExists

Checks for the existence of a search index.

Syntax

```
public boolean
IndexExists(String sIndex,
            String sEngine,
            String sCharset,
            StringBuffer sbError)
```

Parameters

`sIndex`
: Full pathname of search index to find.

`sEngine`
: Search engine name. If null, the value of the Content Server property, cs.searchengine, is used.

`sCharset`
: Constant value representing the character set the index uses. For the AltaVista search engine, this value can be 0, 1, or 2 (ASCII7, ASCII8 (Not supported), or UTF8). If `sCharSet` is `null`, Content Server uses the value of `av.charset` in the properties file.

`sbError`
: For return values; may contain error information.

Returns

Returns `true` if the index exists, `false` otherwise.

errno

Use `GetErrno()`, and see `Appendix A, "Error Conditions,"` for the possible `errno` values.

Example

The following example checks for the existence of an index:

```
ics.ClearErrno();
StringBuffer errs = new StringBuffer();
ics.IndexExists("D:/myIndex", null, null, errs);
int existsErrno = ics.GetErrno();
```

IndexRemove

Removes an entry from an existing search index.

Syntax

```
public boolean
IndexRemove(String sIndex,
            String sEntryName,
            String sEngine,
            String sCharset,
            StringBuffer sbError)
```

Parameters

`sIndex`
: The name of the search index. If `null`, the default index is specified in the ContentServer properties `av.defaultindex` or `verity.defaultindex` as appropriate.

`sEntryName`
: The name of the entry to delete.

`sEngine`
: Search engine name. If `null`, the value of the Content Server property, `cs.searchengine`, is used.

`sCharset`
: Constant value representing the character set the index uses. For the AltaVista search engine, this value can be 0, 1, or 2 (ASCII7, ASCII8 (Not supported), or UTF8). If `sCharSet` is `null`, Content Server uses the value of `av.charset` in the properties file.

`sbError`
: For return values; may contain error information.

Returns

Returns `true` on success, `false` on failure.

errno

Use `GetErrno()`, and see `Appendix A, "Error Conditions,"` for the possible `errno` values.

Example

The following example removes an entry from an existing search index:

```
ics.ClearErrno();
StringBuffer errs = new StringBuffer();
ics.IndexRemove(null, "myEntry", null, null, errs);
int indErrs = ics.GetErrno();
```

IndexReplace

Replaces an existing item in the search index.

Syntax

```
public boolean
IndexReplace(String sIndex,
             String sEntryName,
             String sEntryDetail,
             String sWordList,
             String sWordListDelimiters,
             String sFileList,
             String sDate,
             FTValList vlTextArguments,
             FTValList vlFileArguments,
             FTValList vlDateArguments,
             String sEngine,
             String sCharset,
             StringBuffer sbError)
```

Parameters

`sIndex`
: The name of the search index. If `null`, the default index is specified in the Content Server properties `av.defaultindex` or `verity.defaultindex` as appropriate.

`sEntryName`
: Name of the index entry to add. This value is returned in the search results. The `sEntryName` value must be unique within an index. Choose `sEntryName` values carefully so that the caller of the search can reference the associated article(s).

`sEntryDetail`
: The detail string associated with entry. May be `null`. This value is returned in the search results.

`sWordList`
: Words to add to the index. These are space-separated by default. Note that `vITextArguments` is the preferred way to add words to the index.

`sWordListDelimiters`
: String containing delimiter characters to use to separate words in the value of `vITextArguments`. White space will be used as delimiter characters if `sWordListDelimiters` is not specified.

`sFileList`
: Comma-separated list of files to add. Note that `vIFileArguments` is the preferred way to add files.

`sDate`
: Date string to add to the index entry. If no `sDate` is specified, the `IndexAdd` time is returned in the search results. Format is in Java SQL.

`vlTextArguments`
: The text argument names and values. May be `null`.

`vlFileArguments`
: The file argument names and values. May be `null`.

`vlDateArguments`
: The date argument names and values. May be `null`.

`sEngine`
: Search engine name. If `null`, the value of the Content Server property, `cs.searchengine`, is used.

`sCharset`
: Constant value representing the character set the index uses. For the AltaVista search engine, this value can be 0, 1, or 2 (ISO_LATIN1, ASCII8, or UTF8). If `sCharSet` is `null`, Content Server uses the value of `av.charset` in the properties file.

`sbError`
: For return values; may contain error information.

Returns

Returns `true` on success, `false` on failure.

errno

Use `GetErrno()`, and see `Appendix A, "Error Conditions,"` for the possible `errno` values.

InsertPage

Inserts the results of a Content Server page evaluation at the current position in the output stream.

Syntax

```
public boolean
InsertPage(String pagename,
           FTVAlList vIn)
```

Parameters

`pagename`

Pagename entry in the SiteCatalog.

`vIn`

List of name/value pairs to be passed as input parameters to the page. These are the only variables available to the page. When the evaluation is complete, the original variables are restored. If `vIn` is `null`, the current variable set is used, and may be modified during the page evaluation.

Description

The `InsertPage` method inserts the results of a Content Server page evaluation at the current position in the output stream.

This method invokes a page inline with the current evaluation. It creates a new Content Server instance and evaluates the page in context. The resulting output is streamed as if included inline.

You must have appropriate Content Server ACLs in order to insert the specified `pagnename`.

Returns

Returns `true` for success and `false` for failure.

errno

If an error occurs, `errno` will be set appropriately. Use `GetErrno` to find the error number.

Example

The following code demonstrates how to insert a page. The code does not pass any arguments through the `inList`:

```
String pagename = "FutureTense/Apps/AdminForms/AdminFrame";
FTValList inList = new FTValList();
ics.ClearErrno();
boolean insertpagesuccess = ics.InsertPage(pagename, inList);
if ( ! insertpagesuccess )
{
  int errno = ics.GetErrno();
  // handle error
}
```

See Also

`ReadPage`

ioErrorThrown

Checks to see if an HTTP request has registered an IO error.

Syntax

```
public boolean
ioErrorThrown()
```

Description

The `ioErrorThrown` method checks to see if an HTTP request has registered an IO error. Use this method to check for a broken pipe.

Returns

Returns `true` if there is a registered IO error.

isCacheable

Indicates whether the specified page can be cached.

Syntax

```
public boolean
isCacheable(String pagename)
```

Parameters

`pagename`

Specify the name of the page to check. Specify `null` to check the current page.

Description

Use the `isCacheable` method to determine whether a page can be cached. Note that a page can disable its own cache at any time based on errors or other factors.

You can call `isCacheable` at any time when evaluating pages, though the boolean value it returns may not be the same between calls.

Returns

Returns `true` if the page can be cached and `false` if a page cannot be cached. (The page's `cacheinfo` controls whether it can be cached.)

IsElement

Checks for the existence of an element.

Syntax

```
public boolean
IsElement(String element)
```

Parameters

`element`
The name of the element.

Returns

Returns `true` on success, `false` on failure.

Example

The following code determines whether an element at `JSPs/hello_jsp` exists:

```
boolean exists = ics.IsElement("JSPs/hello_jsp");
if (!exists)
{
   //can’t find element
}
```

isSystemSecure

Checks to see if the current thread is running with Content Server security on.

Syntax

```
public boolean
isSystemSecure()
```

Returns

Returns `true` if Content Server security is enabled. If security is not enabled, no access restrictions will be in place at the core catalog level.

IsTracked

Determines whether Content Server is revision tracking the specified table.

Syntax

```
public boolean
IsTracked(String catalog)
```

Parameters

`catalog`
The name of the table.

Returns

Returns `true` on success, `false` on failure.

Example

The following code determines whether a table named MovieReviews is being revision tracked:

```
boolean tracked = ics.IsTracked("MovieReviews");
if (!tracked)
{
  //table is not tracked
}
```

literal

Generates a literal value specific to a column in a table.

Syntax

```
public String
literal(String sTable,
        String sColumn,
        String sValue);
```

Parameters

`sTable`
The table which contains the column.

`sColumn`
The name of the column.

`sValue`
The value to be converted to a literal.

Description

Content Server works with several underlying DBMSs. Each DBMS requires slightly different treatment of literal values. Call the `literal` method to ensure that you supply a literal value appropriate for your DBMS. The `literal` method relies on values in several properties (such as `cc.stringpicture` and `cc.datepicture`) to format literal values properly.

The SystemSQL table holds SQL statements; the `literal` method is particularly useful for generating variable values that get substituted into those SQL statements. You invoke these SQL statements in a Java program by calling the `CallSQL` method.

Example

Suppose you put the following SQL command into a query named `MyQuery` in the SystemSQL table:

```
SELECT * FROM Names WHERE firstname = Variables.my_literal
```

The following code calls the `literal` method to generate a literal value that is appropriate for the DBMS:

```
String Table = "Names";
String Column = "firstname";
String Value = "Ramin";
String my_literal = cs.literal(Table, Column, Value);
```

The value of `my_literal` will be substituted into the SQL command when your code subsequently invokes `cs.CallSQL`. For example:

```
cs.SetVar("my_literal", my_literal);
IList MyList = cs.CallSQL(MyQuery, listname, -1, true, errstr);
```

Returns

If successful, the method returns a string containing the generated literal. If there is an error, the method returns `null`.

See Also

SQLExp

LoadProperty

Loads a property file or set of property files.

Syntax

```
public boolean
LoadProperty(String name)
```

Parameters

`name`

Name of the property files to load. This is a semicolon-separated list of property files, all in the same folder.

Description

The `LoadProperty` method loads a property file or set of property files. The values loaded from the property files are valid only during the processing of the page containing `LoadProperty`.

Use this method to access a table in a different database, or to load string properties. For example, use it to load a property file to find application-specific values for the local system configuration.

To load property files, those files must be located in the same folder as the Content Server property file. The values loaded from the property file are valid only during the processing of the page containing `LoadProperty`.

If `LoadProperty` fails, the current property set remains intact.

Returns

Returns `true` on success, `false` on failure.

Example

The following call loads one property file:

```
success = ics.LoadProperty("my_special_properties.ini");
```

See Also

`GetProperty, RestoreProperty`

LogMsg

Logs a message to the Content Server log file.

Syntax

```
public void
LogMsg(String msg)
```

Parameters

`msg`
The message to send to the Content Server log file.

Description

The `LogMsg` method logs a message to the Content Server log file. This is useful for debugging or system monitoring.

Example

The following code makes a call and then examines the returned value. If the return value showed a problem, the code calls `LogMsg` to load an error.

```
boolean success = ics.LoadProperty("futuretense.ini");
if (!success)
{
 ics.LogMsg("Failed to load property file.");
 int errno = ics.GetErrno();
 // handle error; current properties are intact

}
```

Mirror

The `Mirror` method copies the rows specified in one or more resultsets to a target Content Server repository. This method is commonly used to copy data from a staging system to a production system.

There are two variants of the Mirror method:

- `Mirror (Variant 1)` copies the rows specified in one or more resultsets to a target Content Server repository
- `Mirror (Variant 2)` copies the rows specified in one or more resultsets to a target Content Server repository and accepts a vector of tree nodes.

To use this tag, you need to have Admin rights. See the *CSEE Administrator's Guide* for complete details.

Mirror (Variant 1)

Copies the rows specified in one or more resultsets to a target Content Server repository.

Syntax

```
public int
Mirror(Vector vILists,
       String sTarget,
       String sUsername,
       String sPassword,
       String sCGIPrefix,
       String sTargetIniFile,
       int nPort,
       boolean bUseProxy,
       boolean bUseHTTPS,
       int nHTTPVersion,
       StringBuffer sbOutput)
```

Parameters

`vILists`
: Vector of `IList` objects to mirror. The resultsets should contain all columns in the designated tables. Columns that exists in the target repository table that are not specified in the resultset are cleared. Columns that exist in the resultset, but not in the target repository table are dropped and no error occurs. You **must** mirror tree nodes using the `vTreeLists` parameter; mirroring tree nodes using the `vILists` parameter will fail.

`sTarget`
: Target server name.

`sUsername`
: User name to use on the remote server. May be `null`.

`sPassword`
: Password to use on remote server. May be `null`.

`sCGIPrefix`
CGI prefix string. May be `null` to indicate that the remote server is the same type as the source.

`sTargetIniFile`
Semicolon-separated list of string property files to push to target server. May be `null` to use the default property file.

`nPort`
Port to use for communication. Zero (0) is interpreted as the default HTTP port (80).

`bUseProxy`
`true` to use proxy server for connection. `false` otherwise.

`bUseHTTPS`
`true` to use secure communication. `false` otherwise.

`nHTTPVersion`
Version of HTTP protocol to be used to connect to remote server. Valid values are `HTTPVERSION10` and `HTTPVERSION11`.

`sbOutput`
Output for return values; may contain error information. May be `null`.

Description

The `Mirror` method copies the rows specified in one or more resultsets to a target Content Server repository. This method is commonly used to copy data from a staging system to a production system.

errno

Use `GetErrno()` to view the error.

Possible values of `errno`:

Value	Description
-618	Mirror bad response.
-619	Temporary table cannot be generated.

Mirror (Variant 2)

Copies the rows specified in one or more resultsets to a target Content Server repository.

Syntax

```
public int
Mirror(Vector vILists,
       Vector vTreeNodes,
       String sTarget,
       String sUsername,
       String sPassword,
       String sCGIPrefix,
       String sTargetIniFile,
       int nPort,
```

```
        boolean bUseProxy,
        boolean bUseHTTPS,
        int nHTTPVersion,
        StringBuffer sbOutput)
```

Parameters

`vILists`
: Vector of `IList` objects to mirror.

  The resultsets should contain all columns in the designated catalogs. Columns that exists in the target repository table that are not specified in the resultset are cleared. Columns that exist in the resultset, but not in the target repository table are dropped and no error occurs. You **must** mirror tree nodes using the `vTreeLists` parameter; mirroring tree nodes using the `vILists` parameter will fail.

`vTreeNodes`
: Vector of `FTTreeNode` objects that describe root nodes of trees to be mirrored. You **must** mirror tree nodes using this parameter; mirroring tree nodes using the `vILists` parameter will fail. The `mirror` method will only mirror the hierarchy; it will not mirror the objects that the nodes point at in the object table.

`sTarget`
: Target server name.

`sUsername`
: User name to use on the remote server. May be `null`.

`sPassword`
: Password to use on remote server. May be `null`.

`sCGIPrefix`
: CGI prefix string. May be `null` to indicate that the remote server is the same type as the source.

`sTargetIniFile`
: Semicolon-separated list of string property files to push to target server. May be `null` to use the default property file.

`nPort`
: Port to use for communication. Zero (0) is interpreted as the default HTTP port (80).

`bUseProxy`
: `true` to use proxy server for connection. `false` otherwise.

`bUseHTTPS`
: `true` to use secure communication; `false` otherwise.

`nHTTPVersion`
: Version of HTTP protocol to be used to connect to remote server. Valid values are `HTTPVERSION10` and `HTTPVERSION11`.

`sbOutput`
: Output for return values; may contain error information. May be `null`.

Description

The `Mirror` method copies the rows specified in one or more resultsets to a target Content Server repository. This method is commonly used to copy data from a staging system to a production system.

errno

Use `GetErrno()` to view the error. Possible values of `errno`:

Value	Description
-618	Mirror bad response.
-619	Temporary table cannot be generated.

NewSeedFromTagname (Deprecated)

Deprecated. *As of version 5.0, use* `runTag`.

Returns the Seed class used to implement a custom tag.

Syntax

```
public Seed
NewSeedFromTagname(String tagname)
```

Parameters

`tagname`
: Name of custom tag (case insensitive)

Returns

Returns the `Seed` class that implements a given dynamic tag. May return `null` if the tag is not implemented dynamically or is undefined in the runtime system.

pageCriteriaKeys

Gets an array of allowable key names for page criteria.

Syntax

```
public String[]
pageCriteriaKeys(String pagename)
```

Parameters

`pagename`
: The name of the page that you want the key names for.

Returns

An array of key names.

pageURL

The `PageURL` method returns the URL of a page, not the whole thread. This method has two variants:

- `pageURL (Variant 1)`: Returns the URL of the current page as a string.
- `pageURL (Variant 2)`: Returns the URL of the specified page and the parameters you passed to it as a string.

---

pageURL (Variant 1)

Returns the URL of the current execution instance (page), not the whole thread.

Syntax

```
public String
pageURL()
```

Returns

The URL of the current page as a string.

---

pageURL (Variant 2)

Returns the URL of the specified page, not the whole thread.

Syntax

```
public String
pageURL(String pname,
        FTValList arguments)
```

Parameters

`pname`
: The name of the page.

`arguments`
: The arguments that were passed to the page in the following format: `arg1=val1&arg2=val2`. This parameter may be empty but may not be null.

Returns

The URL of the current page as a string.

pgCacheDebug

Determines whether page cache debugging is enabled.

Syntax

```
public boolean
pgCacheDebug()
```

Returns

Returns `true` if page cache debugging is enabled, `false` otherwise.

See Also

`rsCacheDebug`, `sessionDebug`, `synchDebug`, `systemDebug`, `timeDebug`, `xmlDebug`

PopObj

Pops the last object with the specified name in the current thread.

Syntax

```
public Object
PopObj(String name)
```

Parameters

`name`
: The name of the object.

Returns

The specified object, or `null` if error.

See Also

`PushObj`

PopVars

Pops the variable stack, returning the variables to their previously stacked state.

Syntax

```
public void
PopVars()
```

Description

The `PopVars` method pops the variable stack, returning the variables to their previously stacked state.

If the stack is empty, `PopVars` does nothing. (The stack will be empty if `PushVars` was not previously called.)

See Also

`PushVars`

PushObj

Saves an object by name, and retains any object previously saved by that same name.

Syntax

```
public boolean
PushObj(String name,
        Object value)
```

Parameters

`name`
: The name of the object.

`value`
: The object you want to push.

Description

The `PushObj` method saves an object by name, and retains any object previously saved by that same name. This method is useful for keeping objects saved during a thread.

See Also

`PopObj`

PushVars

Saves a copy of all the current variables to a stack.

Syntax

```
public void
PushVars()
```

Description

The `PushVars` method saves a copy of all the current variables to a stack. The current variable state is unchanged. It is possible to have more than one entry on the stack.

See Also

`PopVars`

QueryEvents

Queries for a list of events matching the specified criteria.

Syntax

```
public IList
QueryEvents(String sNamePattern,
            String sType,
            Boolean bEnabled,
            String sList)
```

Parameters

`sNamePattern`
: The name pattern. The pattern can contain one wildcard (%), or can be `null` to return all events.

`sType`
: The event type. Valid values are:

- `1` for servlet request
- `2` for email
- `null` for both servlet request and email.

`bEnabled`
: Set this value to `true` for enabled events, `false` for disabled events, or `null` for both.

`sList`
: The name of the `IList`. Use `null` for a system-generated name.

Description

The `QueryEvents` method queries for events matching the specified criteria and returns an `IList` of these events. The events can be queries using an event name pattern, an event type, or whether the event is enabled or disabled.

Returns

Returns an `IList` containing the event names.

ReadPage

Retrieves the results of a Content Server page evaluation.

Syntax

```
public String
ReadPage(String pagename,
         FTValList vIn)
```

Parameters

`pagename`

Pagename entry in the SiteCatalog.

`vIn`

List of name/value pairs to pass as input parameters to the page. These are the only variables available to the page. When the evaluation is complete, the original variables are restored. If `vIn` is `null`, the current variable set is used, and may be modified during the page evaluation.

Description

The `ReadPage` method retrieves the results of a Content Server page evaluation. This method invokes a page inline with the current evaluation. It creates a new ContentServer instance and evaluates the page in context. The resulting output is returned in a string.

You must have appropriate Content Server ACLs in order to read the specified `pagnename`.

Returns

Returns a string containing the results of the page evaluation. Returns `null` on error.

errno

If an error occurs, `errno` will be set appropriately. Use `GetErrno()` to find the error number.

Example

The following code causes the HTML for the specified page to be stored in a string variable named `thePage`:

```
String pagename = "FutureTense/Apps/AdminForms/AdminFrame";
FTValList inList = new FTValList();
ics.ClearErrno();
String thePage = ics.ReadPage ( pagename, inList );
if ( thePage == null )  {
  int errno = ics.GetErrno();
  // handle error
}
else  {
  // Evaluate thePage
}
```

See Also

`InsertPage`

RegisterList

Registers an `IList` by name with Content Server.

Syntax

```
public boolean
RegisterList(String listname,
             IList ilist)
```

Parameters

`listname`
The name that ContentServer will use to refer to this list.

`IList`
The `IList` to be registered.

Description

The `RegisterList` method registers an `IList` by name with Content Server. The registration of the `IList` persists only for the duration of the current page evaluation. The registered name of the `IList` is not persistent across page requests. Once registered, the `IList` can be referenced from an XML tag or by using the `GetList()` method.

If the supplied name is in use in the current scope, this list will replace the existing list object. If the value of the `IList` parameter is `null`, the specified list will be unregistered.

Note that any `IList` created by Content Server is already registered. This method is primarily used to register instances of new `IList` types implemented by the user. If Content Server is to manipulate these lists created by new implementations of the interface, the name of the list must be registered with Content Server first.

Returns

Boolean value indicating the list item was added. An error will be returned if the added list is empty (no rows) or invalid (no columns).

errno

Use `GetErrno()` to view the error number.

Example

The following example registers an `IList` with Content Server:

```
String idVal = "10";
ics.SetVar("id", idVal);
StringBuffer bf = new StringBuffer();
IList ads = ics.SelectTo("NewPortalImage",
                         "urlpicture,href,alttext", "id", null,
                         1, "imagelist", true, bf);
ics.ClearErrno();
ics.RegisterList("image", ads);
int listErr = ics.GetErrno();
```

See Also

`CopyList, GetList, RenameList`

RemoveCounter

Deletes a counter.

Syntax

```
public void
RemoveCounter(String counter)
```

Parameters

`counter`
Name of counter to remove.

Description

The `RemoveCounter` method deletes a counter. After a counter is removed, it is undefined and its value cannot be referenced.

Example

The following example removes a counter named `MyCounter`:

```
ics.RemoveCounter("MyCounter");
```

See Also

`GetCounter, SetCounter`

RemoveSSVar

Removes a session variable.

Syntax

```
public void
RemoveSSVar(String name)
```

Parameters

`name`

The name of the session variable to remove.

Description

The `RemoveSSVar` method removes a session variable. After a session variable is removed, it is no longer defined for the session and its value cannot be referenced.

Example

The following code creates a session variable and then removes it:

```
// First create the session variable.
String ssvName = "ssVarUser";
ics.SetSSVar( ssvName, "Joe User" );
String sv = cs.GetSSVar( ssvName );
// sv should now be the string "Joe User"

// Then remove it.
ics.RemoveSSVar( ssvName );
sv = ics.GetSSVar( ssvName );
// sv should now be null
```

See Also

`GetSSVar`, `GetSSVars`, `SetSSVar`

RemoveVar

Deletes a variable.

Syntax

```
public void
RemoveVar(String name)
```

Parameters

`name`

The name of the variable to delete.

Description

The `RemoveVar` method deletes a variable. After a variable is removed, it is no longer defined for the processing of the page and its value cannot be referenced.

Example

```
// Create a variable named var1
ics.SetVar( "var1", "Born to be deleted." );

// This variable has grown tiresome; delete it.
ics.RemoveVar( "var1" );

// Since the variable has been removed, GetVar will return null
String sv = ics.GetVar( "var1" );
```

See Also

GetVar, GetVars, SetVar

RenameList

Renames a list.

Syntax

```
public boolean
RenameList(String list,
           String newname)
```

Parameters

`list`
The current name of the list.

`newname`
The new name for the list.

Description

The `RenameList` method renames a list.

Renaming to a list that already exists will remove the existing list. Cached resultsets corresponding to the list are flushed and set to null.

Returns

Returns `true` for success and `false` for failure.

Example

The following code renames an `IList` named `oldList` to `LatestList`:

```
String oldname = oldList.getName();
if ( ics.RenameList( oldname, "LatestList" )
{
  // name change successful
}
```

See Also

CopyList, FlushCatalog, GetList, RegisterList

ResolveVariables

This method has the following two variants, both of which resolve any variable, built-in function, query column value, counter, or property embedded in an input string:

- `ResolveVariables (Variant 1)`: Uses the old replacement syntax.
- `ResolveVariables (Variant 2)`: Uses the new replacement syntax.

ResolveVariables (Variant 1)

Resolves any variable, built-in function, query column value, counter, or property embedded in this string using the **old** replacement syntax.

Syntax

```
public String
ResolveVariables(String str)
```

Parameters

`str`
The string that may have embedded variables.

Description

The `ResolveVariables` method resolves any variable, built-in function, query column value, counter, or property embedded in an input string. For example, if passed the following input string:

```
"This is the year: CS.Year"
```

the method returns the following string:

```
"This is the year: 2002"
```

The "old" replacement syntax causes occasional ambiguities. We recommend using Variant 2 instead, which uses the "new" replacement syntax.

Returns

A `String` in which the embedded variables are all resolved.

Example

The following `Greeting` string contains two embedded variables:

```
String Greeting = "Hello, Variables.username. It is cs.Date.";
String sg = ics.ResolveVariables(Greeting);
```

At runtime, `sg` looks similar to the following:

```
"Hello, Tom. It is Mon Mar 06 09:10:07 EST 2002."
```

ResolveVariables (Variant 2)

Resolves any variable, built-in function, query column value, counter, or property embedded in this string using the **new** replacement syntax.

Syntax

```
public String
ResolveVariables(String str,
                 Boolean new)
```

Parameters

`str`
: The string that may have embedded variables.

`new`
: Setting to `true` tells `ResolveVariables` to use the new replacement syntax. Setting to `false` tells `ResolveVariables` to use the old replacement syntax, as described in `ResolveVariables (Variant 1)`.

Description

This variant of `ResolveVariables` uses the new replacement syntax. We recommend using the new syntax rather than the old. This new syntax lets you specify parentheses to clarify the order of evaluation; thus, the syntax is as follows:

`$(`*`Variable`*`)`

For example, if passed the following input string:

```
"This is the year: $(CS.Year)"
```

the method returns the following string:

```
"This is the year: 2002"
```

For simple evaluations, as in the preceding example, the new syntax is not much different from the old syntax. However, for complex evaluations, the new syntax is much clearer. For example, consider the following ambiguous expression that uses the old syntax:

`"Variables.`*`listname`*`.Variables.`*`columnname`*`"`

The preceding expression does not clarify which part of the expression should be evaluated first. Fortunately, the new syntax permits nesting. For example, the following `str` tells `ResolveVariables` to evaluate the expression in stages:

`"$($(Variables.`*`listname`*`).$(Variables.`*`columnname`*`))"`

The order of evaluation in the preceding expression is as follows:

1. Evaluate `Variables.`*`listname`*, yielding `L`
2. Evaluate `Variables.`*`columnname`*, yielding `C`
3. Evaluate `L.C`

Returns

String with embedded variables resolved.

errno

Use `GetErrno` to view the error. Possible values of `errno` include:

Value	Description
`-106`	You set `new` to `true` and specified invalid variable references in `str`.

RestoreProperty

Sets the property file to the default Content Server property file (usually `futuretense.ini`).

Syntax

```
public boolean
RestoreProperty(boolean bClose)
```

Parameters

`bClose`

Controls whether to close the connection to a database that was opened using the current properties. If you set `bClose` to `true`, any open database connection specified by the current properties will be closed before restoring the defaults. This invalidates any resultsets cached against that database. If access to those resultsets is required after `RestoreProperty`, you should set `bClose` to `false`. This will keep the connection open and resultsets against the database will be available.

Returns

Returns `true` on success, `false` on failure.

Example

```
// LoadProperty, ft.new is a property in test.ini
ics.ClearErrno();
success = ics.LoadProperty("test.ini");
ics.SetVar("var1",
           "errno:" + ics.GetErrno() + ", return:" + success);

// RestoreProperty, restores previous set into the system
ics.ClearErrno();
success = ics.RestoreProperty(true);
ics.SetVar("var2", "errno:" + cs.GetErrno() +
           ", return:" + success);
```

See Also

`GetProperty, LoadProperty`

RollbackBatchedCommands

The `RollbackBatchedCommands` method aborts CatalogManager and TreeManager commands that are queued to the Batch Context object.

Syntax

```
public boolean
RollbackBatchedCommands(Object oBatchContext)
```

Parameters

`oBatchContext`
The Batch Context object.

Description

The `RollbackBatchedCommands` method aborts CatalogManager and TreeManager commands that are queued to the Batch Context object. The follwing table lists the CatalogManager and TreeManager commands that accept a Batch Context object:

CatalogManager Commands	TreeManager Commands
addrow	addchild
addrows	addchildren
replacerow	deletechild
replacerows	deletechildren
deleterow	
deleterows	

Returns

Returns `true` if successful.

Example

```
// Start a batch
Object batchObject = ics.StartBatchContext();

// do a CatalogManager operation
FTValList inList  = new FTValList();
inList.setValString("ftcmd",     "addrow");
inList.setValString("tablename", "TestTable");
inList.setValString("id","100");

// Make sure that you pass the batchObject to CatalogManager
boolean retVal1 = ics.CatalogManager(inList, batchObject);

// do another CatalogManager operation
inList  = new FTValList();
inList.setValString("ftcmd",     "deleterow");
inList.setValString("tablename", "TestTable");
inList.setValString("id","101");
```

```
// Again make sure that you pass the SAME batchObject
boolean retVal1 = ics.CatalogManager(inList, batchObject);

// if both the operations were a success, commit all the changes;
// otherwise, roll all of them back
if ( retVal1 && retVal2)
    ics.CommitBatchedCommands(batchObject);
else
    ics.RollbackBatchedCommands(batchObject);
```

See Also

CatalogManager (Variant 2), CommitBatchedCommands, StreamEvalBytes, TreeManager (Variant 2)

rsCacheDebug

The `rsCacheDebug` method indicates whether resultset cache debugging is turned on.

Syntax

```
public boolean
rsCacheDebug()
```

Returns

Returns `true` if resultset caching is enabled.

RTCommit

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

Syntax

```
public int
RTCommit(String sTable,
         String sAsset,
         String sAnnotation,
         boolean bKeepLocked)
```

Parameters

`sTable`
: Name of the tracked table.

`sAsset`
: Value of the primary key for the record being committed.

`sAnnotation`
: Comment that describes the new revision.

`bKeepLocked`
: Indicates the lock status of the record after commit. Set value to `true` to indicate the record will still be locked by user after the commit.

Description

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

Returns

Returns the `errno`. If -1000, check the RT details error.

RTDeleteRevision

Deletes a revision of a row from a revision-tracked table.

Syntax

```
public int
RTDeleteRevision(String sTable,
                 String sAsset,
                 int nVersion)
```

Parameters

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

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

`nVersion`
: Version of the row to delete.

Description

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

Returns

Returns the `errno`. If `-1000`, check the RT details error.

RTHistory

Gets revision history for rows in a revision-tracked table.

Syntax

```
public IList
RTHistory(String sTable,
          String sAsset,
          String sVersion,
          String sLocker,
          String sCreator,
          String sAnnotation,
          String sListName)
```

Parameters

`sTable`
: Name of the tracked table.

`sAsset`
: Value of the primary key for the row whose history to retrieve.

`sVersion`
: String version number, "`first`", "`last`", or a date (in the format of "`YYYY-MM-DD HH:MM:SS`"). "`first`" or "`last`" return the first or last revisions, while a date will return the revision that was or is current as of the specified date.

`sLocker`
: Filters the rows returned based on a user who has rows locked. If the current user does not have `rtaudit` privileges, it is necessary to specify a value for `LOCKER`, `CREATOR`, or both. If only `LOCKER` is specified, the value must be the current user name; otherwise, no data will be returned.

`sCreator`
: Filters the rows returned based on the user who created the row. If the current user does not have `rtaudit` privileges, it is necessary to specify a value for `LOCKER`, `CREATOR`, or both. If only `CREATOR` is specified, the value must be the current user name; otherwise, no data will be returned.

`sAnnotation`
: Filters the rows returned based on the annotation of a row.

`sListName`
: Name of the list.

Description

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

To obtain history for all rows, the current user must have `rtaudit` privileges on the table. If the current user has read privileges on the table, only history on the revisions that the user has created or locked will be returned.

Returns

List containing the returned results.

RTLock

Locks a record by the current user in a revision-tracked table.

Syntax

```
public int
RTLock(String sTable,
       String sAsset)
```

Parameters

`sTable`
Name of the tracked table.

`sAsset`
Value of the primary key of the row to lock.

Description

The `RTLock` method locks a record by the current user in a revision-tracked table.

Once a row is locked, the current user can modify fields in the row. The current user must have write access to the table.

Returns

Returns the `errno`. If -1000, check the RT details error.

RTRelease

Releases a lock on a row from a revision-tracked table.

Syntax

```
public int
RTRelease(String sTable,
          String sAsset)
```

Parameters

`sTable`
: Name of the tracked table.

`sAsset`
: Value of the primary key of the row to release.

Description

The `RTRelease` method releases a lock on a row from a revision-tracked table. Only the user who has the row locked against the table can release it. This will also revert the row in the table to its last committed state, overwriting any changes the user made while the row was locked.

Returns

Returns the `errno`. If `-1000`, check the RT details error.

RTRetrieveRevision

The `RTRetrieveRevision` method returns a specific version for a revision-tracked table. The results returned will be identical to a `"select *"` against the tracked table, assuming the table had been rolled back to the specified version.

This method has two variants:

- `RTRetrieveRevision (Variant 1)`
- `RTRetrieveRevision (Variant 2)`

RTRetrieveRevision (Variant 1)

Returns a specific version for a tracked table.

Syntax

```
public IList
RTRetrieveRevision(String strTableName,
                   String strAsset,
                   int nVersion,
                   String listname)
```

Parameters

`strTableName`
The name of the tracked table.

`strAsset`
The asset to get the results for.

`nVersion`
The version number of the edits.

`listname`
The name of the list to store the results.

Description

The `RTRetrieveRevision` method returns a specific version for a revision-tracked table. The results returned will be identical to a `"select *"` against the tracked table, assuming the table had been rolled back to the specified version.

Returns

Returns the list that stores the results.

RTRetrieveRevision (Variant 2)

Returns a specific version for a revision-tracked table.

Syntax

```
public IList
RTRetrieveRevision(String strTableName,
```

```
                String strAsset,
                String strVersion,
                String listname)
```

Parameters

`strTableName`
: The name of the tracked table.

`strAsset`
: The asset to get the results for.

`strVersion`
: The version number of the edits.

`listname`
: The name of the list that stores the results.

Description

The `RTRetrieveRevision` method returns a specific version for a revision-tracked table. The results returned will be identical to a `"select *"` against the tracked table, assuming the table had been rolled back to the specified version.

This version of the command allows a string version number, `"first"`, `"last"`, or a date (in the format of `"YYYY-MM-DD HH:MM:SS"`). `"first"` and `"last"` return the first and last revisions, while a date will return the revision that was or is current as of the specified date.

Returns

Returns the list that stores the results.

RTRollback

The `RTRollback` method reverts a row in a revision-tracked table to a previous revision. The current user must have the row locked and have `rtaudit` privileges against the table to roll back the row.

This method has two variants:

- `RTRollback (Variant 1)`, which rolls back to a version designated by an integer.
- `RTRollback (Variant 2)`, which rolls back to a version designated by a string.

RTRollback (Variant 1)

Reverts a row in a revision-tracked table to a previous version.

Syntax

```
public int
RTRollback(String sTable,
           String sAsset,
           int nVersion)
```

Parameters

`sTable`
: Name of the table containing the row to revert to a previous version.

`sAsset`
: Value of the primary key for the row to revert to a previous version.

`nVersion`
: Version to make current.

Description

The `RTRollback` method reverts a row in a revision-tracked table to a previous version.

The current user must have the row locked and have `rtaudit` privileges against the table to roll back the row.

This version of `RTRollback()` accepts an integer version number to roll back to.

Returns

Returns the `errno`. If -1000, check the RT details error.

RTRollback (Variant 2)

Reverts a row in a revision-tracked table to a previous version.

Syntax

```
public int
RTRollback(String sTable,
```

```
          String sAsset,
          String strVersion)
```

Parameters

`sTable`
Name of the table containing row to revert to a previous version.

`sAsset`
Value of the primary key for the row to revert to a previous version.

`strVersion`
Version to make current. Specify one of the following strings:

- "first", which rolls back to the initial version.
- "last", which rolls back to the previous version.
- a date in the format `"YYYY-MM-DD HH:MM:SS"`, which rolls back to the version that was current on that date.

Description

The `RTRollback` method reverts a row in a revision-tracked table to a previous revision. The current user must have the row locked and have `rtaudit` privileges against the table to roll back the row.

Returns

Returns the `errno`. If -1000, check the RT details error.

RTSetVersions

Sets the maximum number of revisions stored for each row in a revision-tracked table.

Syntax

```
public int
RTSetVersions(String sTable,
              int nVersions)
```

Parameters

`sTable`
: Table to modify maximum number of versions.

`nVersions`
: Maximum number of versions to store for each row. When the maximum is reached, the oldest version is removed. A value of 0 (zero) indicates an unbounded number of versions will be stored. If the value is smaller than the current number of versions on the table, those versions will remain and can be removed manually.

Description

The `RTSetVersions` method sets the maximum number of revisions stored for each row in a revision-tracked table. The current user must have `rtadmin` privileges on the table.

Returns

Returns the `errno`. If -1000, check the RT details error.

RTTrackTable

Enables revision tracking operations on a table.

Syntax

```
public int
RTTrackTable(String sTable,
             String sStorage,
             int nVersions)
```

Parameters

`sTable`
: Name of the table to track.

`sStorage`
: Location to store revisions associated with a row in the table. This is the root folder. A folder will be created automatically under the given root folder whose name is the same as the table being tracked.

`nVersions`
: Maximum number of versions to store for each row. When the maximum is reached, the oldest version is removed.

Description

The `RTTrackTable` method enables revision tracking operations on a table. The maximum number of revisions per row can be set as well as the storage location for the revisions. The current user must have `rtadmin` privileges on the table.

Returns

Returns the `errno`. If -1000, check the RT details error.

RTUnlockRecord

Unlocks a locked row in a revision-revision-tracked table.

Syntax

```
public int
RTUnlockRecord(String sTable,
               String sAsset)
```

Parameters

`sTable`
Name of the tracked table.

`sAsset`
Value of the primary key of the row to unlock.

Description

The `RTUnlockRecord` method unlocks a locked row. The current user must have `rtadmin` privileges for the table. `RTUnlockRecord` does not restore the row to its most recent revision.

Returns

Returns the `errno`. If -1000, check the RT details error.

RTUntrackTable

Stops tracking a table.

Syntax

```
public int
RTUntrackTable(String sTable)
```

Parameters

`sTable`
: Name of the table to stop tracking.

Description

The `RTUntrackTable` method stops tracking a table. Revision tracking operations will no longer be available on the table. The current user must have `rtadmin` privileges on the table.

Returns

Returns the `errno`. If -1000, check the RT details error.

runTag

Calls an XML tag from within a Java program.

Syntax

```
public String
runTag(String sTag,
       FTValList vIn);
```

Parameters

`sTag`
: Name of the XML tag to invoke. The XML tag must be a singleton, and must not expect any nested argument tags. Do not specify any built-in XML tags, such as `CSVAR`.

`vIn`
: A list containing any name/value pairs you are passing as arguments to the tag specified in `sTag`.

Description

Use `runTag` to call an XML tag from within a Java program. To see if the XML tag ran properly, examine the value of `errno` by calling `GetErrno`.

Returns

The string that would normally be output by the XML tag. Most XML tags do not return any output, so `runTag` usually returns `NULL`.

Example

The Content Server XML tag `OBJECT.SET` expects the following three arguments:

- `NAME`
- `FIELD`
- `VALUE`

The following Java code calls `OBJECT.SET`, passing values for those three arguments:

```
FTValList args = new FTValList();
String output = null;

args.setValString("NAME","obj1");
args.setValString("FIELD","type_char");
args.setValString("VALUE", "c100");

output = cs.runTag("OBJECT.SET", args);
```

Search

Searches an index.

Syntax

```
public IList
Search(String sIndex,
       String sWhat,
       String sRelevance,
       String sQueryParser,
       int nMaxResults,
       FTValList vlAdditionalIndexes,
       String sList,
       String sEngine,
       String sCharset,
       StringBuffer sbError)
```

Parameters

`sIndex`
: The name of the search index. If `null`, the default index is specified in the ContentServer properties `av.defaultindex` or `verity.defaultindex` as appropriate.

`sWhat`
: Query to submit. This should be in the format the given search engine understands.

`sRelevance`
: The relevance string. May be `null`.

`sQueryParser`
: The AltaVista search engine supports three search types—`Simple`, `Advanced`, and `Combined`. `Advanced` is the default.

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

`nMaxResults`
: The maximum number of results to return. Set this to 0 for unlimited results.

`vlAdditionalIndexes`
: The name/value pairs of additional indexes to search. May be `null`.

`sList`
: Name of the output list. Columns returned are:

  - `ENTRY` - Name of the index entry that matches the search criteria.
  - `DETAIL` - Details of the index entry that matches the search criteria.
  - `DATE` - Date the entry was added or the date specified when the index was added. Format is in Java SQL.
  - `RELEVANCE` - Relevance value associated with the search result. The closer the value is to 1, the more useful and relevant the search result is likely to be.

`sEngine`
: Search engine name. If `null`, the value of the Content Server property, `cs.searchengine`, is used.

`sCharset`
Constant value representing the character set the index uses.

- For the AltaVista search engine, this value can be 0, 1, or 2 (ISO_LATIN1, ASCII8, or UTF8). If `sCharSet` is `null`, Content Server uses the value of `av.charset` in the properties file.
- For the Verity search engine, this value specifies the name of the subdirectory of the common directory where the locale is defined. If `sCharSet` is `null`, Content Server uses the value of `verity.charset` in the properties file.

`sbError`
For return values; may contain error information.

Description

The `Search` method searches an index. The resultset of the query is stored in a list.

Returns

Returns the name/value pairs of the result set. If an error occurs, or if there are no results satisfying the query, then a `null` is returned.

errno

Use `GetErrno()`, and see `Appendix A, "Error Conditions,"` for the possible `errno` values.

SearchDateToNative

Converts a date string formatted in Java SQL to the date format native to the search engine.

Syntax

```
public boolean
SearchDateToNative(String sDate,
                   StringBuffer sbOut,
                   String sEngine,
                   String sCharset,
                   StringBuffer sbError)
```

Parameters

`sDate`
The date to convert.

`sbOut`
Buffer where the converted string will be returned.

`sEngine`
Search engine name. If `null`, the value of the Content Server property, `cs.searchengine`, is used.

`sCharset`
Constant value representing the character set the index uses.

- For the AltaVista search engine, this value can be 0, 1, or 2 (ISO_LATIN1, ASCII8, or UTF8). If `sCharSet` is `null`, Content Server uses the value of `av.charset` in the properties file.
- For the Verity search engine, this value specifies the name of the subdirectory of the common directory where the locale is defined. If `sCharSet` is `null`, Content Server uses the value of `verity.charset` in the properties file.

`sbError`
For return values; may contain error information.

Returns

Returns `true` on success, `false` on failure.

errno

Use `GetErrno`(), and see `Appendix A, "Error Conditions,"` for the possible `errno` values.

SelectTo

Performs a simple query from a table.

Syntax

```
public IList
SelectTo(String from,
         String what,
         String where,
         String orderby,
         int limit,
         String listname,
         boolean bCache,
         StringBuffer errstr)
```

Parameters

`from`
Table to query. The table **must** exist.

`what`
Comma-separated list of table columns to return. The string * means all columns.

`where`
Comma-separated list of column names which must have associated variables. The value of the variable is used to match the contents of that column. The variables in the comma-separated list **must** exist; if they do not, `ICS.SelectTo()` returns error `-106`. When you specify a date/time field in *`where`*, Content Server uses the `cc.datepicture` property to convert the supplied datetime to the proper format.

`orderby`
Comma-separated list of columns used to sort results. The ascending/descending order is database-dependent.

`limit`
The maximum number of rows returned from the query. To indicate no limit, set this value to -1.

`listname`
The name of the list created. The list can be referenced using this name from an XML tag or by using the `getList` method. If no listname is specified, the system will generate the name of the list.

`bCache`
`true` to cache the results; `false` otherwise. Regardless of the value of `bCache`, the existing cache will be used to generate the return value, but new resultsets will not be cached.

`errstr`
For return values; may contain error information.

Description

The `SelectTo` method performs a simple query from a table. The results are stored in a list.

Returns

`IList` object that contains the results. An empty result set returns a value of null.

errno

If `IList` is `null`, use `GetErrno` to find the error.

Example

```
// If forumId isn't set, list all forums
if (forumId != null)
  {
    ics.SetVar(Msg.forumid, forumId);
    mList = ics.SelectTo(Msg.forumXTable, Msg.allFields,
                    (forumId == null) ? null : Msg.forumid, null,
                     Msg.noLimit, null, Msg.cacheResults, errstr);
  }
if (mList != null)
  {
    mColNames = Utilities.words(forumCols, Msg.comma);
    mCols = mColNames.size();
  }
```

See Also

`CallSQL, SQL`

SendMail

The SendMail method sends an SMTP e-mail message to one or more recipients. There are two variants, whose differences are as follows:

- SendMail (Variant 1) lets you specify only to, subject, and body.
- SendMail (Variant 2) lets you specify several additional parameters.

The Utilities.sendMail method also lets you send an SMTP e-mail message to one or more recipients.

SendMail (Variant 1)

Sends an SMTP mail message to one or more recipients.

Syntax

```
public boolean
SendMail(String to,
         String subject,
         String body)
```

Parameters

to
: A valid e-mail address; for example, "joe@FatWire.com". You can optionally specify a comma-separated list; for example:

```
"joe@FatWire.com, jane@FatWire.com"
```

subject
: The subject of the message. May be null if the body is not null.

body
: The message body. May be null if the subject is not null.

Returns

Returns true on success, false on failure.

Description

The SendMail method sends an SMTP message to one or more recipients. To send mail, the ContentServer properties file must contain valid values for the cs.emailhost and cs.emailreturnto properties. To receive mail, the ContentServer properties file must contain valid values for the cs.emailhost, cs.emailaccount, and cs.emailpassword properties.

errno

Use GetErrno() to view the error.

Possible values of errno include:

Value	Description
-106	Bad parameters.

Value	Description
-201	Can’t send e-mail.

See Also

EmailEvent

SendMail (Variant 2)

Sends an SMTP mail message to one or more recipients.

Syntax

```
public boolean
SendMail(String to,
         String replyto,
         String subject,
         String body,
         String contenttype)
```

Parameters

`to`
A valid e-mail address; for example, "`joe@FatWire.com`". You can optionally specify a comma-separated list; for example:

```
"joe@FatWire.com, jane@FatWire.com"
```

`replyto`
The email address you want replies sent to. If a value is not supplied, replies will be sent to the sender’s address.

`subject`
The subject of the message. May be null if the `body` is not null.

`body`
The message body. May be null if the `subject` is not null.

`contenttype`
The MIME type of the content you are sending. Default is `text/plain`.

Returns

Returns `true` on success, `false` on failure.

Description

The `SendMail` method sends an SMTP message to one or more recipients. To send mail, the ContentServer properties file must contain valid values for the `cs.emailhost` and `cs.emailreturnto` properties. To receive mail, the ContentServer properties file must contain valid values for the `cs.emailhost`, `cs.emailaccount`, and `cs.emailpassword` properties.

errno

Use `GetErrno()` to view the error.

Possible values of `errno` include:

Value	Description
`-106`	Bad parameters.
`-201`	Can’t send e-mail.

See Also

`EmailEvent`

sessionDebug

Determines whether session debugging is enabled.

Syntax

```
public boolean
sessionDebug()
```

Returns

Returns `true` if session debugging is enabled, `false` otherwise.

SessionExists

Determines whether a session exists.

Syntax

```
public boolean
SessionExists(String id)
```

Parameters

`id`

The ID of the session to be tested.

Returns

Returns `true` if `id` is valid, `false` otherwise.

Example

```
String id = ics.GetVar(sID);
boolean exists = ics.SessionExists(id);
if (exists)
{
  //things to do if session exists
}
```

See Also

SessionID

SessionID

Returns the current session's ID.

Syntax

```
public String
SessionID()
```

Description

The `SessionID` method turns the current session's ID. The form and length of the session ID that is returned is dependent upon the application server.

Returns

A string containing the current session’s id. If the session is inactive or if the session ID cannot be determined, the method returns `null`.

Example

```
String sID = ics.SessionID();
```

See Also

SendMail()

setComplexError

Sets a complex error using the `ftErrors` class.

Syntax

```
public void
setComplexError(ftErrors error)
```

Parameters

`error`
The error object.

See Also

`getComplexError()`

SetCookie

Sets a cookie on the client.

Syntax

```
public boolean
SetCookie(String name,
          String value,
          int timeout,
          String url,
          String domain,
          boolean bSecure)
```

Parameters

`name`
: The name of the cookie to set.

`value`
: The value to set the cookie to. Setting the value to `null` removes the cookie.

`timeout`
: The timeout period for the cookie on the client, in seconds.

`url`
: Valid URL. This value restricts the sending of the cookie from the client to this URL only. Defaults to `"/"` if omitted.

`domain`
: Valid domain name. This value restricts the sending of the cookie from the client to addresses in this domain only.

`bSecure`
: Indicates whether security is set on this cookie.

Description

The `SetCookie` method sets a cookie on the client.

`SetCookie` must appear before anything that streams back data on the page or the cookie will not be set. This is an HTTP requirement.

To remove a cookie, set the cookie value to `null`.

Returns

Returns `true` on success, `false` on failure.

Example

```
ics.ClearErrno();
success = ics.SetCookie("acookie", "sweet", 100, "/", "/" , false
);
int errnum = ics.GetErrno();
if (success)
  ics.SetSSVar ("var4" , cs.GetSSVar("acookie"));
else
 ics.SetSSVar ("var4" , "failed");
```

SetCounter

Creates a counter, or changes the value of an existing counter.

Syntax

```
public void
SetCounter(String counter,
           int value)
      throws Exception
```

Parameters

`counter`
Counter name to set.

`value`
The value to set the counter to.

Throws

Exception if failure.

Example

The following example creates a counter:

```
try
      {
       ics.SetCounter("newCounter", 1);
      }
catch(Exception e)
      {
       //handle exception
      }
```

See Also

`GetCounter`, `RemoveCounter`

SetErrno

Sets an error code.

Syntax

```
public void
SetErrno(int i)
```

Parameters

`i`

Value of the error code.

Description

The `SetErrno` method sets an error code. To avoid setting error codes that may conflict with the existing Content Server codes, make sure all error codes are set to:

```
(ftErrors.maxErrno + yourErrno)
```

where `yourErrno` is a negative integer.

Example

```
ics.ClearErrno();
boolean success = ics.myMethod()
if (!success)
   {
 ics.SetErrno((ftErrors.maxErrno + -4))
};
```

See Also

`ClearErrno, GetErrno`

SetObj

Stores an object by name.

Syntax

```
public boolean
SetObj(String name,
       Object value)
```

Parameters

`name`
: Name of the object.

`value`
: Value of object to save. `null` removes the object.

Description

The `SetObj` method stores an object by name. This method is useful for keeping objects saved during a thread. Any previous objects saved by the same name will be lost.

Returns

Returns `true` on success, `false` on failure.

errno

Use `GetErrno()` to view the error.

Example

```
// Set an existing Object myObj and store the name in string
myObjName
clearResult();
MyObject myObj = new MyObject(9,10);
boolean flag = ics.SetObj("myObjName",myObj);
errNum = ics.GetErrno();
showResult();
```

See Also

`GetObj`

SetSSVar

The `SetSSVar` method sets the value of a session variable. The value of a session variable exists for the duration of the session, unless it is explicitly deleted using `RemoveSSVar`.

This method has two variants:

- `SetSSVar (Variant 1)` sets the value of a session variable where the value of the variable is a string.
- `SetSSVar (Variant 2)` sets the value of a session variable where the value of the variable is an integer.

SetSSVar (Variant 1)

Sets the value of a session variable.

Syntax

```
public void
SetSSVar(String name,
         String value)
```

Parameters

`name`
: The name of the session variable to set.

`value`
: The value to set the session variable to.

Description

The `SetSSVar` method sets the value of a session variable. The value of a session variable exists for the duration of the session, unless it is explicitly deleted using `RemoveSSVar`.

Example

The following fragment creates a session variable named `FavoriteColor` and sets it to `red`:

```
ics.SetSSVar("FavoriteColor", "red");
```

See Also

`GetSSVar, GetSSVars, RemoveSSVar`

SetSSVar (Variant 2)

Sets the value of a session variable.

Syntax

```
public void
SetSSVar(String name,
         int value)
```

Parameters

`name`
: The name of the session variable to set.

`value`
: The value to set the session variable to.

Description

The `SetSSVar` method sets the value of a session variable. The value of a session variable exists for the duration of the session, unless it is explicitly deleted using `RemoveSSVar`.

Example

The following fragment creates a session variable named `FavoriteColor` and sets it to `red`:

```
ics.SetSSVar("FavoriteColor", "red");
```

See Also

`GetSSVar, GetSSVars, RemoveSSVar`

SetVar

The SetVar method sets a Content Server variable. This method has three variants:

- SetVar (Variant 1): Sets the value of a Content Server variable to a string.
- SetVar (Variant 2): Sets the value of a Content Server variable to an integer.
- SetVar (Variant 3): Sets the value of a Content Server variable to an FTValList.

SetVar (Variant 1)

Sets the value of a ContentServer variable.

Syntax

```
public void
SetVar(String name,
       String value)
```

Parameters

name
: The name of the variable to create or set.

value
: The value of the variable.

Description

The SetVar method sets the value of a ContentServer variable. If the variable does not exist, it will be created. This method provides functionality similar to the XML SETVAR tag.

Example

The following example creates a variable named weather and set its value to Sunny:

```
ics.SetVar("weather", "Sunny");
```

See Also

GetVar, GetVars, RemoveVar

SetVar (Variant 2)

Sets the value of a ContentServer variable.

Syntax

```
public void
SetVar(String name,
       Int value)
```

Parameters

`name`
The name of the variable to create or set.

`value`
The value of the variable.

Description

The `SetVar` method sets the value of a ContentServer variable. If the variable does not exist, it will be created. This method provides functionality similar to the XML `SETVAR` tag.

Example

The following example creates a variable named `weather` and set its value to `Sunny`:

```
ics.SetVar("weather", "Sunny");
```

See Also

`GetVar, GetVars, RemoveVar`

SetVar (Variant 3)

Sets the value of a ContentServer variable.

Syntax

```
public void
SetVar(String name,
       FTVal value)
```

Parameters

`name`
The name of the variable to create or set.

`value`
The value of the variable.

Description

The `SetVar` method sets the value of a Content Server variable. If the variable does not exist, it will be created. This method provides functionality similar to the XML `SETVAR` tag.

Example

The following example creates a variable named `weather` and set its value to `Sunny`:

```
ics.SetVar("weather", "Sunny");
```

See Also

`GetVar, GetVars, RemoveVar`

SQL

Executes a SQL statement.

Syntax

```
public IList
SQL(String from,
    String sqlstmt,
    String listname,
    int limit,
    boolean bCache,
    StringBuffer errstr)
```

Parameters

`from`

The name of the table(s) to which the resultset should be cached. To specify multiple tables, separate the table names with commas. For example, the following string tells the SQL method to cache the resultset in tables named `apples` and `bananas`:

```
"apples,bananas"
```

If you specify multiple tables, the first table listed is the "primary" table, meaning that the resultset will be cached based on that table's caching rules (maximum cache size, timeout, and so on).

`sqlstmt`

The SQL statement to execute.

`listname`

The name of the list created. The list can be referenced using this name from an XML tag or using the `GetList` method. If you specify `null`, SQL will generate the name of the list for you.

`limit`

The maximum number of rows returned from the query. To indicate no limit, set this value to `-1`.

`bCache`

`true` to cache the results; `false` otherwise. Regardless of the value of `bCache`, the existing cache will be used to generate the return value, but new resultsets will not be cached. **Setting `bCache` to false can hurt performance.**

`errstr`

For return values; may contain error information.

Description

The `SQL` method executes a SQL statement.

The SQL query may be database-specific. The resultset associated with the query is registered against the table(s) specified in the `from` parameter.

The SystemSQL table allows the specification of the name of the table to cache the results of the query. The resultset associated with the query is registered against the table name designated in the SystemSQL table.

Returns

`IList` object containing the results. When certain SQL statements (delete, update, and so on) are executed successfully, no resultset, and hence no `IList`, is returned.

errno

If `IList` is `null`, use `GetErrno` to find the error. When certain SQL statements (`delete`, `update`, and so on) are executed successfully, no resultset, and hence no `IList`, is returned. If no `IList` is returned, `GetErrno` returns error number `-502`. This does not necessarily indicate an error condition.

Example

See the example for `SQLExp`.

SQLExp

Creates a string that can be used as part of an SQL WHERE clause.

Syntax

```
public String
SQLExp(String table,
       String type,
       String verb,
       String arg,
       String column
       String exp)
```

Parameters

`table`
The table for which the statement is being executed.

`type`
The expression type. Specify one of the following constants:

- `ICS.SQLExpAND` (AND operation)
- `ICS.SQLExpOR` (OR operation)
- `ICS.SQLExpIN` (IN operation)

`verb`
The verb for the `WHERE` clause (for example, "`LIKE`", "`=`", "`!=`", "`IN`", "`NOT IN`").

`arg`
Comma-separated list of values for the expression.

`column`
The column for which the expression is being generated.

`exp (Optional)`
An optional expression; for example:

```
lower(color)
```

Description

The `SQLExp` method creates a string that can be used as part of a SQL where clause. For a description of SQL where clauses, see a SQL tutorial.

`SQLExp` takes a comma-separated string of items and a column name and places a relational operator between each item/column name pair. A logical operator is placed between each item/column name comparison. `SQLExp` deals with values of different types correctly. If you provide an argument for `exp`, the argument

Use the optional `exp` argument to specify an expression to be used as part of the where clause.

The following table demonstrates some sample input parameters and the affect of `exp` on the resulting where clause:

Input Parameters						Resulting Value of where clause
table	type	verb	arg	column	exp	
Flora	AND	=	green	color	omitted	color='green'
Flora	AND	=	green	color	color	color='green'
Flora	AND	=	green	color	lower(color)	lower(color)='green'

Returns

String containing the generated expression.

Example

Consider a database table named `Movies` that contains the following rows:

id	title	director
1000001	The General	Buster Keaton
10000003	Jules et Jim	Francois Truffaut
10000004	Seven Samurai	Akira Kurosawa
10000005	General Post	Thomas Bentley

The following code uses `SQLExp` (without an `exp`) to create a SQL expression that will ultimately find all titles that contain the word `General`:

```
String table = "Movies";
String type = ICS.SQLExpOR;
String verb = "LIKE";
String arg  = "General";
String column = "title";
String WhereClause = ics.SQLExp(table, type, verb,
                                arg, column);
```

When executed, `WhereClause` will contain:

```
title LIKE '%General%'
```

The following code uses the returned `WhereClause` as a portion of a larger SQL statement and then invokes that SQL statement:

```
String SQLStart = "SELECT title,director FROM Movies WHERE ";
String SQLStatement = SQLStart + WhereClause;
String from = "Movies";
String listname = null;
int limit = 5;
boolean bCache = true;
StringBuffer errstr = new StringBuffer(128);
ics.ClearErrno();
IList MyList = ics.SQL(from, SQLStatement, listname,
```

```
                                    limit, bCache, errstr);
```

`MyList` should now contain the rows from the `Movies` table that matched the query; that is, the rows whose `title` contained `General`.

See Also

`literal`

StartBatchContext

Fetches the `BatchContext` object to queue CatalogManager and TreeManager commands.

Syntax

```
public java.lang.Object
StartBatchContext()
```

Description

The `StartBatchContext` method fetches the BatchContext object to queue CatalogManager and TreeManager commands.

The following table lists the CatalogManager and TreeManager commands that accept a `BatchContext` object:

CatalogManager Commands	TreeManager Commands
`addrow`	`addchild`
`addrows`	`addchildren`
`replacerow`	`deletechild`
`replacerows`	`deletechildren`
`deleterow`	
`deleterows`	

Returns

Returns the `BatchContext` object.

Example

```
// Start a batch
Object batchObject = ics.StartBatchContext();

// Perform a CatalogManager operation
FTValList inList  = new FTValList();
inList.setValString("ftcmd",     "addrow");
inList.setValString("tablename", "TestTable");
inList.setValString("id","100");

// Make sure that you pass the batchObject to CatalogManager
boolean retVal1 = ics.CatalogManager(inList, batchObject);

// do another CatalogManager operation
inList  = new FTValList();
inList.setValString("ftcmd",     "deleterow");
inList.setValString("tablename", "TestTable");
inList.setValString("id","101");
// Again make sure that you pass the SAME batchObject
boolean retVal1 = ics.CatalogManager(inList, batchObject);
```

```
// if both the operations were a success, commit all the changes
// otherwise, roll all of them back
if ( retVal1 && retVal2)
    ics.CommitBatchedCommands(batchObject);
else
    ics.RollbackBatchedCommands(batchObject);
```

See Also

CommitBatchedCommands, RollbackBatchedCommands

StreamBinary

Immediately streams binaries to the ICS context.

Syntax

```
public void
StreamBinary(Byte[] b,
             int offset,
             int length)
```

Description

The `StreamBinary` method immediately streams binaries out to the ICS context. This output is unbuffered, and as such may appear before text already in the element's buffered output stream.

Bytes that are output through this method are not subject to page caching.

Parameters

`b`
: An array of bytes to stream.

`offset`
: The offset into the byte array.

`length`
: The number of bytes to stream.

Example

The following example streams all of byte array `b` to the ICS context:

```
ics.StreamBinary(b, 0, b.length)
```

StreamEvalBytes

Writes a string to the current element evaluation stream.

Syntax

```
public void
StreamEvalBytes(String str)
```

Parameters

`str`

String to write to current element evaluation stream.

Description

The `StreamEvalBytes` method writes a string to the current element evaluation stream. It will be appended to the current element output.

StreamHeader

Streams header data immediately.

Syntax

```
public void
StreamHeader(java.lang.String header,
             java.lang.String text)
```

Parameters

`header`
: Content type of the header data.

`text`
: The header data.

Description

The `StreamHeader` method streams header data immediately. Once non-header data has been sent, you can no longer use this method.

StreamText

Use this method to immediately stream text out to the ICS context.

Syntax

```
public void
StreamText(String text)
```

Parameters

`text`
: The text to stream.

synchDebug

Synch debug flag.

Syntax

```
public boolean
synchDebug()
```

Returns

Returns `true` if synch debugging is enabled, `false` otherwise.

systemDebug

Determines whether debugging is enabled on the Content Server system.

Syntax

```
public boolean
systemDebug()
```

Description

The `systemDebug` method indicates whether debugging is enabled on the Content Server system. `systemDebug` corresponds to the `ft.debug` property.

Returns

Returns `true` if debugging is enabled, `false` otherwise.

systemSession

Determines whether sessions are enabled on the Content Server system.

Syntax

```
public boolean
systemSession()
```

Returns

Returns `true` if sessions are enabled, `false` otherwise.

ThrowException

The `ThrowException` method halts further processing of the current element.

Syntax

```
public void
ThrowException()
```

Description

The `ThrowException` method is the interface to the functionality of the `THROWEXCEPTION` XML tag. The `ThrowException` method halts further processing of the current thread. Text streamed prior to the `ThrowException` method appears in the output.

timeDebug

The time debug flag.

Syntax

```
public boolean
timeDebug()
```

Returns

Returns `true` if debugging is enabled, `false` otherwise.

TreeManager

The `TreeManager` method executes a `TreeManager` command. This method has two variants:

- `TreeManager (Variant 1)` executes `TreeManager` commands.
- `TreeManager (Variant 2)` executes `TreeManager` commands and accepts a handle for the `BatchContext` object.

For more information on the various `TreeManager` commands, see the TreeManager command in the *CSEE Developer's Tag Reference*.

Several `TREEMANAGER` commands (`addchild`, `addchildren`, `findnode`, `getchildren`, `getnode`, `getparent`, `listtrees`, and `nodepath`) return a list. Each of those commands has a `treename` argument. The name of the returned list is the value passed as the treename argument.

The following columns are common to every list returned by a `TreeManager` command:

Column Name	What it Holds
`nid`	Tree node ID.
`nparentid`	Parent tree node ID. 0 (zero) means it is a root node.
`nrank`	Application-specific rank of the node.
`otype`	Object table referenced by this node.
`oid`	Object ID referenced by this node.
`oversion`	Reserved for future use by FatWire.
`ncode`	Application-specific code related to the node.

TreeManager (Variant 1)

Use this method to execute a TreeManager command. A list of available commands follows:

`addchild`	`delchildren`	`getnode`	`nodepath`
`addchildren`	`deletetree`	`getparent`	`setobject`
`copychild`	`findnode`	`listtrees`	`validatenode`
`createtree`	`getchildren`	`movechild`	`verifypath`
`delchild`			

For more information on the various TreeManager commands, see the TreeManager command in the *CSEE Developer's Tag Reference*.

Syntax

```
public boolean
TreeManager(FTValList vIn)
```

Parameters

`vIn`

List of name/value pairs passed to TreeManager. The `ftcmd` parameter is required and specifies the specific TreeManager command to execute. Any other parameters depend on the TreeManager command. A `null` indicates using existing variables.

Returns

Returns `true` if TreeManager was invoked successfully.

errno

Use `GetErrno()` to check the results of the specific TreeManager command.

Example

```
// add node
ics.ClearErrno();
inList.removeAll();
inList.setValString("ftcmd", "addchild");
inList.setValString("treename", treeTable);
inList.setValString(Msg.parentnode, node);
inList.setValString(Msg.classType, topicTable);
inList.setValString(Msg.classId, newId);
ics.TreeManager(inList);
ics.GetErrno();
```

TreeManager (Variant 2)

Use this method to execute a TreeManager command.

Syntax

```
public boolean
TreeManager(FTValList vIn,
            Object oBatchContext)
```

Parameters

`vIn`

List of name/value pairs passed to TreeManager. The `ftcmd` parameter is required and specifies the specific TreeManager command to execute. Any other parameters depend on the `TreeManager` command. A `null` indicates using existing variables.

`oBatchContext`

A handle for the `BatchContext` object.

Description

Use this method to execute a TreeManager command. This variant accepts a `BatchContext` object. The following lists the TreeManager commands that accept `BatchContext` objects:

- `addchild`
- `addchildren`
- `deletechild`
- `deletechildren`

Returns

Returns `true` if TreeManager was invoked successfully.

errno

Use `GetErrno()` to check the results of the specific TreeManager command.

UserIsMember

Tests to see if the current user is a member of one or more access groups (ACLs). `Variables.errno` is set to 1 (one) if the user is a member of any of the listed groups.

Syntax

```
public boolean
UserIsMember(String grouplist)
```

Parameters

`grouplist`
: A comma-delimited list of groups to check against.

Returns

Returns `true` if the user is a member, `false` if not.

Example

The following code determines if the current user is a member of the `ElementEditor`, `Browser`, or `SiteGod` ACLs:

```
boolean member =
ics.UserIsMember("ElementEditor,Browser,SiteGod");
if (member)
  // Yes, this user is a member of one of these ACLs
else
  // This user is not a member of any of these ACLs
```

xmlDebug

Determines whether XML debugging is enabled.

Syntax

```
public boolean
xmlDebug()
```

Returns

Returns `true` if debugging is enabled, `false` otherwise.

Chapter 7

IDynamicTag

`IDynamicTag` is an interface for classes implementing dynamic tags. It extends the `Seed` class.

A dynamic tag is another name for a custom tag; that is, an XML tag that you (the application programmer) create.

List of Methods

Table 5: List of IDynamicTag Methods

Method	Summary
`endTag`	Evaluates the end tag of a dynamic tag.
`isCaseSensitive`	Checks whether the tag name and attributes are case-sensitive.
`setParent`	Sets a tag's parent class.
`startTag`	Evaluates the start tag of a dynamic tag.

endTag

Evaluates the end tag of a dynamic tag.

Syntax

```
public int
endTag(ICS ics,
       String sTagName,
       FTValList vIn)
```

Parameters

`ics`
A handle to an ICS object.

`sTagName`
The name of the tag you want to evaluate.

`vIn`
An FTValList of input attributes.

Returns

Returns one of the following integer constants:

`EVALUATE_BODY`
To continue body and end tag evaluation.

`EVALUATE_COMPLETE`
To complete tag evaluation.

isCaseSensitive

Checks whether the tag name and attributes are case-sensitive.

Syntax

```
public boolean
isCaseSensitive()
```

Returns

Returns `false` if the tag name and attributes must be lowercase.

setParent

Sets a tag’s parent class.

Syntax

```
public void
setParent(IDynamicTag oIDynamicTag)
```

Parameters

`oIdynamicTag`
The tag’s parent class.

startTag

Evaluates the start tag of a dynamic tag.

Syntax

```
public int
startTag(ICS ics,
         String sTagName,
         FTValList vIn,
         String sCData)
```

Parameters

`ics`
: A handle to an ICS object.

`sTagName`
: The name of the tag to evaluate.

`vIn`
: An FTValList containing input attributes.

`sCData`
: Compressed child tags.

Returns

Returns one of the following integer constants:

`EVALUATE_BODY`
: To continue body and end tag evaluation.

`EVALUATE_COMPLETE`
: To complete tag evaluation.

Chapter 8

IJSPObject

The `IJSPObject` interface allows you to interact with an `IJSPObject`. It contains the following methods:

Table 6: List of IJSPObject Methods

Method	Summary
`getErrorMsg`	Returns the last detected error on this `IJSPObject`.
`release`	Deletes the JSP file associated with this object.
`reloadIfChanged`	Reloads the specified JSP object if the source bits have changed.
`run`	Runs the specified `IJSPObject`.

getErrorMsg

Returns the last detected error on this `IJSPObject`.

Syntax

```
public String
getErrorMsg()
```

Returns

A string containing the error message.

release

Deletes the JSP file associated with this object.

Syntax

```
public void
release()
```

reloadIfChanged

Reloads the specified JSP object if the source bits have changed.

Syntax

```
public boolean
reloadIfChanged()
```

Returns

Returns `true` if successful.

run

Runs the specified `IJSPObject`.

Syntax

```
public int
run(ICS ics,
    StringBuffer sbError)
```

Parameters

`ics`
A handle to an ICS object.

`sbError`
For error codes.

Returns

Returns zero (0) if successful, an ICS error code (errno) on error.

Chapter 9

IList

The `IList` interface describes the methods available from a Java class when accessing an Content Server query or list object. It also defines the methods that a third party must implement when attempting to construct and register a list object for use within an Content Server XML page.

Each element of the list consists of an ordered set of named columns, numbered from `0` (zero) to `n-1`, where `n` is the number of columns. The order of elements in the list does not correspond to the order of the select list expressions in a SQL query, if the `IList` object is returned as a result of a SQL query execution.

Note

In an `IList`, rows are numbered from `1` to `M`; columns are numbered from `0` to `N-1`.

List of Methods

Table 7: List of IList Methods

Method	Summary
`atEnd`	Determines whether a list has reached the last row (at either end) of data.
`clone`	Shallow copies this list object.
`currentRow`	Returns the row number of the current row.
`flush`	Flushes a list object, destroying any backing data store.
`getColumnName`	Returns the name of a column at a specified offset.

Method	Summary
getFileData	Returns the data contained in a referenced data file.
getFileString	Returns the string data contained in a referenced data file.
getIndirectColumnName	Supplies the upload column name by index.
getName	Retrieves the name of this list.
getObject	Gets an object from the specified column.
getValue	Retrieves the value at the named column of the current row.
hasData	Checks whether this list contains any data.
moveTo	Changes the “current row” to the specified row number.
moveToRow	Changes the “current row” to the specified row position.
numColumns	Gets the number of columns in this list.
numIndirectColumns	Gets the number of columns that are indirect data pointers.
numRows	Gets the number of rows in this list.
rename	Renames this list.
stringInList	Looks for a string within this list.

atEnd

Determines whether a list has reached the last row (at either end) of this list.

Syntax

```
public boolean
atEnd()
```

Description

The `atEnd` method determines whether a list has reached the last row (at either end) of this list.

`atEnd()` is only meaningful if `hasData()` is `true` for the list.

Returns

The returned boolean value depends on the method of navigation and the value of the "done" state if the move succeeds:

Just After You Do This...	atEnd Returns This...
Create the list.	`false`
Call `moveToRow(IList.first,0)`	`false`
Call `moveToRow(IList.last,0)`	`true`
Call `moveToRow(IList.prev,0)`	`true` if the current row was 1 before the move; `false` otherwise.
Call `moveToRow(IList.next,0)`	`true` if current row was the last before the move; unchanged otherwise.
Call `moveTo(n)` or Call `moveToRow(IList.gotorow,n)`	`false` if the `IList` represents an uncached query object. Otherwise, if n is the last row, then `true`, else `false`.

See Also

`hasData`

clone

Shallow copies this list object.

Syntax

```
public IList
clone(String newname)
```

Parameters

`newname`
The name of the copy.

Description

The `clone` method shallow copies a list object.

A user-defined implementation may decide to provide a deep copy.

Returns

Returns null from a user defined list object if support is not implemented or desired.

currentRow

Returns the row number of the current row.

Syntax

```
public int
currentRow()
```

Description

The `currentRow` method returns the number of the current row. Rows are numbered from `1` to `M`, with `M` representing the total number of rows in the specified list.

Returns

Returns the current row number or `-1` if there is no current row. For example, if `hasData` is `false`, there would be no current row, so `currentRow` would return `-1`.

Example

The following code accesses the current row while iterating through a list:

```
String title;
String rating;
String year;
int row;

do {
   row = MatchingMovies.currentRow();
   rating = MatchingMovies.getValue("rating");
   title = MatchingMovies.getValue("title");
   year = MatchingMovies.getValue("year");

         // Process row, rating, title and year here.

}
while (MatchingMovies.moveToRow(IList.next, 0) );
```

See Also

hasData

flush

Flushes a list object, destroying any backing data store.

Syntax

```
public void
flush()
```

Description

The `flush` method flushes a list object, destroying any backing data store.

Example

The following code creates a list object named `mylist` and then calls the `flush` method to destroy it.

```
// Create a list named mylist
...

// Evaluate mylist
...

// When finished using mylist, flush it
mylist.flush();
```

getColumnName

Returns the name of the column at a specified offset.

Syntax

```
public String
getColumnName(int i)
  throws ArrayIndexOutOfBoundsException
```

Parameters

`i`

Index number of a column in the list.

Description

The `getColumnName` method returns the name of the specified column. Columns begin numbering from zero.

Returns

Returns a string containing the column name.

Throws

`ArrayIndexOutOfBoundsException`

If you specify a value of `i` that falls outside the range `0` to `n-1`, where `n` is the number of columns.

Example

The following code fragment accesses all the column names in the `MatchingMovies` list:

```
int NumberOfColumns = MatchingMovies.numColumns();
int c;
String ColumnName;

for (c = 0; c < NumberOfColumns; c++) {
   ColumnName = MatchingMovies.getColumnName(c);

   // Do something with ColumnName
   ...
}
```

getFileData

Returns the data contained in a referenced data file.

Syntax

```
public byte[]
getFileData(String columname)
  throws IllegalArgumentException, NoSuchFieldException
```

Parameters

`colname`
The name of the column to get the data from.

Description

The `getFileData` method returns the data contained in a referenced file. Note that this kind of data retrieval will not be cached; each request will reread the bytes. An exception is thrown if the column is not an indirect data type or if other errors occur.

Returns

Returns the `byte[]` value, which can be null.

Throws

`IllegalArgumentException`, `NoSuchFieldException`

getFileString

Returns the string data contained in a referenced data file.

Syntax

```
public String
getFileString(String columname)
  throws NoSuchFieldException
```

Parameters

`columnname`
: The name of the column.

Description

The `getFileString` method returns the string data contained in the referenced file.

Returns

Returns the string value, which can be null.

Throws

`NoSuchFieldException`
: If you specified a column name that does not belong to this row.

getIndirectColumnName

Supplies the upload column name by index.

Syntax

```
public String
getIndirectColumnName(int index)
```

Parameters

`index`

Index number of an indirect column in the list of indirect columns. The index number of the first indirect column is 0 (not 1).

Returns

Name of the indirect column, preceded by an @ sign.

Normally, a user-defined `IList` should return null.

See Also

`getFileData(), getFileString(), numIndirectColumns()`

getName

Retrieves the name of this list.

Syntax

```
public String
getName()
```

Description

The `getName` method retrieves the name of this list. This name reflects the name by which Content Server knows this list.

Content Server sets the name when it creates the list or modifies the list with ICS methods `ICS.RenameList()` or `ICS.RenameList()`. These methods use `rename()` to change the internal name so that `getName()` will reflect the proper name.

Returns

Returns the name of the list.

See Also

`RegisterList, rename, RenameList, GetList`

getObject

Gets an object from the specified column.

Syntax

```
public Object
getObject(String colname)
     throws NoSuchFieldException
```

Parameters

`colname`
The name of the column to get the object from.

Description

The `getObject` method gets an object from a column.

This method is similar to `getValue` but can return list-specific data. For example, if the list represents a database query, the object returned will be null unless the column represents a byte array. Other lists may return particular types.

Returns

Returns the object from the column.

Throws

`NoSuchFieldException`
When you specify a `colname` that does not exist in the list.

getValue

Retrieves the value at the named column of the current row.

Syntax

```
public String
getValue(String columname)
  throws NoSuchFieldException
```

Parameters

`columnname`
: The name of the column to get the value from.

Returns

Returns a string containing the value of the column for the current row. May be null or empty, depending on the data type.

Throws

`NoSuchFieldException`
: When you specify a `colname` that does not exist in the list.

hasData

Checks whether this list contains any data.

Syntax

```
public boolean
hasData()
```

Description

The `hasData` method checks whether a list has any data. To avoid any potential internal state inconsistencies, you should always check the list to determine if it contains any data prior to manipulating it.

Returns

Returns `true` if the list contains data, `false` if it is empty.

Example

```
String apath = node;
IList ilist = ics.GetList("TreeManagerTest");
if(ilist.hasData())
{
    int r = ilist.numRows();
    String values;
    for(int i = 1; i <= r; i++)
    {
        try{
            ilist.moveTo(i);
            values = ilist.getValue("nid");
            apath = apath+":"+values;
        } catch(NoSuchFieldException e){System.out.println(e);};
    }
}
return apath;
```

moveTo

Changes the “current row” to the specified row number.

Syntax

```
public boolean
moveTo(int i)
```

Parameters

`i`

The row number to move to, starting from 1.

Returns

Returns `true` on success, `false` on failure. Success does not necessarily imply an actual move. For example, a `moveTo()` the current row will succeed even though it does not actually move.

Example

The following code recursively deletes the given node and its children nodes and their associated objects:

```
public boolean DeleteTopic(String delnode)
{
    boolean retval = true;
    IList treelist;
    FTValList inList = new FTValList();
    ics.ClearErrno();
    inList.setValString("ftcmd", "getchildren");
    inList.setValString("treename", treeTable);
    inList.setValString(Msg.parentnode, delnode);
    inList.setValString("join", "false");
    ics.TreeManager(inList);
    treelist = ics.GetList(treeTable);
    String errno = ics.GetVar("errno");
    try
        {
            for (int i=1; i <= treelist.numRows(); i++)
                {
                    treelist.moveTo(i);
                    DeleteTopic(treelist.getValue(Msg.nid));
                }
        }
    catch (NoSuchFieldException e){
                retval = false;
        }
    if (retval)
        retval = DeleteNode(delnode);
    return retval;
}
```

See Also

atEnd, moveToRow

moveToRow

Changes the “current row” to the specified row position.

Syntax

```
public boolean
moveToRow(int how,
          int v)
```

Parameters

`how`

The method of moving to a row:

`IList.first` - Move to the first row in the list.

`IList.last` - Move to the last row in the list.

`IList.next` - Move to the next row in the list.

`IList.prev` - Move to the previous row in the list.

`IList.gotorow` - Move to the row specified by `v`.

`v`

If you set `how` to `IList.gotorow`, then set `v` to the specific row number. The first row is `1` (not `0`). If you set `how` to something other than `IList.gotorow`, then `moveToRow` ignores the value of `v`.

Returns

Returns `true` on success, `false` on failure.

Example

The following code walks through each row in a list:

```
String rating;
String title;
String year;

do {
      rating = MatchingMovies.getValue(“rating”);
      title = MatchingMovies.getValue(“title”);
      year = MatchingMovies.getValue(“year”);
 }
while (MatchingMovies.moveToRow(IList.next, 0) );
```

See Also

`atEnd`

numColumns

Gets the number of columns in this list.

Syntax

```
public int
numColumns()
```

Returns

Returns the number of columns in this list.

Example

See the example for `getColumnName`.

numIndirectColumns

Gets the number of columns that are indirect data pointers.

Syntax

```
public int
numIndirectColumns()
```

Returns

Number of columns that are indirect data pointers. The default implementation for a user-defined `IList` should return 0.

See Also

`getIndirectColumnName()`

numRows

Gets the number of rows in this list.

Syntax

```
public int
numRows()
```

Description

The `numRows` method returns the number of rows in a list.

If the list represents uncached data in a database, a large delay and high memory usage might be encountered, since the list must be scanned **on the database side** to determine the count. Avoid this if possible.

Returns

Returns the number of rows in the list.

Example

```
try
    {
        for (int i=1; i <= treelist.numRows(); i++)
            {
                treelist.moveTo(i);
                DeleteTopic(treelist.getValue(Msg.nid));
            }
    }
```

rename

Renames this list.

Syntax

```
public void
rename(String newname)
```

Parameters

`newname`
The name of the new list.

Description

The `rename` method renames a list. Both `RenameList()` and the `RENAMELIST` tag call `rename`. Using `rename` might change the name of a list from what Content Server is using. If renaming is not desired, the `rename` method will not alter the list name. If you want to rename a list, use an implementation similar to:

```
public void rename(String newname) {mName=newname;}
```

Example

```
dataList.rename("renamelist");
```

See Also

`RenameList()`

stringInList

Looks for a string within this list.

Syntax

```
public boolean
stringInList(String item)
```

Parameters

`item`
: String to search for in list

Description

The `stringInList` method looks for a string within a list.

For one-column lists, it can be implemented to determine whether a given value is in any row. This method only works on `StringList` and `FileList` objects.

Returns

Returns `true` if string is in list. Returns `false` if the string is not in the list, or if an `IList` implementation does not support this method.

Example

```
IList listA = ics.GetList("myList”);
boolean inList = listA.stringInList("this string");
if (!inList)
{
   //the string you supplied is not in the list
   //or the IList implementation does not support this method.
}
```

Chapter 10

IManageUsers (Deprecated)

As of CSEE 4.0, the `IManageUsers` interface has been deprecated. CSEE 4.0 and later do not provide a replacement Java interface or class; use the new XML `DIR` tag family instead.

The `IManageUsers` interface extends the `IUsers` interface. It specifies the functionality needed to implement the ValidateLogin framework component used to manage user information. The functions provided by the ValidateLogin framework are accessed through the CatalogManager and therefore takes its arguments in an FTValList. Some of these values are required by the ValidateLogin framework and are extracted there. The rest are passed on to these methods in the FTValList.

List of Methods

AddUser	DeleteUser	ModifyUser

AddUser

This method is deprecated as of CSEE Version 4.0. There are currently no Java methods to replace this method; however, you can use the `DIR.CREATE` XML tag instead.

Adds a new user to the directory.

Syntax

```
public void
AddUser(String username,
        FTValList valIn,
        FTValList valOut)
throws UserError
```

Parameters

`username`
: String specifying the name of the new user.

`vIn`
: List of name/value pairs that provide any other information required by the directory service that is used.

`vOut`
: Name/value pairs that might be set by the method that appears as variables upon return to ContentServer.

Throws

`UserError`
: If there is a problem adding this user.

See Also

`FTValList`

DeleteUser

This method is deprecated as of CSEE Version 4.0. There are currently no Java methods to replace this method; however, you can use the `DIR.DELETE` XML tag instead.

Deletes an existing user from the directory.

Syntax

```
public void
DeleteUser(String username,
           FTValList valIn,
           FTValList valOut)
throws UserError
```

Parameters

`username`
: String specifying the name of the user.

`vIn`
: List of name/value pairs that provide any other information required by the directory service that is used.

`vOut`
: Name/value pairs that might be set by the method. These values appear as variables upon return to ContentServer.

Throws

`UserError`
: If there is a problem adding this user.

See Also

`FTValList`

ModifyUser

This method is deprecated as of CSEE Version 4.0. There are currently no Java methods to replace this method; however, you can use the `DIR` XML tags instead.

Modifies an existing user's directory entry.

Syntax

```
public void
ModifyUser(String username,
           FTValList valIn,
           FTValList valOut)
 throws UserError
```

Parameters

`username`
: String specifying the name of the user.

`vIn`
: List of name/value pairs that provide any other information required by the directory service that is used.

`vOut`
: Name/value pairs that might be set by the method. These values appear as variables upon return to ContentServer.

Throws

`UserError`
: If there is a problem adding this user.

See Also

`FTValList`

Chapter 11

IServlet

The `IServlet` interface provides access to the J2EE components that Content Server runs on top of.

List of Methods

Table 8: List of IServlet Methods

Method	Summary
`getServlet`	Gets the `HttpServlet` from the current thread.
`getServletRequest`	Gets the `HttpServletRequest` from the current thread.
`getServletResponse`	Gets the `HttpServletResponse` from the current thread.

getServlet

Gets the `HttpServlet` from the current thread.

Syntax

```
public COM.FutureTense.Interfaces.HttpServlet
getServlet()
```

Returns

The `HttpServlet`.

getServletRequest

Gets the `HttpServletRequest` from the current thread.

Syntax

```
public COM.FutureTense.Interfaces.HttpServletRequest
getServletRequest()
```

Returns

The `HttpServletRequest`.

getServletResponse

Gets the `HttpServletResponse` from the current thread.

Syntax

```
public COM.FutureTense.Interfaces.HttpServletResponse
getServletResponse()
```

Returns

The `HttpServletResponse`.

Chapter 12

ISynchHash

The `ISynchHash` object has functionality similar to a hash table, but with cluster synchronization and LRU behaviors.

Note that the use of this object has performance implications. Before you use `ISynchHash`, consider the performance cost of reloading data compared to the memory usage per JVM. Use cluster synchronization only when a backing store is likely to change and invalidate stored results in a given VM.

Operational considerations (from Sun Microsystems) suggest that data read vs. write from a synchronized hash should be in an 80/20 ratio if the synchronization is across multiple machines. Non-synchronized hashes do not have this suggested restriction.

List of Methods

Table 9: List of ISynchHash Methods

Method	Summary
`clear`	Clears a hash and synchronizes the cluster.
`get`	Gets the specified object from hash and performs required data validation/VM synchronization checks as needed.
`getName`	Gets the name of the hash as it is managed.
`put`	Adds an object to the hash and generates appropriate cluster synchronization methods and least-recently-used behavior.
`remove`	Removes an item from the hash and synchronizes the cluster as needed.

clear

Clears a hash and synchronizes the cluster.

Syntax

```
public void
clear()
```

Description

The `clear` method clears a hash and synchronizes the cluster as necessary.

Note that it is more efficient to use the `clear` method to remove a hash then to remove all entries one by one. When removing entries one-by-one, although the first iteration of the loop has little performance cost, subsequent iterations have the additional performance cost of cluster synchronization, which can be very high.

get

Gets the specified object from hash and performs required data validation/VM synchronization checks as needed.

Syntax

```
public Object
get(String key)
```

Parameters

`key`
: The key of the object you want to get.

Returns

The specified object.

getName

Gets the name of the hash as it is managed.

Syntax

```
public String
getName()
```

Description

The `getName` method gets the name of the hash as it is managed. Note that there is no namespace support for hash names.

Returns

The name of the hash.

put

Adds an object to the hash and generates appropriate cluster synchronization methods and least-recently-used behavior.

Syntax

```
public Object
put(String key,
    Object v)
  throws java.lang.NullPointerException
```

Parameters

`key`

A key for the object you want to add. Use meaningful, reproducible key names for proper behavior. Limit key length to 64 characters or less and do not use multi-K length keys. Sixteen characters is the optimal length for performance, but the keys may not be unique.

`v`

The object to add.

Returns

Returns the previous object or null. Note that the method may throw exception(s) if the create operation fails.

Throws

`java.lang.NullPointerException`

remove

Removes an item from the hash and synchronizes the cluster as needed.

Syntax

```
public Object
remove(String key)
```

Parameters

`key`
: The key of the object to remove.

Description

The `remove` method removes an item from the hash and synchronizes the cluster as needed. Note that it is more efficient to use the `clear()` method to remove a hash then to remove all entries one by one; the first iteration of the loop has little performance cost, but subsequent iterations have the additional performance cost of cluster synchronization, which can be very high.

Returns

Returns the object; `null` if not found.

Chapter 13

UserError

Use this class to generate `UserError` conditions that may occur in the `IUsers` context. It may be extended to add new error conditions that are specific to an `IUsers` implementation.

List of Methods

Table 10: List of UserError Methods

INIFILE_BADPROPERTY	INIFILE_NOTFOUND	INIFILE_NOTSUPPLIED
INIFILE_READ	UNKNOWNCOMMAND	UNSUPPORTEDACTION
USEREXCEPTION	USERMANAGER	

INIFILE_BADPROPERTY

Generates the appropriate `UserError` to report a bad value for the specified `.ini` file property.

Syntax

```
static final UserError
INIFILE_BADPROPERTY(String property,
                    String value)
```

Parameters

`property`
: String containing the name of the property.

`value`
: String containing the bad value found for that property.

Returns

An instance of `UserError`.

See Also

`INIFILE_NOTFOUND, INIFILE_NOTSUPPLIED, INIFILE_READ`

INIFILE_NOTFOUND

Generates the appropriate `UserError` to report that a specified `.ini` file was not found.

Syntax

```
static final UserError
INIFILE_NOTFOUND(String inifile)
```

Parameters

`inifile`
: String containing the name of the `.ini` file that was requested.

Returns

An instance of UserError.

See Also

`INIFILE_BADPROPERTY, INIFILE_NOTSUPPLIED, INIFILE_READ`

INIFILE_NOTSUPPLIED

Generates the appropriate UserError to report that an `.ini` file was not supplied by ContentServer.

Syntax

```
public static final UserError
INIFILE_NOTSUPPLIED()
```

Returns

An instance of UserError.

See Also

INIFILE_NOTFOUND, INIFILE_READ

INIFILE_READ

Generates the appropriate UserError to report that a problem occurred reading the specified `.ini` file.

Syntax

```
public static final UserError
INIFILE_READ(String inifile)
```

Parameters

`inifile`
: String containing the name of the .ini file that had the read problem.

Returns

An instance of UserError.

See Also

`INIFILE_BADPROPERTY, INIFILE_NOTSUPPLIED, INIFILE_NOTFOUND`

UNKNOWNCOMMAND

Generates the appropriate UserError to report that a bad command was specified.

Syntax

```
public static final UserError
UNKNOWNCOMMAND(String cmd)
```

Parameters

`cmd`
: String containing the specific bad command.

Description

The `UNKNOWNCOMMAND` method generates the appropriate UserError to report that a bad command was specified. This method is thrown in the ValidateLogin framework and is probably not needed in a specific implementation.

Returns

An instance of UserError.

See Also

`UNSUPPORTEDACTION`

UNSUPPORTEDACTION

Generates the appropriate UserError to report that an unsupported action was not performed.

Syntax

```
public static final UserError
UNSUPPORTEDACTION(String cmd)
```

Parameters

`cmd`

String containing the specific unsupported command.

Description

The `UNSUPPORTEDADCTION` method generates the appropriate UserError to report that an unsupported action was not performed. This method is thrown in the ValidateLogin framework and is probably not needed in a specific implementation.

Returns

An instance of UserError.

See Also

`UNKNOWNCOMMAND`

USEREXCEPTION

Generates the appropriate UserError to report that a generic exception was thrown.

Syntax

```
public static final UserError
USEREXCEPTION()
```

Description

The `USEREXCEPTION` method generates the appropriate UserError to report that a generic exception was thrown. This generic error condition may also be used if the IUsers implementation does not extend UserError to generate exceptions specific to that implementation.

Returns

An instance of UserError.

USERMANAGER

Reports that an `IUsers` object was not created.

Syntax

```
public static final UserError
USERMANAGER()
```

Description

The `USERMANAGER` method reports that an `IUsers` object was not created. This exception is internal to the `ValidateLogin` framework.

Returns

An instance of UserError.

Chapter 14

IUsers

The `IUsers` interface specifies the methods required for the minimal `ValidateLogin` functionality. The functions provided by the `ValidateLogin` framework are accessed through the CatalogManager and therefore takes its arguments in an `FTValList`. Some of these values are required by the `ValidateLogin` framework and are extracted there. The rest are passed on to these methods in the `FTValList`.

List of Methods

Table 11: List of IUsers Methods

Method	Summary
`FlushCache`	Notifies the `IUser` implementation that the `ValidateUser` framework has flushed any user information that was cached.
`GetACLs`	Determines the list of groups or ACLs in which the user has membership.
`Login`	Authorizes a user within an application.
`Logout`	Logs out the specified user.

FlushCache

Notifies the `IUser` implementation that the `ValidateUser` framework has flushed any user information that was cached.

Syntax

```
public void
FlushCache()
  throws UserErrorException
```

Description

The `FlushCache` method notifies the `IUser` implementation that the `ValidateUser` framework has flushed any user information that was cached. It is used internally by the `ValidateLogin` framework, so it is required; however, it need not do much, if anything.

Throws

`UserErrorException`
If there is a problem.

GetACLs

Determines the list of groups or ACLs in which the user has membership.

Syntax

```
public String
GetACLs(String userid)
  throws UserError
```

Parameters

`userid`
String containing the user's ID that is the result returned by the Login

Returns

String containing a comma-separated list of the groups or ACLs. This could be an empty string if there are no groups.

Throws

`UserError`
If there is a problem accessing ACL information for this user.

Login

Authorizes a user within an application.

Syntax

```
public String
Login(String username,
      String password,
      FTValList vIn,
      FTValList vOut)
  throws UserError
```

Parameters

`username`
String specifying the name of an existing user.

`password`
String containing the user's password.

`vIn`
List of name/value pairs that provide any other arguments that might be required to login a user.

`vOut`
Name/value pairs that might be set by the method. These values appear as variables upon return to Content Server.

Returns

String containing the user's ID. Throws a UserError exception if there is a problem.

See Also

`GetACLs`, `Login`

Logout

Logs out the specified user.

Syntax

```
public void
Logout(String username,
       FTValList vIn,
       FTValList vOut)
 throws UserError
```

Parameters

`username`
String specifying the name of an existing user.

`vIn`
List of name/value pairs that provide any other arguments that might be required.

`vOut`
Name/Value pairs that might be set by the method that appears as variables upon return to Content Server.

Description

The `Logout` method logs the specified user out of an application.

Throws

`UserError`
If there is a problem logging out the specified user.

See Also

`Login`

Chapter 15

Utilities

The `Utilities` class extends `Object`. The utility methods provide an easy-to-use interface for some commonly needed functionality. Most of the methods are `static`.

List of Methods

Dates

`calendarFromJDBCString` `jdbcDateFromCal` `jdbcTimeFrom`
`sqlDate`

Email

`getPOP3Messages` `sendMail`

Encryption

`cryptoEnabled` `decryptString` `encryptString`

Files and Directories

`appendFile` `createTempFile` `deleteFile`
`directoryList` `emptyFolder` `ensureFolder`
`fileExists` `fileFromSpec` `fileLockLoadOK`
`isFile` `isFolder` `lockFile`
`osIgnoreFileCase` `readByteFile` `readFile`

unlockFile unzipFile writeFile

Strings and Vectors

deBabble decryptString encryptString
goodString itemIsInList replaceAll
sortWords sortWordsAlpha stringFromValList
words

URLs

readByteURL readURL

Miscellaneous

genID getParams keysFromMap
osSafeSpec pathFromSpec validateZip

appendFile

The `appendFile` method appends data to a disk file. This method has two variants:

- `appendFile (Variant 1)` appends a string to a disk file.
- `appendFile (Variant 2)` appends an array of bytes to a disk file.

---

appendFile (Variant 1)

Appends a string to a disk file.

Syntax

```
public static final boolean
appendFile(String fname,
           String str)
```

Parameters

`fname`
: The path and file name of the file to append to.

`str`
: The file data to append, in string form.

Description

The `appendFile` method appends a string to a disk file.

Returns

Returns `true` on success, `false` on failure.

See Also

`osSafeSpec`

---

appendFile (Variant 2)

Appends an array of bytes to a disk file.

Syntax

```
public static final boolean
appendFile(String fname,
           byte[] bits,
           int size)
```

Parameters

`fname`
: The path and file name of the file to append.

`bits`
: The file data to append, in a byte array.

`size`
The number of bytes to append.

Description

The `appendFile` method appends an array of bytes to a disk file. The file name is internally converted by `osSafeSpec()`.

Returns

Returns `true` on success, `false` on failure.

See Also

`osSafeSpec`

calendarFromJDBCString

Returns a calendar object from JDBC date/time.

Syntax

```
public static Calendar
calendarFromJDBCString(String str)
```

Parameters

`str`
String in the form "YYYY-MM-DD HH:MM:SS". HH is in 24-hour format.

Returns

Returns the appropriate calendar object.

Example

The following method assigns the date (August 31, 2002) and time (one second before midnight) to a `Calendar` object named `cal`:

```
Calendar cal =
   Utilities.calendarFromJDBCString( "2002-08-31 23:59:59" );
```

See Also

`jdbcDateFromCal, jdbcTimeFrom`

createTempFile

Creates a temporary file.

Syntax

```
public static final File
createTempFile(boolean bManageDelete)
```

Parameters

`bManageDelete`

Specifying `false` tells the method to delete the file as soon as the process exits. Specifying `true` tells the method to take no action on the temporary file. If you specify `true`, then your code can decide to delete the file later on, if necessary.

Returns

A `File` object that represents the temporary file.

cryptoEnabled

Checks if encryption is enabled.

Syntax

```
public static final boolean
cryptoEnabled()
```

Description

The `cryptoEnabled` method checks whether encryption is enabled. This requires that the encryption package be installed. `cryptoEnabled` should always return `true`, since an encryption package is included as part of ContentServer.

Returns

Returns `true` if encryption is enabled, `false` otherwise.

Example

The following code calls `cryptoEnabled` to ensure that encryption is enabled prior to encrypting a string.

```
if (Utilities.cryptoEnabled())
{
    String clear_text = "Attack at dawn.";
    String encrypted_text = Utilities.encryptString(clear_text);
    String decrypted_text =
                        Utilities.decryptString(encrypted_text);
    if( !decrypted_text.equals(clear_text) )
    {
        // something turned out wrong
    }
}
```

See Also

`decryptString`

deBabble

Replaces the character encoding strings—%20 (space), %22 ("), %12 (\n), %3c (<), %3e (>), and %26 (&)—in an input string with the characters they represent.

Syntax

```
static final String
deBabble(String str)
```

Parameters

`str`
: String to convert.

Description

Certain reserved characters (such as <) can be encoded into their hexadecimal equivalents (for example, `%3c`). The `deBabble` method replaces these hexadecimal equivalents with the actual character name; for example, `deBabble` converts `%3c` back into the less-than sign (`<`). The `deBabble` method can convert the following hexadecimal values:

Hexadecimal String	deBabble Replacement
`%20`	space
`%22`	double quote (")
`%25`	percent (%)
`%26`	ampersand (&)
`%0a`	line feed (\n)
`%09`	tab
`%0d`	carriage return (\r)
`%3c`	less than (<)

Returns

Returns the converted string, or the same string if no conversion was done.

Example

The following fragment converts a string containing some embedded hexadecimal characters:

```
String AStringContainingHex = "apples%26bananas%3coranges";
String deBabbledStr = Utilities.deBabble(AStringContainingHex);
```

After executing the preceding code, the `deBabbledStr` object should contain the following:

```
apples&bananas<oranges
```

decryptString

The decryptString method decrypts a string. This method has two variants:

- `decryptString (Variant 1)` decrypts a string with the default key using the DES algorithm.
- `decryptString (Variant 2)` decrypts a string with the key you used in `encryptString (Variant 2)`.

---

decryptString (Variant 1)

Decrypts a string with the default key using the DES algorithm.

Syntax

```
static String
decryptString(String s)
```

Parameters

`s`

The string to decrypt.

Returns

Returns the decrypted string.

Example

```
if (Utilities.cryptoEnabled())
{
  String clear_text = "Attack at dawn.";
  String encrypted_text = Utilities.encryptString( clear_text );
  String decrypted_text = Utilities.decryptString(encrypted_text);
}
```

See Also

`cryptoEnabled, encryptString`

---

decryptString (Variant 2)

Decrypts a string with the key you used in `encryptString (Variant 2)`.

Syntax

```
static String
decryptString(String s,
              byte [] key)
```

Parameters

`s`

The string to decrypt.

`key`
: A byte array containing byte values that represents the key used by the encryption algorithm. The array should be at least 8 bytes long.

Returns

Returns the decrypted string.

See Also

`cryptoEnabled, encryptString`

deleteFile

Deletes a file from disk.

Syntax

```
static final boolean
deleteFile(String fname)
```

Parameters

`fname`
: The full path name of the file to delete.

Description

The `deleteFile` method deletes a file from disk. The argument is first converted by `osSafeSpec`, which will fix the pathname separators, if necessary.

Returns

Returns `true` on success, `false` on failure. Possible reasons for failure include:

- The path name does not exist.
- The path name does exist, but you do not have permission to delete it.

Example

The following line deletes the file at Windows pathname `c:\temp\text.txt` even though the given pathname uses UNIX-style pathname separators.

```
Utilities.deleteFile( "c:/temp/test.txt" );
```

See Also

`osSafeSpec`

directoryList

Returns a vector of file names in a given directory that satisfy a specified filter pattern.

Syntax

```
public static Vector
directoryList(String path,
              String filter,
              boolean recurse,
              int maxcount)
```

Parameters

`path`
Path name of the directory to search.

`filter`
File filter, that typically includes wildcards. The two wildcards supported are:

- *, which matches any string (or no string).
- ?, which matches any single character.

For example, the filter `"*.bat"` would match any file ending in the suffix `.bat`. The filter `"???"` would match any file whose name contains exactly three characters. Setting `filter` to `null` tells `directoryList` to match all files in the specified path.

`recurse`
A `true` value tells the method to look for matching files (based on the filter) in the subdirectories of the supplied path. A `false` value tells the method to restrict the search to the directory specify in path.

`maxcount`
Specifies the maximum number of files that will be returned in the vector. It is possible, though rare, to generate `outofmemory` exceptions by specifying a very large number.

Description

The `directoryList` method returns a vector of file names in a given directory that satisfy a specified filter pattern. Both the directory path and the filter are converted using `osSafeSpec()`.

Returns

Returns the vector of file names in the path that match the filter.

Example

The following code finds all the Java source code files in directory `/users/bob` and all its subdirectories. Only the first 100 matching files will be returned:

```
String Directory = "/users/bob";
String Filter = "*.java";
boolean Recurse = true;
int Maxcount = 100;
Vector JavaSourceFiles = Utilities.directoryList(Directory,
                                    Filter, Recurse, Maxcount);
errNum = cs.GetErrno();
```

See Also

osSafeSpec

emptyFolder

The `emptyFolder` method deletes the contents of a folder. This method has two variants:

- `emptyFolder (Variant 1)` recursively deletes the contents within a folder, and then, optionally, the folder itself.
- `emptyFolder (Variant 2)` recursively deletes all files that match the chosen pattern from the specified directory tree.

---

emptyFolder (Variant 1)

Recursively deletes the contents within a folder, and then, optionally, the folder itself.

Syntax

```
public static final int
emptyFolder(String path,
            boolean bDelete)
```

Parameters

`path`
: The pathname of the directory whose contents are to be deleted.

`bDelete`
: Specify `true` to delete the folder itself. Specify `false` to suppress deleting the folder itself.

Returns

Returns zero (0) on success, or error code on error.

Example

The following code empties the contents of the `/tmp` directory and any of its subdirectories but does not delete the `/tmp` itself:

```
if ( Utilities.emptyFolder( "/tmp", false ) )
{
    // the tmp folder is now empty
}
```

---

emptyFolder (Variant 2)

Recursively deletes all files which match the chosen pattern from the specified directory tree.

Syntax

```
public static final int
emptyFolder(String path,
            String pattern)
```

Parameters

`path`
: The pathname of the directory whose contents are to be deleted.

`pattern`
: The pattern you want to find and delete.

Returns

Returns zero (0) on success, or error code on error.

encryptString

The `encryptString` method encrypts a string. This method has two variants:

- `encryptString (Variant 1)` encrypts a string using the DES algorithm and a default key.
- `encryptString (Variant 2)` encrypts a string using the key that you select.

---

encryptString (Variant 1)

Encrypts a string.

Syntax

```
public static final String
encryptString(String s)
```

Parameters

`s`

The string to encrypt.

Description

The `encryptString` method encrypts a string. The encryption method uses the DES algorithm and a default key.

Returns

Returns the encrypted string. If encryption is not enabled, it returns the original string.

Example

The following code encrypts the string "`Attack at dawn`":

```
if (Utilities.cryptoEnabled())
{
  String clear_text = "Attack at dawn.";
  String encrypted_text = Utilities.encryptString(clear_text);
};
```

See Also

`cryptoEnabled, decryptString`

---

encryptString (Variant 2)

Encrypts a string using an encryption key that you select.

Syntax

```
public static final String
encryptString(String s, byte [] key)
```

Parameters

`s`

Input. The string to encrypt.

`key`
: A byte array containing byte values that represent the key used by the encryption algorithm. The array should be at least 8 bytes long.

Returns

Returns the encrypted string. If encryption is not enabled, it returns the original string.

See Also

`cryptoEnabled, decryptString`

ensureFolder

Makes sure a folder exists, creating subfolders as needed.

Syntax

```
static final boolean
ensureFolder(String path)
```

Parameters

`path`
The path to and name of the folder you want to check.

Returns

Returns `true` if the folder exists.

Example

The following example first retrieves a property from `futuretense.ini` and then uses that property's value to determine whether that directory exists:

```
String sTempHome = ics.GetProperty("ft.home");
if(Utilities.ensureFolder(sTempHome))
{
  // Yes, the folder exists.
}
else
{
  // The folder does not exist.
}
```

See Also

`fileExists`

fileExists

Makes sure a file exists.

Syntax

```
static final boolean
fileExists(String path)
```

Parameters

`path`
: The path to and name of the file you want to check.

Returns

Returns `true` if the file exists.

See Also

`ensureFolder`

fileFromSpec

Returns a file from a file specification. The specification does not have to be a complete or proper file path.

Syntax

```
public static final String
fileFromSpec(String spec)
```

Parameters

`spec`
A file specification.

Returns

Returns the file name.

fileLockLoadOK

Checks to see if the file lock native library loaded successfully.

Syntax

```
public static boolean
fileLockLoadOK()
```

Returns

Returns `true` if successful.

fileName

Creates a full path, including file name, given a folder and file name.

Syntax

```
public static final String
fileName(String folder,
         String fname)
```

Parameters

`folder`
Input. Specify the pathname of the folder (directory).

`fname`
Input. Specify the filename.

Description

The `fileName` method creates a full path, including file name, given a folder and file name. The path is internally converted by `osSafeSpec()`.

Returns

The full name and path of the file.

Example

The following code sets a string object named `Pathname` to the value `/tmp/test.txt`:

```
String sFolder = "/tmp";
String sFile = "test.txt";
String Pathname = Utilities.fileName( sFolder, sFile );
```

See Also

`osSafeSpec`

genID

The `genID` method generates an ID. This method has three variants, one of which is deprecated:

- `genID (Variant 1)` generates an ID that is safe within a virtual machine, but is not guaranteed cluster safe.
- `genID (Variant 2)` generates an ID that is cluster safe.
- `genID (Deprecated)` has been replaced by `genID (Variant 2)`.

---

genID (Variant 1)

Generates a unique number based on the current time.

Syntax

```
public static final String
genID()
```

Description

The `genID` method generates a unique number based on the current time.

This method is safe within the virtual machine. The ID is not guaranteed to be unique across several virtual machines.

Returns

Returns the string containing the number, or null on error.

---

genID (Variant 2)

Use this method to generate a unique number based on the current time. This version of `genID()` is cluster safe.

Syntax

```
public static final String
genID(boolean bClusterSafe )
```

Parameters

`bClusterSafe`
: A value of `true` generates an ID that is cluster safe.

Returns

Returns the string containing the number, or null on error.

genID (Deprecated)

Deprecated. *As of version 4.0, use* `genID (Variant 2)`.

Syntax

```
public static final String
genID(ICS icsThread)
```

Parameters

`icsThread`
: Input. Specify an interface object to synchronize this number across virtual machines and/or the cluster.

Description

The `genID` method generates a unique number based on the current time.

Returns

Returns the string containing the number, or null on error.

getParams

The `getParams` method creates a map or an `FTValList` from the name/value pairs that you supply. The name/value pairs must have the format:

```
a=b&c=d...
```

This method has two variants:

- `getParams (Variant 1)` builds an `FTValList` of parameters from the name/value pairs.
- `getParams (Variant 2)` builds a map of parameters from the name/value pairs.

---

getParams (Variant 1)

Builds an `FTValList` of parameters given input in the form of `"a=b&c=d..."`

Syntax

```
public static FTValList
getParams(String in)
```

Parameters

`in`
Input. Specify the string containing the parameters.

Returns

Returns an `FTValList` of parameters.

Example

The following code creates an `FTValList` object whose name/value pair list is initialized to `a=10`, `c=12`, and `x=8`.

```
FTValList ft = Utilities.getParams( "a=10&c=12&x=8" );
```

---

getParams (Variant 2)

Builds a map of parameters given input in the form of `"a=b&c=d..."`

Syntax

```
public static void
getParams(String in,
          Map map,
          boolean bDecode)
```

Parameters

`in`
Input. Specify the string containing the parameters.

`map`
The output map of the key/value pairs.

`bDecode`
A value of `true` enables decoding.

getPOP3Messages

The `getPOP3Messages` method gets a vector of mail messages from a POP3 server. There are two variants; their differences are as follows:

- `getPOP3Messages (Variant 1)` uses a default timeout value.
- `getPOP3Messages (Variant 2)` lets you set the timeout value.

---

getPOP3Messages (Variant 1)

Retrieves a vector of messages from a POP3 server.

Syntax

```
public static Vector
getPOP3Messages(String host,
                String user,
                String password,
                boolean bRemove)
```

Parameters

`host`
: The host name of the POP3 server that holds this user's e-mail.

`user`
: The user's account name.

`password`
: The user's password.

`bRemove`
: A boolean value. Set to `true` to remove the messages from the POP3 server after they are downloaded. Set to `false` to keep the messages on the POP3 server.

Description

The `getPOP3Messages` method retrieves a vector of messages from a POP3 server. The default connection timeout is 120 seconds.

The `Utilities` class provides a `msgBody` field, which you use to locate the body portion of a returned `getPOP3Messages()` element. It holds the name of the field in the message hashtable that contains the body of the message. (Other fields contain header information.)

Returns

Returns a vector of messages, or null. A message is a hashtable where each key is the name of a header field (the names are converted to lower case) and the value is the content of that header field. The name of the field that stores the body of the message is provided by `Utilities.Methods`.

Example

The following code downloads joe_user's e-mail and then displays its subject:

```
Vector theMail =
       Utilities.getPOP3Messages("popserver", "joe_user",
```

```
                                    "please", false);
    if (theMail == null)
        return "You do not have e-mail!";

    // For each message, print the subject field
    int numberMsgs = theMail.size();
    Hashtable aMessage = null;
    for (int i = 0; i < numberMsgs; i++)
    {
        aMessage = (Hashtable) theMail.elementAt(i);

        if (aMessage != null)
        {
      // Print the subject of the message (from the Subject: field)
            String s = "Message " + Integer.toString(i) + ": " +
            aMessage.get("subject");
            System.out.println(s);
        }
    }

    // Extract the body of the last message
    String theBody = (String) aMessage.get(Utilities.msgbody);
```

---

getPOP3Messages (Variant 2)

Retrieves a vector of messages from a POP3 server.

Syntax

```
public static Vector
getPOP3Messages(String host,
                String user,
                String password,
                boolean bRemove,
                int ctimeout)
```

Parameters

`host`
: The host name of the POP3 server that holds this user's e-mail.

`user`
: The user's account name.

`password`
: The user's password.

`bRemove`
: A boolean value. Set to `true` to remove the messages from the POP3 server after they are downloaded. Set to `false` to keep the messages on the POP3 server.

`ctimeout (optional)`
: The connection timeout in seconds. If not set, the default connection timeout is 120 seconds.

Description

The `getPOP3Messages` method retrieves a vector of messages from a POP3 server. The default connection timeout is 120 seconds.

Returns

Returns a vector of messages, or null. A message is a hashtable where each key is the name of a header field (the names are converted to lower case) and the value is the content of that header field. The name of the field that stores the body of the message is provided by `Utilities.Methods`.

goodString

Indicates whether a string is not empty or null.

Syntax

```
public static final boolean
goodString(String str)
```

Parameters

`str`
Input. Specify the string to evaluate.

Returns

Returns `true` if string is not null and length is not zero (0), otherwise, returns `false`.

Example

The following code determines whether the `cs.pagecachefolder` property contains a value:

```
String Property = "CS.Property.cs.pagecachefolder";
String path = ics.ResolveVariables(Property);
if ( !Utilities.goodString( path )
{
    // not set in properties; check arguments
    path = vIn.getValString( "pagecachefolder");
}
```

isFile

Verifies the existence of a file.

Syntax

```
public static final int
isFile(String path)
```

Parameters

`path`

Input. Specify the pathname of the file to check.

Description

The `isFile` method verifies the existence of a file. The method first checks that the argument is a `goodString()` and then converts it using `osSafeSpec()` before checking that the file exists.

Returns

Returns zero (0) on success, or error code on error.

Example

The following code gets the directory assigned to the `cs.pagecachefolder` property and then determines if that directory contains a file named `foo.txt`:

```
String Property = "CS.Property.cs.pagecachefolder";
String path = ics.ResolveVariables(Property);
if ( Utilities.isFile ( path + "/foo.txt" ) == 0 )
{
    // process file
}
```

See Also

`goodString, osSafeSpec`

isFolder

Verifies the existence of a folder (directory).

Syntax

```
public static final int
isFolder(String path)
```

Parameters

`path`

Input. Specify the pathname of the folder (directory) to check.

Description

The `isFolder` method verifies the existence of a folder (directory). It first checks that the argument is a `goodString()` and then converts it using `osSafeSpec()` before checking that the folder exists.

Returns

Returns zero (0) on success, or error code on error.

Example

The following code gets the directory assigned to the `cs.pagecachefolder` property and then determines if that directory exists:

```
String Property = "CS.Property.cs.pagecachefolder";
String path = ics.ResolveVariables(Property);
if ( Utilities.isFolder ( path ) == 0 )
{
    // process the folder
}
```

See Also

`goodString, osSafeSpec`

itemIsInList

Determines if a given string is in a vector.

Syntax

```
public static final boolean
itemIsInList(Vector list,
             String str,
             boolean case)
```

Parameters

`list`
Input. Specify the Vector containing a list of strings.

`str`
Input. Specify the string to find in `l`.

`case`
Input. Specify `true` to perform a case-insensitive search, specify `false` to specify a case-sensitive search.

Returns

Returns `true` if the string is in the vector, `false` if it is not.

Example

The following code determines if `Future` is one of the strings in the `actualVector`:

```
Vector actualVector = new Vector();
actualVector.addElement("Open");
actualVector.addElement("Future");
actualVector.addElement("FatWire");
boolean flag =
Utilities.itemIsInList(actualVector,"Future",false);
if (flag) {
     // Yes, Future is one of the strings in actualVector.
}
```

jdbcDateFromCal

Returns a string containing JDBC-formatted date and time from a calendar object.

Syntax

```
public static final String
jdbcDateFromCal(Calendar cal)
```

Parameters

`cal`

Input. Specify a calendar object.

Returns

Returns the JDBC-formatted string containing date and time.

Example

```
Calendar cal = Utilities.calendarFromJDBCString("2002-12-31
23:59:59");
String date = Utilities.jdbcDateFromCal(cal);
```

See Also

`calendarFromJDBCString, jdbcTimeFrom`

jdbcTimeFrom

Returns a JDBC formatted time from a calendar object.

Syntax

```
public static final String
jdbcTimeFrom(Calendar cal)
```

Parameters

`cal`

Calendar with date.

Returns

Returns the JDBC formatted time string, in the form "HH:MM:SS".

Example

```
Calendar cal =
    Utilities.calendarFromJDBCString("2002-12-31 23:59:59");
String date = Utilities.jdbcTimeFrom(cal);
```

See Also

`calendarFromJDBCString, jdbcDateFromCal`

keysFromMap

Returns a string of keys from a Map.

Syntax

```
public static String
keysFromMap(Map map)
```

Parameters

`map`

The map to convert to keys. See the Java SDK documentation for details about `Map`.

Returns

A string in the form "`key1,key2,key3`."

lockFile

Locks a file for exclusive use by this thread.

Syntax

```
public static boolean
lockFile(String file)
```

Parameters

`file`

Filename to lock. This must be a file specification that is valid for the operating system.

Description

The lockFile method locks a file for exclusive use by this thread.

Don't forget to release the lock. Otherwise, it is held until the virtual machine shuts down. `lockFile()` and `unlockFile()` provide a semaphore primitive that is useful in a cluster environment.

Returns

Boolean indicating success or failure.

See Also

`unlockFile`

osIgnoreFileCase

Indicates whether the server's operating system has case-sensitive file system naming.

Syntax

```
public static boolean
osIgnoreFileCase()
```

Returns

Returns `true` if the naming is case-insensitive, `false` if the naming is case sensitive or if case sensitivity cannot be determined.

osSafeSpec

Returns a file specification where the "/" or "\" characters have been replaced by the system specific path character.

Syntax

```
public static final String
osSafeSpec(String spec)
```

Parameters

`spec`
: The input file specification.

Returns

The converted string, specific to the operating system.

Example

```
String path = "/tmp/test.txt";
path = Utilities.osSafeSpec( path );
```

pathFromSpec

Returns a path from a file specification. The specification does not have to be a complete or proper file path.

Syntax

```
public static final String
pathFromSpec(String spec)
```

Parameters

`spec`
A file specification.

Returns

Returns the path to the file.

readByteFile

Reads in the file indicated by the passed file path, and returns the bytes.

Syntax

```
public static final byte[]
readByteFile(String spec)
```

Parameters

`spec`
: The path and file name of the file to read.

Description

The `readByteFile` method reads in the file indicated by the passed file path, and returns the bytes. The path is internally converted by `osSafeSpec()`.

Returns

A byte array of the files contents or null.

See Also

`osSafeSpec`

readByteURL

Reads the passed URL, and returns its bytes.

Syntax

```
public static final byte[]
readByteURL(String url)
```

Parameters

`url`
The URL to read; for example, "`http://www.FatWire.com/`".

Returns

The contents of the URL in a byte array, or null.

Example

The following example reads the URL `http://www.FatWire.com`:

```
byte byteArray[] =
           Utilities.readByteURL("http://www.FatWire.com/");
```

readFile

The `readFile` method reads the specified file and either returns its contents or writes the bytes to the I/O stream.

This method has three variants:

- `readFile (Variant 1)` reads the specified file and returns its contents. Assumes default encoding.
- `readFile (Variant 2)` reads the specified file and returns its contents. You can specify a nondefault encoding.
- `readFile (Variant 3)` reads the specified file and writes its contents to the specified output stream.

---

readFile (Variant 1)

Reads the specified file and returns its contents.

Syntax

```
public static final String
readFile(String pathname)
```

Parameters

`pathname`
: The pathname of the file to read.

Description

The `readFile` method reads the file indicated by the passed file path, and returns its contents. The path is internally converted by `osSafeSpec()`. The method assumes that the character of the file are in the default encoding. (The default encoding is set by the Java VM.)

Returns

If successful, the method returns the contents of the file. If unsuccessful, the method returns `null`.

Throws

`IOException`
: If the method could not read from `pathname` or write to `os`. Common reasons for failure include the following:
  - The file specified in `pathname` does not exist.
  - The file specified in `pathname` exists but the caller does not have permission to read from it.

Example

The following example reads and displays the contents of file `D:/java/sonnet.txt`. The code assumes that the contents of this file are in the default encoding:

```
String text = Utilities.readFile("D:/java/sonnet.txt");
System.out.println(text);
```

See Also

osSafeSpec

---

readFile (Variant 2)

Reads the specified file and returns its contents. Specify the encoding of the input file.

Syntax

```
public static final String
readFile(String pathname,
         String encoding)
  throws IOException
```

Parameters

`pathname`
: The pathname of the file to read.

`encoding`
: Specify the encoding of the contents of the file at `pathname`. Typical encoding values include `"UTF-8"` or `"Latin-1"`.

Description

The previous variant of `readfile` assumes that the contents of `pathname` are in the default encoding (which is set by the Java VM). By contrast, Variant 2 or `readfile` allows you to specify the encoding.

Returns

If successful, the method returns the contents of the file. If unsuccessful, the method returns `null`.

Throws

`IOException`
: If the method could not read from `pathname` or write to `os`. Common reasons for failure include the following:

- The file specified in pathname does not exist.
- The file specified in `pathname` exists but the caller does not have permission to read from it.

See Also

osSafeSpec

readFile (Variant 3)

Reads the file indicated by the passed file path and writes the bytes to the output stream.

Syntax

```
public static final boolean
readFile(String pathname,
         OutputStream os)
  throws IOException
```

Parameters

`pathname`
: The pathname of the file to read.

`os`
: The output stream to which this method should write the contents of the file.

Description

This `readFile` method reads the file indicated by the passed `pathname`, and writes its contents to the specified output stream. The path is internally converted by `osSafeSpec()`.

Returns

Returns `true` if no error.

Throws

`IOException`
: If the method could not read from `pathname` or write to `os`. Common reasons for failure include the following:

- The output stream specified in `os` is not writable.
- The file specified in `pathname` does not exist.
- The file specified in `pathname` exists but the caller does not have permission to read from it.

See Also

`osSafeSpec`

readURL

Reads the passed URL, and returns its contents as a string.

Syntax

```
public static final String
readURL(String url)
```

Parameters

`url`

The URL to read, for example, `"http://www.FatWire.com/"`.

Description

The `readURL` method reads the passed URL, and returns its contents as a string. This method strips out characters returned by `Character.isIdentifierIgnorable()`.

Returns

The contents of the URL in a string, or null.

Example

The following code writes the contents of `www.FatWire.com` into variable `strURL`:

```
String strURL = Utilities.readURL("http://www.FatWire.com/");
```

See Also

`readByteFile`

replaceAll

Replaces all occurrences in a string of a specified substring with a new string.

Syntax

```
public static final String
replaceAll(String source,
           String what,
           String with)
```

Parameters

`source`
: The source string to be modified. `NULL` or `empty` returns the same string.

`what`
: The substring to replace. To leave the `source` string unaltered, set `what` to `null`.

`with`
: The replacement text. To erase the substring identified by `what`, set `with` to `null`.

Description

The `replaceAll` method replaces all occurrences in a string of a specified substring with a new string.

Returns

Returns the new string.

Example

The following code fragment replaces both occurrences of `red` with `green`:

```
clearResult();
String str = "This red looks really red.";
String repStr = Utilities.replaceAll(str, "red", "green");
```

The resulting value of `repStr` will be:

```
This green looks really green.
```

sendMail

The `SendMail` method has a number of variants. All versions send an SMTP message to one or more recipients. The differences are as follows:

- `sendMail (Variant 1)`, which lets you specify both a MIME type and a character set.
- `sendMail (Variant 2)`, which lets you specify a MIME type. You accept the default character set.
- `sendMail (Variant 3)`, which forces you to accept the default MIME type and character set.
- `sendMail (Variant 4)`, which is the same as Variant 1 except that you provide an array of recipients.
- `sendMail (Variant 5)`, which is the same as Variant 2 except that you provide an array of recipients.
- `sendMail (Variant 6)`, which is the same as Variant 3 except that you provide an array of recipients.

To send mail, the Content Server properties file must contain valid values for the `cs.emailhost` and `cs.emailreturnto` properties.

---

sendMail (Variant 1)

Sends an STMP mail message, letting you designate a MIME type and a character set.

Syntax

```
public static final boolean
sendMail(String from,
         String to,
         String subject,
         String replyto,
         String messagestr,
         String server,
         String contenttype,
         String charset,
         StringBuffer err)
```

Parameters

`from`
: Specify the e-mail address of the sender.

`to`
: Specify the e-mail address of the recipient(s). If there are multiple recipients, use a comma to separate each e-mail address.

`subject`
: Specify the one-line summary (the subject) of the e-mail message.

`replyto`
: Specify the e-mail address to which the recipient should reply.

`body`
: Specify the e-mail message itself.

`server`
: Specify the name of the mail server that will send the message to the recipient.

`contenttype`
: Specify the MIME type of the body; for example, `text/plain` or `text/html`.

`charset`
: The character set for the `subject`. Popular values include:

- English: "`us-ascii`"
- UTF-8: "`utf-8`"
- Latin1: "`iso-8859-1`"
- Japanese: "`iso-2022-jp`"

`err`
: If there is a problem sending the message, `sendMail` returns errors here.

Returns

Returns `true` on success, `false` on failure. If `false` is returned, examine `err` to determine the reason.

Example

The following code sends a short e-mail message to two recipients:

```
String from = "frink@test.FatWire.com";
String to = "lenny@qa.FatWire.com, karl@qa.FatWire.com";
String subject = "Iberian precipitation";
String replyto = "support@test.FatWire.com";
String body = "The rain in Spain falls mainly on the plain.";
String server = "mail1";
String contenttype = "text/plain";
String charset = "iso-8859-1";
StringBuffer err = new StringBuffer();

boolean result = Utilities.sendMail(from, to, subject, replyto,
                                    body, server, contenttype,
                                    charset, err);
if (!result) {
  // Examine err
}
```

sendMail (Variant 2)

Sends an STMP mail message, letting you designate a content type. The character set defaults to the character set used by your J2EE platform.

Syntax

```
public static final boolean
sendMail(String from,
         String to,
         String subject,
         String replyto,
         String messagestr,
         String server,
         String contenttype,
         StringBuffer err)
```

Parameters

`from`
Specify the e-mail address of the sender.

`to`
Specify the e-mail address of the recipient(s). If there are multiple recipients, use a comma to separate each e-mail address.

`subject`
Specify the one-line summary (the subject) of the e-mail message.

`replyto`
Specify the e-mail address to which the recipient should reply.

`messagestr`
Specify the e-mail message itself.

`server`
Specify the name of the mail server that will relay the message to the recipient.

`contenttype`
Specify the MIME type of the message body; for example, `text/plain` or `text/html`.

`err`
If there is a problem sending the message, `sendMail` returns errors here.

Returns

Returns `true` on success, `false` on failure. If `false` is returned, examine `err` to determine the reason.

sendMail (Variant 3)

Sends an STMP mail message. The content type defaults to `text/plain`. The character set defaults to the character set used by your J2EE platform.

Syntax

```
public static final boolean
sendMail(String from,
         String to,
         String subject,
         String replyto,
         String messagestr,
         String server,
         StringBuffer err)
```

Parameters

`from`
Specify the e-mail address of the sender.

`to`
Specify the e-mail address of the recipient(s). If there are multiple recipients, use a comma to separate each e-mail address.

`subject`
Specify the one-line summary (the subject) of the e-mail message.

`replyto`
Specify the e-mail address to which the recipient should reply.

`messagestr`
Specify the e-mail message itself.

`server`
Specify the name of the mail server that will relay the message to the recipient.

`err`
If there is a problem sending the message, `sendMail` returns errors here.

Returns

Returns `true` on success, `false` on failure. If `false` is returned, examine `err` to determine the reason.

sendMail (Variant 4)

Sends an STMP mail message, letting you designate a MIME type and a character set. Specify an array of recipients.

Syntax

```
public static final boolean
sendMail(String from,
         String to[],
         String subject,
         String replyto,
         String messagestr,
         String server,
         String contenttype,
         String charset,
         StringBuffer err)
```

Parameters

`from`
Specify the e-mail address of the sender.

`to`
Specify the e-mail address of the recipient(s), each in a different cell of an array of strings.

`subject`
Specify the one-line summary (the subject) of the e-mail message.

`replyto`
Specify the e-mail address to which the recipient should reply.

`body`
Specify the e-mail message itself.

`server`
Specify the name of the mail server that will send the message to the recipient.

`contenttype`
Specify the MIME type of the body; for example, `text/plain` or `text/html`.

`charset`
The character set for the `subject`. Popular values include:

- English: "`us-ascii`"
- UTF-8: "`utf-8`"
- Latin1: "`iso-8859-1`"
- Japanese: "`iso-2022-jp`"

`err`
If there is a problem sending the message, `sendMail` returns errors here.

Returns

Returns `true` on success, `false` on failure. If `false` is returned, examine `err` to determine the reason.

Example

The following code sends a short e-mail message to two recipients:

```
String from = "frink@test.FatWire.com";
String[] to = new String[2];
to[0] = new String("lenny@qa.FatWire.com");
to[1] = new String("karl@qa.FatWire.com");
String subject = "Iberian precipitation";
String replyto = "support@test.FatWire.com";
String body = "The rain in Spain falls mainly on the plain.";
String server = "mail1";
String contenttype = "text/plain";
String charset = "iso-8859-1";
StringBuffer err = new StringBuffer();

boolean result = Utilities.sendMail(from, to, subject, replyto,
                                    body, server, contenttype,
                                    charset, err);
if (!result) {
  // Examine err
}
```

---

sendMail (Variant 5)

Sends an STMP mail message, letting you designate a content type. The character set defaults to the character set used by your J2EE platform. Specify an array of recipients.

Syntax

```
public static final boolean
sendMail(String from,
         String to[],
         String subject,
         String replyto,
         String messagestr,
         String server,
         String contenttype,
         StringBuffer err)
```

Parameters

`from`
: Specify the e-mail address of the sender.

`to`
: Specify the e-mail address of the recipient(s), each in a different cell of an array of strings.

`subject`
: Specify the one-line summary (the subject) of the e-mail message.

`replyto`
: Specify the e-mail address to which the recipient should reply.

`messagestr`
Specify the e-mail message itself.

`server`
Specify the name of the mail server that will relay the message to the recipient.

`contenttype`
Specify the MIME type of the message body; for example, `text/plain` or `text/html`.

`err`
If there is a problem sending the message, `sendMail` returns errors here.

Returns

Returns `true` on success, `false` on failure. If `false` is returned, examine `err` to determine the reason.

---

sendMail (Variant 6)

Sends an STMP mail message. The content type defaults to `text/plain`. The character set defaults to the character set used by your J2EE platform. Specify an array of recipients.

Syntax

```
public static final boolean
sendMail(String from,
         String to[],
         String subject,
         String replyto,
         String messagestr,
         String server,
         StringBuffer err)
```

Parameters

`from`
Specify the e-mail address of the sender.

`to`
Specify the e-mail address of the recipient(s), each in a different cell of an array of strings.

`subject`
Specify the one-line summary (the subject) of the e-mail message.

`replyto`
Specify the e-mail address to which the recipient should reply.

`messagestr`
Specify the e-mail message itself.

`server`
Specify the name of the mail server that will relay the message to the recipient.

`err`
If there is a problem sending the message, `sendMail` returns errors here.

Returns

Returns `true` on success, `false` on failure. If `false` is returned, examine `err` to determine the reason.

sortWords

Sorts a list of strings by length, from longest to shortest.

Syntax

```
public static final Vector
sortWords(Vector words)
```

Parameters

`words`
Vector of strings.

Description

The `sortWords` method sorts a list of strings by length, from longest to shortest. A new vector is returned, leaving the original unchanged.

Returns

New vector of strings (the original is untouched).

Example

The following code sorts a list of strings from the shortest to the longest:

```
Vector aVector = new Vector();
aVector.addElement("bear");
aVector.addElement("walrus");
aVector.addElement("cow");
aVector.addElement("llama");
Vector sortedVector = Utilities.sortWords(aVector);
```

After executing the preceding code, the items in `sortedVector` are in the following order:

```
"cow"
"bear"
"llama"
"walrus"
```

sortWordsAlpha

Sorts a list of strings alphanumerically.

Syntax

```
public static final Vector
sortWordsAlpha(Vector words)
```

Parameters

`words`
Vector of strings.

Description

The `sortWordsAlpha` method sorts a list of strings alphanumerically. A new vector is returned, leaving the original unchanged.

Returns

New vector of strings (the original is untouched).

Example

The following code sorts a list of strings alphabetically:

```
Vector aVector = new Vector();
aVector.addElement("bear");
aVector.addElement("walrus");
aVector.addElement("cow");
aVector.addElement("llama");
Vector sortedVector = Utilities.sortWordsAlpha(aVector);
```

After executing the preceding code, the items in `sortedVector` are in the following order:

```
"bear"
"cow"
"llama"
"walrus"
```

stringFromValList

The `stringFromValList` method creates a URL-like set of name/value pairs from a map. This method has two variants:

- `stringFromValList (Variant 1)` creates a URL-like set of name/value pairs from a map.
- `stringFromValList (Variant 2)` creates a URL-like set of name/value pairs from a map and allows you to encode names and values from outside of Content Server.

---

stringFromValList (Variant 1)

Creates a URL-like set of name/value pairs from a map.

Syntax

```
public static String
stringFromValList(Map map)
```

Parameters

`map`
The map you want to compress into a string.

Description

The `stringFromValList` method creates a URL-like set of name/value pairs from a map; for example, `x=y&a=b&c=d`. In previous versions of Content Server, `stringFromValList` accepted an `IList` rather than a map. In the current version of Content Server, IList is a map.

`stringFromValList` returns a string, but that string needs to be encoded to be URL safe. Note that empty strings are not preserved.

Returns

The string of name/value pairs.

---

stringFromValList (Variant 2)

Creates a URL-like set of name/value pairs from a map.

Syntax

```
public static String
stringFromValList(Map map,
                  Boolean bEncode)
```

Parameters

`map`
The map you want to compress into a string.

`bEncode`
: Set this value to `true` to encode keys and values from outside Content Server.

Description

The `stringFromValList` method creates a URL-like set of name/value pairs from a map; for example, `x=y&a=b&c=d`. In previous versions of Content Server, `stringFromValList` accepted an `IList` rather than a map. In the current version of Content Server, `IList` is a map.

`stringFromValList` returns a string, but that string needs to be encoded to be URL safe. Note that empty strings are not preserved.

Returns

The string of name/value pairs.

sqlDate

Returns a string formatted as JDBC date/time from a calendar object.

Syntax

```
public static final String
sqlDate(Calendar cal)
```

Parameters

`cal`
Calendar with date and time.

Returns

Returns the date and time in form YYYY-MM-DD HH:MM:SS.

Example

The following code writes the final second of 2002 into `dateStr`:

```
Calendar cal =
    Utilities.calendarFromJDBCString("2002-12-31 23:59:59");
String dateStr = Utilities.sqlDate(cal);
```

See Also

`calendarFromJDBCString`

unlockFile

Unlocks a file from exclusive use by this thread.

Syntax

```
public static boolean
unlockFile(String file)
```

Parameters

`file`

File name to unlock. This must be a file specification that is valid for the operating system.

Description

The `unlockFile` method unlocks a file from exclusive use by this thread. `lockFile()` and `unlockFile()` provide a semaphore primitive that is useful in a cluster environment.

Returns

Returns `true` if the specified file was successfully unlocked; otherwise, returns `false`. Reasons for failure include specifying a file that was not locked.

See Also

`lockFile`

unzipFile

Extracts the files contained in a zip file into a specified folder. .

Syntax

```
public static final int
unzipFile(String zip,
          String destination)
```

Parameters

`zip`
The path and name of the file to unzip.

`destination`
The directory where the extracted files are to be stored.

Returns

Returns zero (0) on success, or error code on error.

Description

Call `unzipFile` to extract the specified `zip` file into the specified `destination` directory. The unzipFile method automatically calls `osSafeSpec` prior to performing the extraction. Prior to calling `unzipFile`, your code should first call `validateZip`.

Example

See the example for `validateZip`.

See Also

`osSafeSpec, validateZip`

validateZip

Validates the integrity of a zip file.

Syntax

```
public static final boolean
validateZip(String zip)
```

Parameters

`zip`
The pathname of the zip file to validate.

Returns

Returns `true` if zip file is good, `false` if not.

Example

Validate and unzip the zip file stored at `c:\xx.zip`:

```
String MyZipFile = "c:\xx.zip";
if ( Utilities.validateZip(MyZipFile) )
{
 // zip file is valid; unzip it into the \temp directory
   String UnzipTarget = "c:\temp";
   boolean UnzipResult;
   UnzipResult = Utilities.unzipFile(MyZipFile, UnzipTarget);
   if (!UnzipResult) {
     // Problem unzip'ing the zip file.
   }
}
```

See Also

`unzipFile`

words

The `words` method parses a string into a vector of words. The separator specifies the character that serves as the delimiter between words. White space is removed from both ends of each word parsed, using `String.trim()`.

This method has two variants:

- `words (Variant 1)` accepts integers as delimiters.
- `words (Variant 2)` accepts strings as delimiters.

---

words (Variant 1)

Parses a string into a vector of words.

Syntax

```
public static final
Vector words(String str,
             int sep)
```

Parameters

`str`
: String of words to parse.

`sep`
: The separator character.

Description

The `words` method parses a string into a vector of words. The separator specifies the character that serves as the delimiter between words. White space is removed from both ends of each word parsed, using `String.trim()`.

Returns

Returns a vector of words (may be empty).

---

words (Variant 2)

Parses a string into a vector of words.

Syntax

```
public static final Vector
words(String str,
      String sep)
```

Parameters

`str`
: String of words to parse.

`sep`
: String of separating characters.

Description

The `words` method parses a string into a vector of words. The separator specifies the string of characters that serves as the delimiter between words. Whitespace is removed from both ends of each word parsed, using `String.trim()`.

Returns

Returns a vector of words (may be empty).

Examples

The following code returns a vector containing the three strings "Jack", "Jill", "Little Alice".

```
Utilities.words( "Jack and Jill and Little Alice", "and" );
```

The following code returns an empty vector, not a vector containing two empty strings:

```
Utilities.words ( " and ", "and" );
```

writeFile

The `writeFile` method writes a file to disk, given a path and the contents of the file. The contents are in a byte array. The path is internally converted by `osSafeSpec()`.

This method has two variants:

- `writeFile (Variant 1)` accepts a string.
- `writeFile (Variant 2)` accepts an array of bytes.

---

writeFile (Variant 1)

Writes a file to disk, given a pathname and the contents of the file.

Syntax

```
public static final boolean
writeFile(String fname,
          String str)
```

Parameters

`fname`
: The pathname of the target file.

`str`
: The data to write to the file.

Returns

Returns `true` on success, `false` on failure.

Description

Writes a file to disk, given a path and the contents of the file. The path is internally converted by `osSafeSpec()`.

See Also

`osSafeSpec()`

---

writeFile (Variant 2)

Writes a file to disk, given a pathname, the contents of the file, and the number of bytes to write.

Syntax

```
public static final boolean
writeFile(String fname,
          byte[] bytes,
          int len)
```

Parameters

`fname`
: The pathname of the target file.

`bytes`
: The byte array to write to the file.

`len`
: The number of bytes to write to the file.

Description

The `writeFile` method writes a file to disk, given a path and the contents of the file. The contents are in a byte array. The writeFile method writes up to `len` bytes of the input byte array. The path is internally converted by `osSafeSpec`.

Returns

Returns `true` on success, `false` on failure.

See Also

`osSafeSpec`

Index

D

E

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y