Content Server Enterprise Edition

Version: 5.5

Web Services Reference

Document Revision Date: Oct. 31, 2003

Web Services Reference
Document Revision Date: Oct. 31, 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

WSDL Overview

This chapter provides an overview of the predefined web services shipped with Content Server. Each supplied web service is defined in WSDL (web services description language) format.

About WSDL

WSDL is an XML format that describes distributed services on the Internet. A WSDL file describes the location of the service and the data to be passed in messages for particular operations. With regard to Content Server, these messages contain remote procedure calls.

Web services for Content Server are defined in supplied WSDL files. Each WSDL file contains descriptions of multiple operations, each of which corresponds to a Content Server delivery function. Operations are grouped by WSDL file according to related function. Like all web services, operations are accessible via XML and SOAP (simple object access protocol) messaging over the Internet.

WSDL files, which contain the information necessary to code a SOAP-compliant interface to Content Server, are intended to be read by various third-party tools that automatically output client code. The generated client code passes required infomation as remote procedure calls to Content Server via SOAP. After you pass the required inputs to the client program, the client creates and sends a SOAP request based on the specified operation. The resulting SOAP request is handled by a suppled Content Server page, which packages the request and return data.

Note

If you are not using the predefined services provided with Content Server, you must create your own WSDL file to describe your web service and Content Server page to handle data. For more information about using predefined web services and creating custom web services, refer to the *CSEE Developer's Guide*.

Supported WSDL Version

Content Server 5.5 supports WSDL 1.1. You will need to understand the web services description language to use the predefined WSDL files shipped with Content Server. The XML standard syntax is described in detail at the following W3C web site:

http://www.w3.org

Supplied WSDL Files

Supplied WSDL files group operations by related Content Server delivery functions. Identify the WSDL file that contains the Content Server functions you need. Then generate a client interface for your intended web service application. The following tables list related operations according to the WSDL file in which they are defined:

- Asset.wsdl
- AssetSet.wsdl
- Miscellaneous.wsdl
- SitePlan.wsdl

Asset.wsdl

Defines asset operations that can list and load asset fields or retrieve children and parents for assets and flex assets.

AssetGetChildren	AssetGetSiteNode	AssetGetSiteParent
AssetList	AssetLoad	

AssetSet.wsdl

Defines operations required to retrieve attribute values for flex assets.

ASGetAssetCount	ASGetAttributeValues	ASGetMultipleValues

Also defines the following inputs to the AssetSet operations, which are used to build a SearchState. SearchState Objects are used to generate a collection of asset properties based on specified criteria.

LikeConstraint	NestedConstraint	RangeConstraint	RichTextConstraint
StandardConstraint			

Miscellaneous.wsdl

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

MiscGetBlob	MiscGetMungoBlob	MiscSearch

SitePlan.wsdl

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

SPGetChildren	SPGetProperties

WSDL File Location

Predefined WSDL files for Content Server are automatically installed with the CS-Direct application in the following location:

`http://install_dir/futuretense_cs/Xcelerate/wsdl/*.wsdl`

Chapter 2

Operations

This chapter describes predefined functions that are available to web services clients for accessing selected information from the Content Server repository. It describes syntax and required inputs for your web services client, as defined by the WSDL files supplied with Content Server 5.5.

Operations are grouped according to function and WSDL file, as follows:

- Asset Operations
- AssetSet Operations
- Miscellaneous Operations
- SitePlan Operations

Note

Syntax for web services operations and client-code examples are provided in Java. The exact syntax you use will depend on the programming language in which you code your client program.

Regardless of the tool and language you use to generate the client, parameter names for required inputs and the order in which they are passed remain the same, and the output is always SOAP (simple object access protocol). The examples in this chapter were generated using the Apache Axis tool.

Asset Operations

Asset operations cover the functions required to retrieve basic attribute values for assets and flex assets.

The following asset operations invoke basic asset functions:

- AssetGetChildren
- AssetGetSiteNode
- AssetGetSiteParent
- AssetList
- AssetLoad

AssetGetChildren

Queries the AssetRelationTree table and builds a list of child assets for the specified parent asset.

Syntax

```
assetGetChildren(java.lang.String authusername,
java.lang.String authpassword,
java.lang.String TYPE,
java.lang.String OBJECTID,
java.lang.String FIELD,
java.lang.String VALUE,
java.lang.String CODE,
java.lang.String CHILDTYPE,
java.lang.String CHILDID,
java.lang.String ORDER)
```

Parameters

`authusername (required)`
: (String) Name of the user to log in, as registered in Content Server.

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

`TYPE (required)`
: (String) The asset type of the asset that you want to retrieve from the database. For your web service, the value must be `page`.The list of children is a join of the AssetRelationTree and the asset table for the type specified.

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

`OBJECTID (required)`
: (String) The unique identifying number that references the `Asset` object. Not required if you use TYPE with the FIELD and VALUE paired parameters (defined below).

`FIELD (optional)`

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

Not required if you use `TYPE` with the `OBJECTID` parameter.

`VALUE (optional)`

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

Not required if you use `TYPE` with the `OBJECTID` parameter.

`CODE (optional)`

(String). Restricts the list to include only the child assets that have the relationship (association) specified by this parameter. You can restrict the list by named associations or by unnamed relationships. This value comes from the `ncode` field of the AssetRelationTree.

For example, if you include `OBJECTTYPE="Image"` and `CODE="MainImage"` for an article asset, CS-Direct lists only the image asset that is related to that article asset by the Main Image named association. Without the `CODE` entry, CS-Direct lists all the images associated with the article.

To list only the child assets that are associated with the parent asset by unnamed relationships, use a hyphen (-). For example: `CODE= "-"`

If you do not specify `OBJECTTYPE` or `CODE`, the list includes all children with named associations or unnamed relationships to the parent asset.

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

`CHILDTYPE (optional)`

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

`CHILDID (optional)`

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

`ORDER (optional)`

(String) The fields to sort the list by, and whether the sort result on those fields is `ascending` or `descending`. For example, you can specify `ID`, `name`, `date`, `created by user`, and so on. By default, the sort is `ascending`. If you specify more than one field, separate the field names with a comma.

For example, if you specify `ORDER="nrank"`, the list is sorted by rank starting at number 1. If you want the list sorted by descending rank, use `ORDER="nrank desc"`.

Description

This operation queries the AssetRelationTree table for a list of the children of the asset that you specify, listing each child with a value for all of the fields from that table (`nid`,

nparentid, nrank, otype, oid, and ncode). You must load the parent asset with the AssetLoad operation before you can invoke this operation to query for and list its children. The list is a standard Content Server list.

Typical use for this operation is to retrieve the assets referred to by a collection asset, the image assets associated with an article asset, and so on. You can then loop through the list, or reference the data returned in the list to display the relevant information from those assets.

You can restrict the list of children by association name (CODE), object type, object ID, and rank (ORDER).

If you use the TYPE parameter, the resulting list of children is a join of the AssetRelationTree and the asset table for the type specified and contains data from both tables. In that case, you do not need to use the AssetLoad operation for a child asset of that type if that asset type stores all of its asset data in the primary asset table. Instead, you can get that information from the list.

Returns

Ilist containing children.

errno

The possible values of errno include:

Value	Description
-111	The asset has no children.
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-10006	The object ID is not valid.
-10007	The version of the object is not valid.
-10009	More than one object met the specified criteria.

Example

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

```
AssetService service = (AssetService)new AssetService_Impl();
AssetPortType port = service.getAssetPort();
IList myList = port.assetGetChildren("user_author", "demo",
"Page", "968685128066", null, null, null, "Query", null, null);
```

AssetGetSiteNode

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

Syntax

```
assetGetSiteNode(java.lang.String authusername,
java.lang.String authpassword,
java.lang.String TYPE,
java.lang.String OBJECTID,
java.lang.String FIELD,
java.lang.String VALUE,
java.lang.String EXCLUDE,
java.lang.String FIELDLIST)
```

Parameters

`authusername (required)`
: (String) Name of the user to log in, as registered in Content Server.

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

`TYPE (required)`
: (String) The asset type of the asset that you want to retrieve from the database. For your web service, the value must be `page`.

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

`OBJECTID (required)`
: (String) The unique identifying number that references the `Asset` object. Not required if you use TYPE with the FIELD and VALUE paired parameters.

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

  Not required if you use `TYPE` with the `OBJECTID` parameter.

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

  Not required if you use `TYPE` with the `OBJECTID` parameter.

Description

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

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

Returns

String containing the site node ID.

errno

The possible values of `errno` include:

Value	Description
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-10006	The object ID is not valid.
-10007	The version of the object is not valid.
-10009	More than one object met the specified criteria.

Example

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

```
AssetService service = (AssetService)new AssetService_Impl();
AssetPortType port = service.getAssetPort();
String retVal = port.assetGetSiteNode("user_author", "demo",
"Page", "968685128066", null, null, null, null);
```

AssetGetSiteParent

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

Syntax

```
assetGetSiteParent(java.lang.String authusername,
java.lang.String authpassword,
java.lang.String TYPE,
java.lang.String OBJECTID,
java.lang.String FIELD,
java.lang.String VALUE,
java.lang.String EXCLUDE,
java.lang.String FIELDLIST)
```

Parameters

`authusername (required)`
: (String) Name of the user to log in, as registered in Content Server.

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

`TYPE (required)`
: (String) The asset type of the asset that you want to retrieve from the database. For a web service, the value must be `page`.

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

`OBJECTID (optional)`
: (String) The unique identifying number that references the `Asset` object. Not required if you use `TYPE` with the `FIELD` and `VALUE` paired parameters (defined below).

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

  Not required if you use `TYPE` with the `OBJECTID` parameter.

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

  Not required if you use `TYPE` with the `OBJECTID` parameter.

`EXCLUDE (optional)`
: (Boolean) Depending on whether the value of `EXCLUDE` is `True` or `False`, this operation either returns the fields specified in the `FIELDLIST` parameter or returns the fields not contained in the list.

  `True` indicates that all fields except the fields in the `FIELDLIST` are to be returned.

  `False` indicates that only fields in `FIELDLIST` are to be returned. That is, it returns fields specified with the `FIELDLIST` parameter. The default value is `False`.

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

Description

This operation queries the `SitePlanTree` table and then loads, as an object, the parent page of the specified page asset. It functions like `AssetLoad`.

You typically use this operation to display information about the hierarchical position of the current page, for example, to create a link to the current page's parent page. This operation determines the parent page and then loads the parent page.

Returns

`Asset` object that represents the site parent.

errno

The possible values of `errno` include:

Value	Description
-112	No parent exists.
-10001	The implementing class is invalid.
-10002	There is a missing operation for the implementing class.
-10003	The operation could not be invoked successfully.
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-10006	The object ID is not valid.
-10007	The version of the object is not valid.
-10008	The node ID of the object is not valid.
-10009	More than one object met the specified criteria.
-12007	The specified value is not valid.

Example

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

```
AssetService service = (AssetService)new AssetService_Impl();
AssetPortType port = service.getAssetPort();
LoadedAsset la = port.assetGetSiteParent("user_author", "demo",
"Page", "990743462410", null, null, null, null);
```

AssetList

Returns a list of assets for a specific asset type. The list is filtered according to specified criteria.

Syntax

```
assetList(java.lang.String authusername,
java.lang.String authpassword,
java.lang.String TYPE,
java.lang.String FIELD1,
java.lang.String FIELD2,
java.lang.String FIELD3,
```

```
java.lang.String FIELD4,
java.lang.String FIELD5,
java.lang.String FIELD6,
java.lang.String FIELD7,
java.lang.String FIELD8,
java.lang.String FIELD9,
java.lang.String VALUE1,
java.lang.String VALUE2,
java.lang.String VALUE3,
java.lang.String VALUE4,
java.lang.String VALUE5,
java.lang.String VALUE6,
java.lang.String VALUE7,
java.lang.String VALUE8,
java.lang.String VALUE9,
java.lang.String ORDER,
java.lang.String SITEID,
java.lang.String EXCLUDEVOIDED)
```

Parameters

`authusername (required)`
: (String) Name of the user to log in, as registered in Content Server.

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

`TYPE (required)`
: (String) The asset type of the asset that you want to retrieve from the database. For your web service, the value must be `page`.

`FIELD (1–9) (optional)`
: Input. The name of a field to use to restrict the list. You can specify up to nine fields. If you specify a field name, you must also pass in a corresponding VALUE for the field. For example: FIELD1="Category" VALUE1= "Sports".

`VALUE (1–9) (optional)`
: Input. Required if FIELD is specified. The field value to use to restrict the list. You can specify up to nine FIELD/VALUE pairs. For example, VALUE1, VALUE2, and so on.

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

  Input. The fields to sort the list by, and whether the sort result on those fields is `ascending` or `descending`. By default, the sort is `ascending`. If you specify more than one field, separate the field names with a comma.

  For example, if you specify `ORDER="nrank"`, the list is sorted by rank starting at number *1*. If you want the list sorted by descending rank, use `ORDER="nrank desc"`.

`SITEID (optional)`
: ID of the site (formerly publication) to which the query is restricted. When this optional parameter is used, the query is a database join operation against the AssetPublication table.

`EXCLUDEVOIDED (optional)`

The specified value determines whether previously deleted (voided) assets are returned:

`True` returns assets that have not been deleted and that meet the specified criteria. The default value is `True`.

`False` returns deleted and undeleted assets that meet the specified criteria.

Description

The returned list that contains rows that exactly match the specified name/value pairs for the supplied asset type. A FIELD and VALUE pair comprise the name of the field and its value. Because you are trying to match the value, you must know it beforehand. If no name/value pairs are specified, all rows and their associated values are returned. In addition, all columns in the AssetTypes main table are returned.

Inputs

Accepts specified name/value pairs as input.

Returns

Rows from the asset main table that include values that match the specified columns.

errno

The possible values of `errno` include:

Value	Description
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing.

Example

```
AssetService service = (AssetService)new AssetService_Impl();
AssetPortType port = service.getAssetPort();
IList myList = port.assetList("user_author", "demo", "Products",
"status", null, null, null, null, null, null, null, null, "PL",
null, null, null, null, null, null, null, null, null, null, null);
```

AssetLoad

Queries the database for the specified asset, loads an instance of the asset into memory within the `Asset` object, and returns the loaded asset in an XML wrapper.

Note

This operation does not load flex assets.

Syntax

```
assetLoad(java.lang.String authusername,
java.lang.String authpassword,
java.lang.String TYPE,
java.lang.String OBJECTID,
java.lang.String FIELD,
java.lang.String VALUE,
java.lang.String EXCLUDE,
java.lang.String FIELDLIST)
```

Parameters

`authusername (required)`
: (String) Name of the user to log in, as registered in Content Server.

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

`TYPE (required)`
: (String) The asset type of the asset that you want to retrieve from the database. For your web service, the value must be `page`.

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

`OBJECTID (optional)`
: (String) The unique identifying number that references the `Asset` object. Not required if you use `TYPE` with the `FIELD` and `VALUE` paired parameters.

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

  Not required if you use `TYPE` with the `OBJECTID` parameter.

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

  Not required if you use `TYPE` with the `OBJECTID` parameter.

`EXCLUDE (optional)`

(Boolean) Depending on whether the value of `EXCLUDE` is `True` or `False`, the `AssetLoad` operation either returns the fields specified in the `FIELDLIST` or returns the fields that are not contained in the list.

`True` indicates that all fields except the fields in the `FIELDLIST` are to be returned.

`False` indicates that only fields in `FIELDLIST` are to be returned. That is, it returns fields specified with the `FIELDLIST` parameter. The default value is `False`.

`FIELDLIST (optional)`

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

Description

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

Multiple parameters provide different ways to identify the asset to load. Enter the parameter or parameter combination that is the most convenient identifier for your application, and leave blank placeholders for the unused parameters. Although only the `TYPE` parameter is required, you must also supply at least the `OBJECTID` or the `FIELD` and `VALUE` paired parameters. You can identify the asset that you want to load by specifying its asset type and object ID, or by specifying its asset type combined with a field/value pair that uniquely identifies the asset.

Typically, each web service calls the `AssetLoad` operation to start so that subsequent code can extract and display the loaded data on pages. All input parameters are strings.

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

Returns

Asset object that contains the requested field values.

errno

The possible values of `errno` include:

Value	Description
-10001	The implementing class is invalid.
-10002	There is a missing method for the implementing class.
-10003	The method could not be invoked successfully.
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-10006	The object ID is not valid.

Value	Description
-10007	The version of the object is not valid.
-10009	More than one object met the specified criteria.

Example

This code loads a page asset, identifying it by type and object ID:

```
AssetService service = (AssetService)new AssetService_Impl();
AssetPortType port = service.getAssetPort();
LoadedAsset la = port.assetLoad("user_author", "demo", "Page",
"968685128066", null, null, null, null);
```

See Also

AssetGetChildren

AssetSet Operations

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

- ASGetAssetCount
- ASGetAssetList
- ASGetAttributeValues
- ASGetMultipleValues

ASGetAssetCount

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

Syntax

```
ASGetAssetCount(java.lang.String authusername,
java.lang.String authpassword,
com.FatWire.IList LIST,
java.lang.String ASSETTYPES,
com.FatWire.IList ASSETTYPESLIST,
com.FatWire.Searchstate SEARCHSTATE,
java.lang.String LOCALE,
com.FatWire.IList ASSETS)
```

Parameters

`authusername (required)`
(String) Name of the user to log in, as registered in Content Server.

`authpassword(required)`
(String) Password for the associated user name.

`LIST (Optional)`
(`IList` object) Input parameter. Name of the list that determines sort order. The list describes how the returned assets are to be sorted.

The `IList` object has three columns:

`attributetypename`

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

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

`ASSETTYPESLIST (Optional)`
(`IList` object) Input list containing one `assettype` column that includes values that restrict the count.

Specify either `ASSETTYPESLIST` and `ASSETS` or `SEARCHSTATE` and `ASSETTYPES`, but not both. If you specify `ASSETTYPESLIST`, `ASSETS` is required. If both parameters are supplied, `ASSETTYPESLIST` is used by default.

`SEARCHSTATE (Optional)`

(`Searchstate` object) Input parameter. `Searchstate` object that contains the criteria to match. The `Searchstate` object describes the search constraints, if any.

Specify either `SEARCHSTATE` and `ASSETTYPES` or `ASSETTYPESLIST` and `ASSET`, but not both. If both parameters are supplied, `ASSETTYPESLIST` and `ASSETS` is used by default. To create a SearchState, use the SearchState Objects.

`ASSETTYPES (Optional)`

(String) Input parameter. Name of a list of asset types to include in building the asset count. Comma separated list of flex asset types to match. If null, then all assets in the system are considered. Specify either `ASSETTYPESLIST` and `ASSETS` or `SEARCHSTATE` and `ASSETTYPES`, but not both.

`LOCALE (Optional)`

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

`ASSETS`

(IList) Name of a list of assets passed to Content Server that form the asset set. The list of assets is an IList with two columns (`assetid` and `assettype`). This operation is required if you are supplying a list of assets to build the asset set instead of a SearchState. Specify either `ASSETTYPELIST` and `ASSETS` or `ASSETTYPES` and `SEARCHSTATE`, but not both. If both parameters are supplied, `ASSETS` is used by default.

Returns

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

Example

The following code creates an asset types list from SearchState input.

```
AssetsetService service = (AssetsetService)new
AssetsetService_Impl();
AssetsetPortType port = service.getAssetsetPort();

Searchstate ss = new Searchstate();

String retVal = port.ASGetAssetCount("user_author", "demo", null,
"AArticles", null, ss, null, null);
```

ASGetAssetList

Returns a list of flex assets that exactly match the specified criteria or the created assetset.

Syntax

```
ASGetAssetList(java.lang.String authusername,
java.lang.String authpassword,
com.FatWire.IList LIST,
java.lang.String MAXCOUNT,
java.lang.String METHOD,
```

```
java.lang.String ASSETTYPES,
com.FatWire.IList ASSETTYPESLIST,
com.FatWire.Searchstate SEARCHSTATE,
java.lang.String LOCALE,
com.FatWire.IList ASSETS)
```

Parameters

`authusername (required)`
(String) Name of the user to log in, as registered in Content Server.

`authpassword(required)`
(String) Password for the associated user name.

`LIST (Optional)`
`(IList object)` Name of the list that determines how the returned assets are to be sorted. The list has three columns:

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

`ASSETTYPESLIST (Optional)`
(`IList` object) Input list containing one `assettype` column that includes values that restrict the count.

Specify either `ASSETTYPESLIST` and `ASSETS` or `SEARCHSTATE` and `ASSETTYPES`, but not both. If you specify `ASSETTYPESLIST`, `ASSETS` is required. If both parameters are supplied, `ASSETTYPESLIST` is used by default.

`SEARCHSTATE (Optional)`
(`Searchstate` object) Input parameter. `Searchstate` object that contains the criteria to match. The `Searchstate` object describes the search constraints, if any.

Specify either `SEARCHSTATE` and `ASSETTYPES` or `ASSETTYPESLIST` and `ASSET`, but not both. If both parameters are supplied, `ASSETTYPESLIST` and `ASSETS` is used by default.

`ASSETTYPES (Optional)`
(String) Input parameter. Name of a list of asset types to include in building the asset count. Comma separated list of flex asset types to match. If null, then all assets in the system are considered. Specify either `ASSETTYPESLIST` and `ASSETS` or `SEARCHSTATE` and `ASSETTYPES`, but not both.

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

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

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

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

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

ASSETS
: (IList) Name of a list of assets passed to Content Server that form the asset set. The list of assets is an IList with two columns (assetid and assettype). This operation is required if you are supplying a list of assets to build the asset set instead of a SearchState. Specify either ASSETTYPELIST and ASSETS or ASSETTYPES and SEARCHSTATE, but not both. If both parameters are supplied, ASSETS is used by default.

Description

Returns a list of flex assets that exactly match the criteria specified by either the SEARCHSTATE parameter or the AssetList. The operation only works with flex assets, which are assets supplied with the CS-Direct Advantage and CS-Engage products. The list contains two columns: assettype and assetid.

Returns

A list of assets for the created assetset.

Example

The following code demonstrates GetAssetList:

```
AssetsetService service = (AssetsetService)new
AssetsetService_Impl();
AssetsetPortType port = service.getAssetsetPort();

IList myList = port.ASGetAssetList("user_author", "demo", null,
null, null, "Products", null, null, null, null);
```

ASGetAttributeValues

Returns specified attribute values for all flex assets for the created assetset.

Syntax

```
ASGetAttributeValues(java.lang.String authusername,
java.lang.String authpassword,
java.lang.String TYPENAME,
java.lang.String ATTRIBUTE,
java.lang.String ORDERING,
java.lang.String ASSETTYPES,
com.FatWire.IList ASSETTYPESLIST,
com.FatWire.Searchstate SEARCHSTATE,
java.lang.String LOCALE,
com.FatWire.IList ASSETS)
```

Parameters

`authusername (required)`
(String) Name of the user to log in, as registered in Content Server.

`authpassword(required)`
(String) Password for the associated user name.

`TYPENAME (optional)`
(String) Input parameter. The internal asset name for the attribute (either CAttributes for content attribute, or PAttributes for product attribute). CAttribute specifies the content flex asset type. PAttribute indicates the product flex asset type. If you do not specify `TYPENAME`, a value is supplied from a property in the `gator.ini` property file: `mwb.defaultattributes=PAttributes`. The default value, which is `PAttributes`, may be changed.

`ATTRIBUTE (required)`
Input parameter. Name of the attribute for which you want to retrieve values.

`ORDERING (optional)`
Input parameter. Indicates whether the result list should be in ascending or descending order. Value can be either `ascending` or `descending`.

`ASSETTYPES (Optional)`
(String) Input parameter. Name of a list of asset types to include in building the asset count. Comma separated list of flex asset types to match. If null, then all assets in the system are considered. Specify either `ASSETTYPESLIST` and `ASSETS` or `SEARCHSTATE` and `ASSETTYPES`, but not both.

`ASSETTYPESLIST (Optional)`
(`IList` object) Input list containing one `assettype` column that includes values that restrict the count.

Specify either `ASSETTYPESLIST` and `ASSETS` or `SEARCHSTATE` and `ASSETTYPES`, but not both. If you specify `ASSETTYPESLIST`, `ASSETS` is required. If both parameters are supplied, `ASSETTYPESLIST` is used by default.

`SEARCHSTATE (Optional)`
(`Searchstate` object) Input parameter. `Searchstate` object that contains the criteria to match. The `Searchstate` object describes the search constraints, if any.

Specify either `SEARCHSTATE` and `ASSETTYPES` or `ASSETTYPESLIST` and `ASSET`, but not both. If both parameters are supplied, `ASSETTYPESLIST` and `ASSETS` is used by default.

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

`ASSETS (Optional)`
(IList) Name of a list of assets passed to Content Server that form the asset set. The list of assets is an IList with two columns (`assetid` and `assettype`). This operation is required if you are supplying a list of assets to build the asset set instead of a SearchState. Specify either `ASSETTYPELIST` and `ASSETS` or `ASSETTYPES` and `SEARCHSTATE`, but not both. If both parameters are supplied, `ASSETS` is used by default.

Returns

List with a column named `value` that contains values for the specified attribute for all flex assets in the created assetset.

Example

The following code creates a SearchState, passes the SearchState to the `GetAttributeValues` operation, and returns values of the attributes.

```
AssetsetService service = (AssetsetService)new
AssetsetService_Impl();
AssetsetPortType port = service.getAssetsetPort();

Searchstate ss = new Searchstate();

IList myList = port.ASGetAttributeValues("user_author", "demo",
"PAttributes", "FundFamily", null, "Products", null, ss , null,
null);
```

ASGetMultipleValues

Returns lists of values for specified attributes for assets that match criteria included in the `SEARCHSTATE` parameter.

Syntax

```
ASGetMultipleValues(java.lang.String authusername,
java.lang.String authpassword,
com.FatWire.IList LIST,
java.lang.String BYASSET,
java.lang.String PREFIX, java.lang.String ASSETTYPES,
com.FatWire.IList ASSETTYPESLIST,
com.FatWire.Searchstate SEARCHSTATE,
java.lang.String LOCALE,
com.FatWire.IList ASSETS)
```

Parameters

`authusername (required)`
: (String) Name of the user to log in, as registered in Content Server.

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

`LIST (required)`
: (`IList` object) Input parameter. Name of the list that determines sort order. This is an object that describes how the returned assets are to be sorted.

  The list has three columns:

  `attributetypename`

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

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

BYASSET (required)
: (Boolean) Either TRUE or FALSE. The value indicates whether to group and create lists for each attribute for each individual asset in the assetset (TRUE), or to group all the attribute values for all assets together into one list per attribute (FALSE).

PREFIX (required)
: (String) String to prefix to the named of each list returned.

ASSETTYPES (Optional)
: (String) Input parameter. Name of a list of asset types to include in building the asset count. Comma separated list of flex asset types to match. If null, then all assets in the system are considered. Specify either ASSETTYPESLIST and ASSETS or SEARCHSTATE and ASSETTYPES, but not both.

ASSETTYPESLIST (Optional)
: (IList object) Input list containing one assettype column that includes values that restrict the count.

  Specify either ASSETTYPESLIST and ASSETS or SEARCHSTATE and ASSETTYPES, but not both. If you specify ASSETTYPESLIST, ASSETS is required. If both parameters are supplied, ASSETTYPESLIST is used by default.

SEARCHSTATE (Optional)
: (Searchstate object) Input parameter. Searchstate object that contains the criteria to match. The Searchstate object describes the search constraints, if any.

  Specify either SEARCHSTATE and ASSETTYPES or ASSETTYPESLIST and ASSET, but not both. If both parameters are supplied, ASSETTYPESLIST and ASSETS is used by default.

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

ASSETS (Optional)
: (IList) Name of a list of assets passed to Content Server that form the asset set. The list of assets is an IList with two columns (assetid and assettype). This operation is required if you are supplying a list of assets to build the asset set instead of a SearchState. Specify either ASSETTYPELIST and ASSETS or ASSETTYPES and SEARCHSTATE, but not both. If both parameters are supplied, ASSETS is used by default.

Returns

Lists of attribute values in a column named value.

Example

The following code creates a SearchState, passes the SearchState and list objects to the GetMultipleValues operation, and returns names and values of the attributes:

```
AssetsetService service = (AssetsetService)new
AssetsetService_Impl();
AssetsetPortType port = service.getAssetsetPort();
IList inList = new IList();

String colName[] = {"attributename", "attributetypename",
"direction" };
```

```
inList.setColName(colName);

String item1[] = {"Name", "PAttributes", "ascending"};
String item2[] = {"FundType", "PAttributes", "ascending"};

StringRowsType items[] = new StringRowsType[2];
items[0] = new StringRowsType();
items[1] = new StringRowsType();

items[0].setItem(item1);
items[1].setItem(item2);

inList.setStringRow(items);
Searchstate ss = new Searchstate();
ss.setOP("and");
IList outList = port.ASGetMultipleValues("user_author", "demo",
inList, "true", "th", "Products", null, ss, null, null);
```

Miscellaneous Operations

Miscellaneous operations cover basic Content Server functions such as returning a blob stored in a particular table. Separate operations return blobs for standard assets and flex assets.

The Miscellaneous group comprises the following operations:

- MiscGetBlob
- MiscGetMungoBlob
- MiscSearch

MiscGetBlob

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

Syntax

```
miscGetBlob(java.lang.String authusername,
java.lang.String authpassword,
java.lang.String BLOBHEADER,
java.lang.String BLOBTABLE,
java.lang.String BLOBCOL,
java.lang.String BLOBWHERE,
java.lang.String BLOBKEY)
```

Parameters

`authusername (required)`
: (String) Name of the user to log in, as registered in Content Server.

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

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

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

`BLOBCOL (required)`
: (String) The name of the column that contains the binary data.

`BLOBWHERE (required)`
: (String) Value of the primary key for the row that contains the binary data.

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

Returns

Blob object; for example, a PDF file. The blob is base-64 encoded and wrapped in XML.

Exceptions

None.

Example

The following Java code loads an article identified by object ID and gets text properties and binary properties.

```
MiscService service = (MiscService)new MiscService_Impl();
MiscPortType port = service.getMiscPort();
byte[] blobOUT = port.miscGetBlob("user_author", "demo", null,
"Article", "urlbody", "984156693953", "id");
```

MiscGetMungoBlob

Returns a blob (binary large object) stored as an attribute of type blob for a Content Server flex asset.

Syntax

```
miscGetMungoBlob(java.lang.String authusername,
java.lang.String authpassword, j
ava.lang.String BLOBKEY)
```

Parameters

`authusername (required)`
: (String) Name of the user to log in, as registered in Content Server.

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

`BLOBKEY (Required)`
: (String) ID for the mungoblob flex asset. The ID value comes from an ASGetAttributeValues operation that specifies a flex asset attribute of type `blob`.

Exceptions

Possible error values include:

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

Example

The following code loads and saves a mungo blob (PDF file).

```
MiscService service = (MiscService)new MiscService_Impl();
MiscPortType port = service.getMiscPort();
```

```
byte[] blobOUT = port.miscGetMungoBlob("user_author", "demo",
"1011495632110");
```

See Also

ASGetAttributeValues

MiscSearch

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

Syntax

```
miscSearch(java.lang.String authusername,
java.lang.String authpassword,
java.lang.String INDEX,
java.lang.String WHAT,
java.lang.String QUERYPARSER,
java.lang.String RELEVANCE,
java.lang.String LIMIT,
java.lang.String CHARACTERSET,
java.lang.String SEARCHENGINE)
```

Parameters

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

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

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

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

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

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

`LIMIT (optional)`
: Maximum number of assets to return.

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

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

Description

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

Returns

Returns a list with the following columns:

`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.

errno

The possible values of `errno` include:

Value	Description
-101	No search results.
-800	Bad search type.
-801	Cannot load search engine.
-802	Unsupported search function.
-805	No default index specified.
-806	Unknown search engine.
-809	Search failed.
-810	Bad character set.
-811	Could not call native method.
-812	Index does not exist.

Example

The following code searches for an AltaVista index given a query specified by the `What` parameter and an index. The example assumes that the AltaVista search is configured for article assets:

```
MiscService service = (MiscService)new MiscService_Impl();
MiscPortType port = service.getMiscPort();
IList myList = port.miscSearch("user_author", "demo",
"c:\\JumpStart\\futuretense\\Storage\\sedb\\Article.avx",
```

```
                "description:Burlington", "Simple", null, null, null,
    null);
```

SitePlan Operations

SitePlan operations enable you to examine the page hierarchy of the site. They return the children and properties of any node in the hierarchy.

The SitePlan group comprises the following operations:

- SPGetChildren
- SPGetProperties

SPGetChildren

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

Syntax

```
SPGetChildren(java.lang.String authusername,
java.lang.String authpassword,
java.lang.String NODEID,
java.lang.String CODE,
java.lang.String CHILDTYPE,
java.lang.String CHILDID,
java.lang.String ORDER)
```

Parameters

`authusername (required)`
(String) Name of the user to log in, as registered in Content Server.

`authpassword(required)`
(String) Password for the associated user name.

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

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

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

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

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

`ORDER (optional)`

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

The fields to sort the list by, and whether the sort result on those fields is `ascending` or `descending`. By default, the sort is `ascending`. If you specify more than one field, separate the field names with a comma.

For example, if you specify `ORDER="nrank"`, the list is sorted by rank starting at number *1*. If you want the list sorted by descending rank, use `ORDER="nrank desc"`.

Description

This operation queries the SitePlanTree table for a list of the child nodes of the node that you specify, listing each child with a value for all of the fields from that table (`nid`, `nparentid`, `nrank`, `otype`, `oid`, and `ncode`).

You can restrict the list of children nodes by association name (`CODE`), `CHILDTYPE`, `CHILDID`, and rank (`ORDER`).

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

Returns

A list of SitePlan child nodes.

errno

The possible values of `errno` include:

Value	Description
-111	The asset has no children.
-10004	A required parameter is missing.
-10005	The requested object is not in the object pool (is not loaded into memory).
-10006	The object ID is not valid.
-10007	The version of the object is not valid.

Example

This following example code logs in the user, extracts the child pages for the page asset identified as the node ID (`nid`) of the page:

```
SitePlanService service = (SitePlanService)new
SitePlanService_Impl();
SitePlanPortType port = service.getSitePlanPort();
IList myList = port.SPGetChildren("user_author", "demo",
"968685129229", null, null, null, null);
```

SPGetProperties

Retrieves the properties for the specified node ID.

Syntax

```
SPGetProperties(java.lang.String authusername,
java.lang.String authpassword,
java.lang.String NODEID)
```

Parameters

`authusername (required)`
(String) Name of the user to log in, as registered in Content Server.

`authpassword(required)`
(String) Password for the associated user name.

`NODEID (Required)`
(String) ID number of the parent SitePlan node. Content Server generates the node ID.

Description

Retrieves the properties for the specified node ID.To get the NodeID, first call the AssetLoad operation to retrieve the asset, and then call the AssetGetSiteNode operation to extract the node ID number.

Returns

A list that contains the SitePlan node properties.

errno

The possible values of `errno` include:

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

Example

This code extracts the child pages for the page asset identified as the node ID of the page:

```
SitePlanService service = (SitePlanService)new
SitePlanService_Impl();
SitePlanPortType port = service.getSitePlanPort();
IList myList = port.SPGetProperties("user_author", "demo",
"968685129229");
```

Chapter 3

Objects

This chapter describes input objects to web services operations. It also lists the methods that can be called on them.

The following objects are inputs to the AssetSet Operations:

- IList Objects
- SearchState Objects

Note

Syntax for web services operations and client-code examples are provided in Java. The exact syntax you use will depend on the programming language in which you code your client program.

Regardless of the tool and language you use to generate the client, parameter names for required inputs and the order in which they are passed remain the same, and the output is always SOAP (simple object access protocol). The examples in this chapter were generated using the Apache Axis tool.

SearchState Objects

The `SearchState` object is an optional input to any of the AssetSet Operations defined in the `AssetSet.wsdl` file. Searchstate methods, which reside inside classes generated by your client program, create the `SearchState` object and instantiate and populate SearchState constraint objects.

SearchStates and the AssetSet operations apply to flex assets only. AssetSet operations accept either a SearchState or an IList as input, but not both.

The SearchState group comprises the following objects:

- LikeConstraint
- NestedConstraint
- RangeConstraint
- RichTextConstraint
- SearchState
- StandardConstraint

The SearchState object must be instantiated first because it contains the rest of the SearchState methods.

LikeConstraint

A `LikeConstraint` object is an input to a `SearchState` object. LikeConstraint methods instantiate the `LikeConstraint` object and set parameters.

Methods

`new LikeConstraint()`
: Constructor method that instantiates the `LikeConstraint` object and creates methods that can be called on it. In turn, the `LikeConstraint` object can be added to the `SearchState` object.

`setBUCKET(java.lang.String BUCKET)`
: Sets the value for the `BUCKET` parameter. This method has a corresponding `get` method.

`setTYPENAME(java.lang.String TYPENAME)`
: Sets the value for the `TYPENAME` parameter. This method has a corresponding `get` method.

`setATTRIBUTE(java.lang.String ATTRIBUTE)`
: Sets the value for the `ATTRIBUTE` parameter. This method has a corresponding `get` method.

`setIMMEDIATEONLY(java.lang.String IMMEDIATEONLY)`
: Sets the value for the `IMMEDIATEONLY` parameter. There is a corresponding `get` method for this method.

`setLIST(com.FatWire.IList LIST)`
: Sets the value for the `LIST` parameter. This method has a corresponding `get` method.

`setCASEINSENSITIVE(java.lang.String CASEINSENSITIVE)`
: Sets the value specified by the `CASEINSENSITIVE` parameter. This method has a corresponding `get` method.

Parameters

The following parameters can be set using a corresponding LikeConstraint method:

`BUCKET (optional)`
: Input parameter. The bucket name. If not specified, the attribute name is used.

`TYPENAME (optional)`
: Input parameter. The internal asset name for the attribute (either CAttributes for content attribute, or PAttributes for product attribute). If you do not specify `TYPENAME`, a value is supplied from a property in the `gator.ini` property file: `mwb.defaultattributes=PAttributes.` The default is PAttributes and the value may be changed.

`ATTRIBUTE (required)`
: Input parameter. Name of the attribute to constrain.

`LIST (optional)`
: Input parameter. A list of the constrained values for the attribute. If specified, one or more of the values must match the attribute for a product to meet the constraint. The default is that all assets that have any value for the attribute match the constraint. The column is called `value`.

`IMMEDIATEONLY (optional)`
: Input parameter. A Boolean value: `true` indicates that the search is limited to values directly associated with the specified attribute; `false` (the default) extends the search to include values inherited from a parent.

`CASEINSENSITIVE (optional)`
: Input parameter. A Boolean value: `true` indicates that the comparison is case-insensitive; `false` (the default) considers case in the comparison.

Description

Associated methods set parameter values that populate the `LikeConstraint` object. The constraint, which is similar to a database `LIKE` operation, accepts wild cards; for example, `%`. If the attribute name is already in the SearchState, then the new constraint replaces the old constraint.

The `LikeConstraint` object can be added to the `SearchState` object with the correponding SearchState methods.

Example

This code instantiates a LikeConstraint object and sets parameters for it:

```
LikeConstraint like_cons = new LikeConstraint();
like_cons.setTYPENAME("PAttributes");
like_cons.setATTRIBUTE("FundFamily");
```

See Also

Methods for adding various constraint objects to a SearchState. These are available when you instantiate the SearchState object.

NestedConstraint

RangeConstraint

RichTextConstraint

StandardConstraint

NestedConstraint

A `NestedConstraint` object is an input to a `SearchState` object. NestedConstraintConstraint methods instantiate the `NestedConstraint` object and set different parameters.

Methods

`new NestedConstraint()`
: Constructor method that instantiates the `NestedConstraintConstraint` object and creates methods that can be called on it. In turn, the `NestedConstraintConstraint` object can be added to the `SearchState` object.

`setBUCKET(java.lang.String BUCKET)`
: Sets the value for the `BUCKET` parameter. This method has a corresponding `get` method.

`setSEARCHSTATE(com.FatWire.Searchstate SEARCHSTATE`
: Sets the value for the `SEARCHSTATE` parameter. This method has a corresponding `get` method.

Parameters

`BUCKET(required)`
: Input parameter. The bucket name. If not specified, the attribute name is used.

`SEARCHSTATE (required)`
: Input parameter. Name of the SearchState object to nest inside of the object specified by `NAME`.

Description

Associated methods set parameter values that populate the `NestedConstraint` object. The `NestedConstraint` object can be added to the `SearchState` object with the correponding SearchState methods.

Example

This code instantiates the `NestedConstraint` object and sets the `BUCKET` parameter:

```
NestedConstraint nest_cons = new NestedConstraint();
nested_cons.setBUCKET("PAttributes");
```

See Also

Methods for adding various constraint objects to a SearchState. These are available when you instantiate the SearchState object.

LikeConstraint

RangeConstraint

RichTextConstraint

StandardConstraint

RangeConstraint

A `RangeConstraint` object is an input to a `SearchState` object. RangeConstraint methods instantiate the `RangeConstraint` object and set different parameters.

Methods

`new RangeConstraint( )`
Constructor method that instantiates the `RangeConstraint` object and creates methods that can be called on it. In turn, the `RangeConstraint` object can be added to the `SearchState` object.

`setBUCKET(java.lang.String BUCKET)`
Sets the value for the `BUCKET` parameter. This method has a corresponding `get` method.

`setTYPENAME(java.lang.String TYPENAME)`
Sets the value for the `TYPENAME` parameter. This method has a corresponding `get` method.

`setATTRIBUTE(java.lang.String ATTRIBUTE)`
Sets the value for the `ATTRIBUTE` parameter. This method has a corresponding `get` method.

`setLOWEREQUAL(java.lang.String LOWEREQUAL)`
Sets the value for the `LOWEREQUAL` parameter. This method has a corresponding `get` method.

`setLOWER(java.lang.String LOWER)`
Sets the value for the `LOWER` parameter. This method has a corresponding `get` method.

`setUPPEREQUAL(java.lang.String UPPEREQUAL)`
Sets the value for the `UPPEREQUAL` parameter. This method has a corresponding `get` method.

`setUPPER(java.lang.String UPPER)`
Sets the value for the `UPPER` parameter. This method has a corresponding `get` method.

`setCASEINSENSITIVE(java.lang.String CASEINSENSITIVE)`
Sets the value specified by the `CASEINSENSITIVE` parameter. This method has a corresponding `get` method.

Parameters

The following parameters can be set using a corresponding LikeConstraint method:

`BUCKET (optional)`
Input parameter. The bucket name. If not specified, the attribute name is used.

`TYPENAME (optional)`
Input parameter. The internal asset name for the attribute (either `CAttributes` for content attribute, or `PAttributes` for product attribute). If you do not specify `TYPENAME`, a value is supplied from a property in the `gator.ini` property file: `mwb.defaultattributes=PAttributes.` The default is PAttributes and the value may be changed.

`ATTRIBUTE (required)`
Input parameter. Name of the attribute to constrain.

`LOWER | LOWEREQUAL (required)`
Input parameter. The bottom end of the range.

`UPPER | UPPEREQUAL (required)`
Input parameter. The top end of the range.

`CASEINSENSITIVE (optional)`
Input parameter. A Boolean value: `true` indicates that the comparison is case-insensitive; `false` (the default) considers case in the comparison.

Description

Associated methods set parameter values that populate the `RangeConstraint` object. The `RangeConstraint` object can be added to the `SearchState` object with the correponding SearchState methods.

Example

This code instantiates the `RangeConstraint` object and sets the `TYPENAME` and `ATTRIBUTE` parameters.

```
RangeConstraint range_cons = new RangeConstraint();
range_cons.setTYPENAME("PAttributes");
range_cons.setATTRIBUTE("FundFamily");
```

See Also

Methods for adding various constraint objects to a SearchState. These are available when you instantiate the SearchState object.

LikeConstraint

NestedConstraint

RichTextConstraint

StandardConstraint

RichTextConstraint

The `RichTextConstraint` object is an input to a `SearchState` object. RichTextConstraint methods instantiate the `RichTextConstraint` object and set different parameters.

Methods

`new RichTextConstraint()`
Constructor method that instantiates the `RichTextConstraint` object and creates methods that can be called on it. In turn, the `RichTextConstraint` object can be added to the `SearchState` object.

`setBUCKET(java.lang.String BUCKET)`
Sets the value for the `BUCKET` parameter. This method has a corresponding `get` method.

`setTYPENAME(java.lang.String TYPENAME)`
Sets the value for the `TYPENAME` parameter. This method has a corresponding `get` method.

`setATTRIBUTE(java.lang.String ATTRIBUTE)`
Sets the value for the `ATTRIBUTE` parameter. This method has a corresponding `get` method.

`setVALUE(java.lang.String VALUE)`
Sets the value for the `VALUE` parameter. This method has a corresponding `get` method.

`setPARSER(java.lang.String PARSER)`
Sets the value for the `PARSER` parameter. This method has a corresponding `get` method.

`setCONFIDENCE(java.lang.String CONFIDENCE)`
Sets the value for the `CONFIDENCE` parameter. This method has a corresponding `get` method.

`setMAXCOUNT(java.lang.String MAXCOUNT)`
Sets the value for the `MAXCOUNT` parameter. This method has a corresponding `get` method.

Parameters

`BUCKET (optional)`
Input parameter. The bucket name. If not specified, the attribute name is used.

`TYPENAME (optional)`
Input parameter. The internal asset name for the attribute (either `CAttributes` for content attribute, or `PAttributes` for product attribute). If you do not specify `TYPENAME`, a value is supplied from a property in the `gator.ini` property file: `mwb.defaultattributes=PAttributes`. The default is PAttributes and the value may be changed.

`ATTRIBUTE (required)`
Input parameter. Name of the attribute to constrain.

`VALUE (required)`
Input parameter. The rich-text search criteria, which should apply to the attribute.

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

`CONFIDENCE (required)`
Input parameter. The minimum confidence level for the match. This parameter is search engine dependent; adjust the value lower if you are not getting the desired results.

`MAXCOUNT (optional)`
Input parameter. The maximum number of answers desired for the match. If this parameter is not specified, the number of results is limited only by the confidence and the number of products.

Description

Associated methods set parameter values that populate the `RichTextConstraint` object. The `RichTextConstraint` object can be added to the `SearchState` object with the correponding SearchState methods.

Adds an index name and rich-text expression to the list of rich-text criteria for items. If the attribute name is already mentioned as part of a rich-text constraint in the SearchState, then the existing constraint is removed first.This tag requires that the attribute was created as indexed.

Example

This code instantiates a `RichTextConstraint` object and set the `TYPENAME` and `ATTRIBUTE` parameters.

```
RichTextConstraint richtext_cons = new RichTextConstraint();
richtext_cons.setTYPENAME("PAttributes");
richtext_cons.setATTRIBUTE("FundFamily");
```

See Also

Methods for adding various constraint objects to a SearchState. These are available when you instantiate the SearchState object.

LikeConstraint

NestedConstraint

RangeConstraint

StandardConstraint

SearchState

A SearchState is an input to the AssetSet operations. SearchState methods instantiate the SearchState object and add different search constraints to it. There are corresponding `get` methods for all the `set` methods described for this object.

Methods

`new SearchState( )`
: Constructor method that instantiates the SearchState object. Creates an empty `SearchState` object, and provides methods for creating search constraints. Once the `SearchState` object is instantiated, you can call associated SearchState methods on it.

`setOP (OP)`
: Specifies the operation to be applied to the `SearchState` object. Adds a search constraint that will be appended as an `AND` operation or an `OR` operation to other constraints contained in the Searchstate instance.

`setLikeConstraint(com.FatWire.LikeConstraint[] likeConstraint)`
: Passes a `LikeConstraint` object to a `SearchState` object. This method has a corresponding `get` method.

`setNestedConstraint(com.FatWire.NestedConstraint[] nestedConstraint)`
: Passes a `NestedConstraint` object to a `SearchState` object. This method has a corresponding `get` method.

`setRangeConstraint(com.FatWire.RangeConstraint[] rangeConstraint)`
: Passes a `RangeConstraint` object to a `SearchState` object. This method has a corresponding `get` method.

`setRichTextConstraint(com.FatWire.RichTextConstraint[] richTextConstraint)`
: Passes a `RichTextConstraint` object to a `SearchState` object. This method has a corresponding `get` method.

`setStandardConstraint(com.FatWire.StandardConstraint[] standardConstraint)`

Passes a `StandardConstraint` object to a `SearchState` object. This method has a corresponding `get` method.

Parameters

`OP (optional)`

Input parameter for the `setOP` method. SearchState type, either `AND` or `OR`. The default is `AND`.

Description

Methods update the named object or SearchState to include the new constraint. If the attribute is already in the SearchState, then the new constraint replaces the old constraint.

Example

This code instantiates a `SearchState` object called `SS`, specifies an `AND` operation, and adds a standard constraint:

```
Searchstate ss = new Searchstate();
ss.setOP("and");
ss.setStandardConstraint(stand_cons);
```

See Also

Methods for adding the following constraint objects to a SearchState.

LikeConstraint

NestedConstraint

RangeConstraint

RichTextConstraint

StandardConstraint

StandardConstraint

A `StandardConstraint` object is an input to a SearchState object. StandardConstraint methods instantiate the `StandardConstraint` object and set different parameters.

Methods

`new StandardConstraint()`

Constructor method that instantiates the `StandardConstraint` object and creates methods that can be called on it. In turn, the `StandardConstraint` object can be added to the `SearchState` object.

`setBUCKET(java.lang.String BUCKET)`

Sets the value for the `BUCKET` parameter. This method has a corresponding `get` method.

`setTYPENAME(java.lang.String TYPENAME)`

Sets the value for the `TYPENAME` parameter. This method has a corresponding `get` method.

`setATTRIBUTE(java.lang.String ATTRIBUTE)`
Sets the value for the `ATTRIBUTE` parameter. This method has a corresponding `get` method.

`setIMMEDIATEONLY(java.lang.String IMMEDIATEONLY)`
Sets the value for the `IMMEDIATEONLY` parameter. This method has a corresponding `get` method.

`setLIST(com.FatWire.IList LIST)`
Sets the value for the `LIST` parameter. This method has a corresponding `get` method.

`setCASEINSENSITIVE(java.lang.String CASEINSENSITIVE)`
Sets the value specified by the `CASEINSENSITIVE` parameter. This method has a corresponding `get` method.

Parameters

`BUCKET (optional)`
Input parameter. The bucket name. If not specified, the attribute name is used.

`TYPENAME (optional)`
Input parameter. The internal asset name for the attribute (either `CAttributes` for content attribute, or `PAttributes` for product attribute). If you do not specify `TYPENAME`, a value is supplied from a property in the `gator.ini` property file: `mwb.defaultattributes=PAttributes`. The default is PAttributes and the value may be changed.

`ATTRIBUTE (required)`
Input parameter. Name of the attribute to constrain.

`LIST (optional)`
Input parameter. A list of the constrained values for the attribute. If specified, one or more of the values must match the attribute for a product to meet the constraint. The default is that all assets that have any value for the attribute match the constraint. The list has a single column called `value`.

`IMMEDIATEONLY (optional)`
Input parameter. A Boolean value: `true` indicates that the search is limited to values directly associated with the specified attribute; `false` (the default) extends the search to include values inherited from a parent.

`CASEINSENSITIVE (optional)`
Input parameter. A Boolean value: `true` indicates that the comparison is case-insensitive; `false` (the default) considers case in the comparison.

Description

Associated methods set parameter values that populate the `StandardConstraint` object. The `StandardConstraint` object can be added to the `SearchState` object with one of the corresponding SearchState methods.

Example

This code instantiates a `StandardConstraint` object and sets the `TYPENAME` and `ATTRIBUTE` parameters.

```
StandardConstraint stand_cons = new StandardConstraint();
stand_cons.setTYPENAME("PAttributes");
stand_cons.setATTRIBUTE("FundFamily");
```

See Also

Methods for adding various constraint objects to a SearchState. These are available when you instantiate the SearchState object.

LikeConstraint

NestedConstraint

RangeConstraint

RichTextConstraint

IList Objects

The `IList` object is an optional input to any of the AssetSet Operations defined in the `AssetSet.wsdl` file. Other supporting objects are inputs to the `IList` object. IList methods, which reside inside the IList class generated by your client program, are used to create support objects that populate the `IList` object.

ILists and the AssetSet Operations apply to flex assets only. AssetSet operations accept either an IList or a SearchState as input, but not both.

IList comprises the following objects:

- IList
- StringRowsType
- URLRowsType
- URLType

IList

An `IList` object is an array that contains arrays of rows and columns (of either type string or type URL). The component row and column arrays of the larger `IList` are constructed by supporting methods. Associated IList methods instantiate the IList object and add row and column arrays to it. When populated, the IList object is an input to the AssetSet Operations.

Methods

`new IList( )`
: Constructor method that instantiates the `IList` object. Creates an empty `IList` object, and provides methods for creating arrays of rows and columns. Once the `IList` object is instantiated, you can call associated IList methods on it.

`setColName(java.lang.String[] colName)`
: This method has a corresponding `get` method.

`setUrlColName(java.lang.String[] urlColName)`
: This method has a corresponding `get` method.

`setStringRow(com.FatWire.StringRowsType[] stringRow)`
: This method has a corresponding `get` method.

`setUrlRow(com.FatWire.UrlRowsType[] urlRow)`
: This method has a corresponding `get` method.

Example

The following code creates an IList using a `StringRowsType` object.

```
IList inList = new IList();
String colName[] = {"attributename", "attributetypename",
"direction" };
inList.setColName(colName);
String item1[] = {"Name", "PAttributes", "ascending"};
String item2[] = {"FundType", "PAttributes", "ascending"};
StringRowsType items[] = new StringRowsType[2];
items[0] = new StringRowsType();
```

```
items[1] = new StringRowsType();
items[0].setItem(item1);
items[1].setItem(item2);
inList.setStringRow(items);
```

See Also

StringRowsType

URLRowsType

URLType

StringRowsType

A `StringRowsType` object is an array of rows, of type string, that is input to an `IList` object. Associated methods construct a single component-row array as an object of type `StringRowsType`. The populated `IList` object is, in turn, an input to the AssetSet Operations.

Methods

`new StringRowsType( )`

Constructor method that instantiates the `StringRowsType` object. Creates an empty `StringRowsType` object, and provides methods for creating arrays of rows. Once the `StringRowsType` object is instantiated, you can call the associated `setItem` method on it.

`setItem(java.lang.String[] item)`

Sets arrays of strings in the `StringRowsType` object. This method has a corresponding `get` method.

Example

The following code creates an IList using a `StringRowsType` object.

```
IList inList = new IList();
String colName[] = {"attributename", "attributetypename",
"direction" };
inList.setColName(colName);
String item1[] = {"Name", "PAttributes", "ascending"};
String item2[] = {"FundType", "PAttributes", "ascending"};
StringRowsType items[] = new StringRowsType[2];
items[0] = new StringRowsType();
items[1] = new StringRowsType();
items[0].setItem(item1);
items[1].setItem(item2);
inList.setStringRow(items);
```

See Also

IList

URLRowsType

A `URLRowsType` object is an array of rows, of type URL, that is input to an `IList` object. Associated methods construct a single component-row array as an object of type `StringRowsType`. The populated `IList` object is, in turn, an input to the AssetSet Operations. You should not need to set this object as input, but because it appears in the WSDL file and client code generated from the WSDL file, it is described here for completeness.

Methods

`new URLRowsType( )`

Constructor method that instantiates the `URLRowsType` object. Creates an empty `URLRowsType` object, and provides methods for creating arrays of rows of type URL. Once the `StringRowsType` object is instantiated, you can call the associated `setURLstruct` method on it.

`setUrlstruct(com.FatWire.URLType[] urlstruct)`

Sets an array of URL type objects. This method has a corresponding `get` method.

See Also

IList

URLType

Creates a new `URLType` object and methods that can be called on that object. You should not need to set this object as input, but because it appears in the WSDL file and client code generated from the WSDL file, it is described here for completeness.

Methods

`new URLType( )`

Constructor method that instantiates the `URLType` object. Creates an empty `URLRowsType` object, and provides methods for creating URL pointers. Once the `URLType` object is instantiated, you can call the associated `set` methods on it.

`setUrlfile(java.lang.String urlfile)`

Input string. `urlfile` is a file name. This method has a corresponding `get` method.

`setUrlvalue(byte[] urlvalue)`

Byte array that corresponds to the file set in the `setUrlFile` method. Accepts a byte array (binary) read from the file named `urlfile`. This method has a corresponding `get` method.

See Also

IList

Index

S

W