



ICEcoreTM
Discover Cool Collaboration

ICEcore Web Services Guide

ICEcore Version 1.0



Legal Notices

SiteScape, Inc., makes no representations or warranties with respect to the contents or use of this documentation, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, SiteScape, Inc., reserves the right to revise this publication and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes.

Further, SiteScape, Inc., makes no representations or warranties with respect to any software, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, SiteScape, Inc., reserves the right to make changes to any and all parts of SiteScape software, at any time, without any obligation to notify any person or entity of such changes.

Any products or technical information provided under this Agreement may be subject to U.S. export controls and the trade laws of other countries. You agree to comply with all export control regulations and to obtain any required licenses or classification to export, re-export or import deliverables. You agree not to export or re-export to entities on the current U.S. export exclusion lists or to any embargoed or terrorist countries as specified in the U.S. export laws. You agree to not use deliverables for prohibited nuclear, missile, or chemical biological weaponry end uses.

Copyright © 2008 SiteScape, Inc. All rights reserved. No part of this publication may be reproduced, photocopied, stored on a retrieval system, or transmitted without the express written consent of the publisher.

SiteScape, Inc., has intellectual property rights relating to technology embodied in the product that is described in this document.

SiteScape, Inc.
12 Clock Tower Place, Suite 210
Maynard, MA 021754
U.S.A.
www.SiteScape.com

Third-Party Materials

All third-party trademarks are the property of their respective owners.

Images of personnel in the screen shots supplied by Copyright © Comtech Enterprises, Inc. (<http://www.comteche.com>).

Contents

About This Manual	5
1 Web Services Overview	7
1.1 ICEcore Web Services Implementation	7
1.1.1 Sample Clients	8
1.1.2 ICEcore-Specific Terminology	9
1.2 Connecting to the Server	10
1.2.1 Notes About Security	12
1.3 Sending Messages	12
1.4 Other Helpful Source Files	13
1.5 Notes about Specific Messages	13
1.5.1 Adding Folders and the Binder Configuration Identifier	13
1.5.2 Adding Entries and the Definition Identifier	15
1.5.3 Tips for All Messages that Add and Modify	15
1.5.4 Attaching Files	16
1.5.5 Fetching Attachments	17
1.5.6 Adding Calendar Entries	17
1.5.7 Binder Pages and getWorkspaceTreeAsXML	17
1.6 Extending ICEcore Web Services	19
2 Message Reference	23
addFolder	25
addFolderEntry	26
getAllPrincipalsAsXML	28
getDefinitionAsXML	29
getDefinitionConfigAsXML	30
getFolderEntriesAsXML	31
getFolderEntryAsXML	32
getPrincipalAsXML	33
getTeamMembersAsXML	34
getTeamsAsXML	35
getWorkspaceTreeAsXML	36
modifyFolderEntry	38
uploadCalendarEntries	39
uploadFolderFile	40

About This Manual

This manual provides reference information for programmers who want to access ICEcore from another program (as opposed to the user interface presented in a web browser). To accomplish this task, programmers use the ICEcore Web services.

Audience

This guide is intended for programmers.

Software and Documentation Version

This manual describes features in ICEcore Version 1.0.2. This is Version 1.0 of this manual.

Contents of this Manual

This manual provides information about the following:

- ♦ Overview of ICEcore Web services
- ♦ Alphabetized list of Web services messages

Conventions

This manual uses the conventions described in the following paragraphs and table.

A greater-than symbol (>) is used to separate actions within a step and items in a cross-reference path.

A trademark symbol (®, ™, etc.) denotes a SiteScope trademark. An asterisk (*) denotes a third-party trademark.

When a single pathname can be written with a backslash for some platforms or a forward slash for other platforms, the pathname is presented with a backslash. Users of platforms that require a forward slash, such as Linux or UNIX, should use forward slashes as required by your software.

What you see	What it means
Click the <i>Add a team workspace</i> button.	Items that are clickable on the page, programming variables, or syntax parameters are presented in italic font.
Click the <i>Getting Started</i> link.	
Blog summary - Provides a....	Defined terms in a list, note headers, section headers on a reference page, and list items on a reference page are presented in bold font.
Note: Remember that....	
Type <code>status</code> , then press Enter.	Text that you must type, file names, commands, command options, routines, Web services messages, and parameters are presented in Courier font when occurring in a body of text.
Open the <code>ManagerGuide.pdf</code> file.	
Use the <code>open_db</code> routine with its <code>lock</code> parameter.	
[page]	Optional syntax parameters are enclosed in brackets ([]).

What you see	What it means
..., paramSyntax1 paramSyntax2,...	Required parameters that accept two or more optional syntaxes are separated by the vertical-line character.
(V1+)	The version of ICEcore that supports the Web services operation ("Version 1.0 or later")

Additional Documentation

You can find more information in the ICEcore documentation, which is accessible from links within the ICEcore software:

- ♦ ICEcore Help Mode
- ♦ ICEcore Quick Start Guide
- ♦ ICEcore User Guide
- ♦ ICEcore Installation and Configuration Guide
- ♦ ICEcore Administration Guide

Web Services Overview

1

SiteScape offers a set of messages that you can use in client programs to exchange information with the server (a running installation of ICEcore). This chapter includes these sections:

- ♦ [Section 1.1, “ICEcore Web Services Implementation,” on page 7](#)
- ♦ [Section 1.2, “Connecting to the Server,” on page 10](#)
- ♦ [Section 1.3, “Sending Messages,” on page 12](#)
- ♦ [Section 1.5, “Notes about Specific Messages,” on page 13](#)
- ♦ [Section 1.6, “Extending ICEcore Web Services,” on page 19](#)

1.1 ICEcore Web Services Implementation

ICEcore implements Java Web services, which provide a set of messages and parameters that client programs can use to exchange information with ICEcore. The [alphabetized reference section of messages](#) in this manual provides syntax and examples. In addition, you can access the ICEcore Web Services Description Language (WSDL) file here:

```
http://localhost:8080/ssf/ws/Facade?wsdl
```

Replace the `localhost` specification with the host and port for your ICEcore installation.

NOTE: ICEcore does not currently publish its WSDL file using Universal Description, Discovery, and Integration (UDDI) or the Web Services Inspection Language (WSIL). Use the [alphabetized reference section of messages](#) in this manual or the URL-generated WSDL file to understand the ICEcore message interface.

For ICEcore Version 1.0, regarding messages shown in the product source code, the `search` message is under development and subject to change or deletion at any time. Do not use this message in your client applications.

To implement lower-level details of its Web service calls, ICEcore uses methods included in the Apache Axis toolkit. As a very brief summary, ICEcore code calls Apache Axis methods. These methods convert the message to Simple Object Access Protocol (SOAP) calls. SOAP calls use Remote Procedure Calls (RPC) to call ICEcore Java methods. Conversion of information to and from EXtensible Markup Language (XML) is a part of this process.

NOTE: When presenting Web service examples in this manual, the code includes Apache Axis methods. If you are not using the Apache Axis toolkit, map the Axis methods to the ones provided by your vendor's Web services toolkit.

SiteScape highly recommends using the supporting code for the `wsclient.bat` sample as a template for constructing your own Web service calls. This sample client is described in [the next section](#).

When your application sends a message and its parameters, ICEcore returns either a string of XML data or a number. Messages that return a number are often ones that create a new object in ICEcore (for example, a folder or an entry), and the number uniquely identifies the newly created object.

The XML returned by ICEcore is free form and does not have a schema. When using ICEcore Web services, you need to read and intuit the XML elements in order to code your application so that it can parse the returned XML.

A subsequent section in this chapter provides tips for obtaining values needed for various message parameters.

This section contains the following subsections:

- ♦ [Section 1.1.1, “Sample Clients,” on page 8](#)
- ♦ [Section 1.1.2, “ICEcore-Specific Terminology,” on page 9](#)

1.1.1 Sample Clients

ICEcore provides sample clients that can assist you in learning how to use its Web services. These sample clients are located here in the ICEcore code base:

```
/ssf/samples/remotingclient
```

Here is a list of sample clients and the first ICEcore version that includes them:

- ♦ **wsclient.bat (V1+)**: This Windows batch file allows you to specify commands and parameters that use the ICEcore Web services. The initial pages of [the chapter containing the alphabetical reference of ICEcore Web service messages](#) provides a table mapping each of the `wsclient.bat` commands to its corresponding Web services message. Regardless of platform, SiteScape recommends that you use the Java code that implements this batch file as a template for making ICEcore Web service calls.
- ♦ **wsExport.bat and wsImport.bat (V1.1+)**: These Windows batch files take data from a portion of the workspace and folder hierarchy and reproduce it on another file system. These tools are not a “true” import and export facility, because they do not retain the workflow states, access-control settings, and history of the original objects.

The following Java file implements the `wsclient.bat` batch file:

```
/ssf/samples/remotingclient/src/com/sitescape/team/samples/remoting/  
client/ws/axis/WSClient.java
```

The following file provides log-in information for `wsclient.bat` (start with the same path as shown in the previous example):

```
.../ws/security/PWCallbackText.java
```

This is another file used by `wsclient.bat` to log in:

```
/ssf/samples/remotingclient/client_deploy.wsdd
```

Enabling `wsclient.bat` (Windows systems only)

Before using `wsclient.bat`, you need to do some work in your build to enable it. Using the Eclipse build environment, click and drag this file into your Ant window:

```
/ssf/samples/remotingclient/build.xml
```

In the Ant window, open the `wsclient` build file, and double click `zip`.

ICEcore adds the `remotingclient-sample.zip` file to the `/ssf/samples` directory.

To use `wsclient.bat`, do the following:

1. Use a command-line window to `cd` to the `/ssf/samples/remotingclient` directory.
2. Type `wsclient.bat`.
3. On the same line, type a `wsclient.bat` command name and its parameters.
Command names are listed at the beginning of the chapter containing the [alphabetized Web services messages](#).
4. Press the Return key.

If the command executes successfully, ICEcore displays the return value in the command-line window.

For example, to see an XML string containing summary data for all workspaces and folders that have defined teams, execute the following command in the `/ssf/samples/remotingclient` directory:

```
> wsclient.bat printTeams
```

If you want to execute the `wsclient.bat` file outside of the scope of the build environment's file system, you can unzip the `/ssf/samples/remotingclient-sample.zip` file elsewhere on a Windows file system, use a command-line window to `cd` to the location of that copy of the `wsclient.bat` file, and use the batch file there.

1.1.2 ICEcore-Specific Terminology

The following are ICEcore-specific definitions that can assist you when using the ICEcore Web services:

- ♦ **binder:** A place, such as a workspace or folder.
- ♦ **binder configuration ID:** A number that identifies the template used to create and configure a new workplace or folder. ([An upcoming section](#) provides additional information about this identifier.)
- ♦ **binder ID:** A unique number that identifies a specific workspace or folder.
- ♦ **data item name:** A hidden-tag value that maps an HTML form element with a value stored in the ICEcore database.
- ♦ **definition ID:** A unique, 32-character hexadecimal identifier that maps to a definition for a specific type of entry. (You modify and create definitions using the designers in the administration portlet.) You need to specify this value when creating a new entry in a folder.
- ♦ **page:** A hierarchical level in the workspace hierarchy that represents a subset of binders, most often used to group personal workspaces into sets that are convenient for display on the screen. ([An upcoming section](#) provides additional information about this hierarchical level.)
- ♦ **principal:** A registered user or a group.
- ♦ **principal ID:** A unique number that identifies a specific user or group.

1.2 Connecting to the Server

To show how a client application connects to the server, this section uses the source code for the `wsclient.bat` batch file to illustrate concepts. Also, the Java source code that enables `wsclient.bat` makes calls to the Apache Axis toolkit methods; if you are not using Apache Axis, map these methods to ones used by your Web services methods.

The `wsclient.bat` batch file is implemented using the methods located in the `WSClient.java` file (as mentioned in [the section describing sample clients](#)). This file contains the `fetch` method, which is a wrapper that performs most of the direct calls to the Apache Axis toolkit methods, which in turn make the call to Web services. In summary, the `fetch` method connects the `wsclient.bat` client to the server, sets up the remote procedure calls, names the operation, and invokes the passing of messages using Web services.

This section describes step-by-step the initial lines of code in the `fetch` method to show how to connect your client to the server. Consider these initial lines of code:

```
// Replace the hostname in the endpoint appropriately.
String endpoint = "http://localhost:8080/ssf/ws/Facade";
```

The `wsclient.bat` file is designed to be executed on the same machine that runs the ICEcore installation. Many client applications connect to the server over the Internet. Provide the appropriate host and port for the ICEcore server to which you want to connect.

Consider the next lines of code:

```
// Make sure that...client_deploy.wsdd...is accessible to the program.
EngineConfiguration config = new FileProvider("client_deploy.wsdd");
    .
    .
    .
call.setProperty(WSHandlerConstants.USER, "admin");
```

As required by the source code for the client, the `client_deploy.wsdd` file is located in the same directory as the `wsclient.bat` file. It contains XML needed by the Apache Axis to implement security. It establishes a username that a client can use as its persona when working with Web services (by default, it is the `admin` account). It also points to two other files used by Apache Axis as part of the way it implements security:

- ♦ `PWCallbackDigest.java`
- ♦ `PWCallbackText.java`

A subsection that follows provides helpful notes about security choices for your client application.

For example, the `wsclient.bat` client uses the persona of the `admin` ICEcore account. To specify the password for this persona, the `PWCallbackText.java` file contains these lines, which set the password to `test`:

```
if ("admin".equals(id)) {
    pc.setPassword("test");
}
```

The call to the `call.setProperty` method establishes a constant that specifies the `admin` account as being the persona to be used by the client. It is rare that a client uses the same persona for all its work on the server.

Remember that this way of establishing a persona for your client software is specific to Apache Axis. If you are not using Apache Axis, use these examples to inform your work with your Web services toolkit.

These lines of code are the remaining ones required by Apache Axis to set up and then make the call to the server:

```
Service service = new Service(config);

Call call = (Call) service.createCall();

call.setTargetEndpointAddress(new URL(endpoint));

// We are going to invoke the remote operation to fetch the workspace
// or folder to print.
call.setOperationName(new QName(operation));

// Programmatically set the username. Alternatively you can specify
// the username in the WS deployment descriptor client_deploy.wsdd
// if the username is known at deployment time and does not change
// between calls, which is rarely the case in Aspen.
call.setProperty(WSHandlerConstants.USER, "admin");

if(filename != null) {
    DataHandler dhSource = new DataHandler(new FileDataSource(new
File(filename)));

    call.addAttachmentPart(dhSource); //Add the file.

    call.setProperty(Call.ATTACHMENT_ENCAPSULATION_FORMAT,
Call.ATTACHMENT_ENCAPSULATION_FORMAT_DIME);
} //End of if

Object result = call.invoke(args);

.
.
.
```

1.2.1 Notes About Security

Your client application can implement security in several ways. Each method has advantages and disadvantages, which you should consider. This section contains the following subsections, in order from the most secure to the least secure:

- ♦ “Password Digest” on page 12
- ♦ “Password Text” on page 12

Password Digest

On the client side of the Web services transaction, the client code provides a username and password, and the Web services security framework digests the password and sends it to the server.

On the server side, ICEcore retrieves the user’s password from its database and passes it to the Web services security framework.

One of the disadvantages to this method of implementing security is that, because ICEcore stores the password in encrypted form, it cannot pass a clear-text password to the framework for comparison. To solve this problem, before digest and transmission, the client can use the ICEcore password-encryptor class to apply the same encryptor to the clear-text password and can then pass the encrypted password to the security framework.

So, although secure, this method requires work on the client side.

Password Text

On the client side of the Web services transaction, the client code provides a username and password to the Web services security framework, and the framework passes the password as plain text.

On the server side, the security framework allows ICEcore to retrieve the clear-text password from the message using an application programming interface (API) call. Once retrieved, ICEcore applies its internal password encryptor and compares the result with the password stored in the database for the user. ICEcore performs the authentication; if successful, ICEcore returns the clear-text password to the Web services security framework to complete its process.

The primary disadvantage of this method is that it is not secure unless it’s being done over SSL. Another disadvantage is that, if the password does not match, ICEcore must throw an exception in order to cause the security framework to abort its process. (This problem with mismatching passwords does not occur using Password Digest.)

So, this method is easier to code on the client side, but it has several disadvantages.

1.3 Sending Messages

In the code example at the end of the previous section, these are the two lines that are responsible for sending the message to the server:

```
call.setOperationName(new QName(operation));  
.  
.  
.  
Object result = call.invoke(args);
```

The operation parameter maps to the names of the messages documented in [the alphabetical reference](#) in this manual (addFolderEntry, uploadCalendarEntries, and so on). The args parameter maps to the message parameters documented in the reference section.

Using Apache Axis, “hard coded” calls that send messages from the client to the server appear as follows:

```
call.setOperationName(new QName("AddFolder"));
Object result = call.invoke(new Object[] {new Long(21), new Long(146),
new String("My new folder")});
```

1.4 Other Helpful Source Files

For additional information about ICEcore Web services, there are a few additional source files that may assist you in understanding Web services and product data structures.

This file contains most of the methods that implement the facade:

```
/ssf/main/src/com/sitescape/team/remoting/impl/AbstractFacade.java
```

This file implements the portions of the facade that use Apache Axis code, especially code managing the sending and receiving of attachments:

```
/ssf/main/src/com/sitescape/team/remoting/ws/FacadeImpl.java
```

If you want to review the code that builds the workspace tree (along with some helper classes), then you can review this file:

```
/ssf/main/src/com/sitescape/team/web/tree/WsDomTreeBuilder.java
```

1.5 Notes about Specific Messages

The use of some messages is less intuitive than for other messages. This section provides additional information for those messages and includes the following subsections:

- ♦ [Section 1.5.1, “Adding Folders and the Binder Configuration Identifier,” on page 13](#)
- ♦ [Section 1.5.2, “Adding Entries and the Definition Identifier,” on page 15](#)
- ♦ [Section 1.5.3, “Tips for All Messages that Add and Modify,” on page 15](#)
- ♦ [Section 1.5.4, “Attaching Files,” on page 16](#)
- ♦ [Section 1.5.6, “Adding Calendar Entries,” on page 17](#)
- ♦ [Section 1.5.5, “Fetching Attachments,” on page 17](#)
- ♦ [Section 1.5.7, “Binder Pages and getWorkspaceTreeAsXML,” on page 17](#)

1.5.1 Adding Folders and the Binder Configuration Identifier

When adding a folder, you need to specify a binder configuration identifier, which identifies the template used to configure a folder of a particular type. For example, the blog-folder template specifies settings used to configure a new blog folder.

To review the blog-folder template, do the following:

1. Click *Manage workspace and folder templates* in the administration portlet.
2. In the “Currently defined templates” section, click *Blog*.
3. Click *Manage this target > Configure*.

ICEcore displays a page, the top of which appears as follows:

Configure defaults

Current folder: Blog

Blog

Definition inheritance ⓘ

Not inheriting definition settings.

Inherit definitions?

☐ yes ☒ no

Allowed views ⓘ

- ☐ File folder
- ☐ Task
- ☐ Guestbook
- ☐ Milestone folder
- ☒ Blog
- ☐ Photo album
- ☐ File folder
- ☒ Calendar
- ☒ Discussion folder viewed as a list
- ☐ Survey folder
- ☒ Discussion folder viewed as a table
- ☐ Wiki
- ☐ Calendar
- ☒ Wiki

Here is the complete list of the configuration settings available in the template:

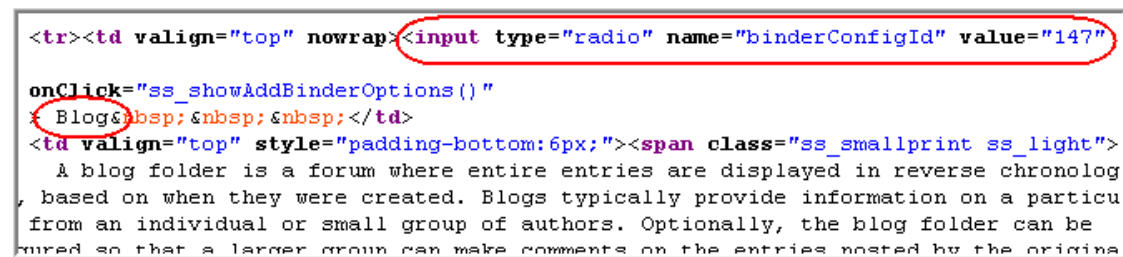
- ♦ Definition inheritance
- ♦ Allowed views
- ♦ Default view
- ♦ Default entry types
- ♦ Workflow associations
- ♦ Allowed workflows

At the time of this writing, ICEcore does not provide a message that you can use to retrieve the binder configuration identifier for a particular type of folder. Instead, to obtain the binder configuration identifier for the folder you want to create, do the following in the UI:

1. View any workspace or folder.
2. Click *Manage > Add folder*.
3. While viewing the “Add new folder” page, use your browser to view the HTML source code for the page.
4. Search for the type of folder you want to create (for example, discussion, blog, calendar, and so on).
5. In the input HTML tag that creates the radio button for that type of folder, note the `name="binderConfigId"` and `value="nnn"` pair of tag elements.

The number specified by the `value` element is the binder configuration identifier of the folder you want to create.

Here is a picture of the binder configuration information for a blog folder, as found in the HTML source for the “Add new folder” page:



1.5.2 Adding Entries and the Definition Identifier

When creating entries, ICEcore requires a definition identifier, which maps to the definition for the type of entry you want to create. You work with definitions when using the designers in the administration portlet.

The easiest way to work with definition identifiers is to specify `null` for this value. When you specify `null`, ICEcore automatically applies the definition identifier for the default entry type of the folder in which you are creating a new entry. For example, by default, say you want to create an entry in a blog folder. If you pass `null` as the definition identifier, ICEcore automatically applies the definition identifier for a blog entry.

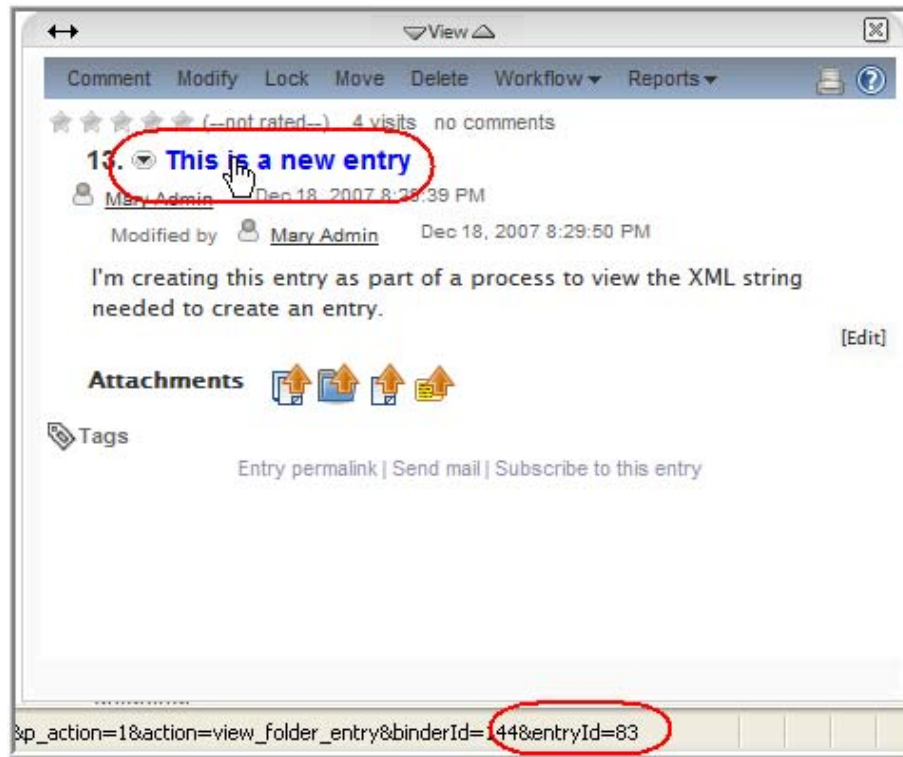
As another option, you can use the `getDefinitionConfigAsXML` message to get information about all definitions. Then, you can parse the XML string for the definition identifier of the type of entry you want.

1.5.3 Tips for All Messages that Add and Modify

When you **add** or **modify** an entry, you must pass a string of XML containing elements for the entry you want to create or modify. Because ICEcore XML does not adhere to a schema, you need to review a copy of a complete entry to understand the structure of the XML string that you need to specify as a parameter to these messages.

To review the structure of XML required to add or modify an entry, do the following:

1. In the ICEcore UI, create an entry of the type you want your Web services client to be able to create or modify, and specify a value for every element on the creation form.
2. Hover your mouse over the title of the created entry and note the entry identifier in the displayed URL:



The entry identifier in the previous picture is 83.

3. Either use the `printFolderEntry` command for `wsclient.bat` or use the `getFolderEntryAsXML` message to view the XML contents of the entry you just created in the UI.
4. Use the XML information you retrieved as a template for the string you need to pass to create a new entry or to modify one.

1.5.4 Attaching Files

In ICEcore, attachments are files that are associated with an entry. An entry may have more than one attached file.

Using Web services, an attachment is a file exchanged in conjunction with a message being passed between the client and server. ICEcore recognizes only the first file attachment to a message being sent to the server and ignores all other attachments.

To attach more than one file to an ICEcore entry, you must pass the `uploadFolderFile` message multiple times. So, to attach 17 files to an ICEcore entry, you must call `uploadFolderFile` 17 times. Your client source code establishes where it finds or places files used as attachments to messages.

The `uploadFolderFile` message requires that you pass a data item name. This identifier maps to the value specified in the `name` attribute of the `input` HTML tag used to upload the file; this value is also used in a `hidden` HTML tag that communicates values between the HTML form and the ICEcore database.

To upload a file into the standard form element used to contain attachments, specify `ss_attachFile` as the data item name. If you are uploading files into a custom form element, follow [the instructions in the previous section](#) for viewing the XML of a folder entry, obtain the name of the custom form element containing attached files, and specify it when adding more attachments.

1.5.5 Fetching Attachments

When you use `getFolderEntryAsXML` to obtain information about an entry, you use a boolean parameter to indicate if you want the entry's attachments. If you specify that you do want the attachments, your client establishes where on its system it wishes to place the attached files.

1.5.6 Adding Calendar Entries

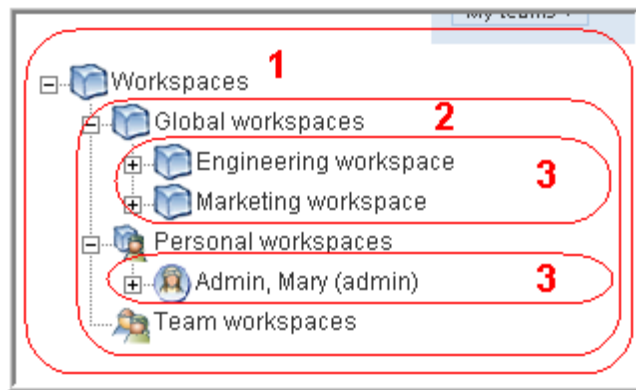
When you pass the `uploadCalendarEntries` message to the server, the Web services framework uses both an XML formatted string of iCal data passed as the second parameter to the message ("`<doc><entry>iCal data</entry></doc>`"), and an iCal file that may or may not be passed as an attachment to the message. Although you can split the source data across both the parameter and the attachment, clients generally use one or the other to specify entry data. So, if you want the iCal file attachment to be the source of data for calendar entries, then pass an empty XML document as the second parameter to `uploadCalendarEntries` ("`<doc></doc>`").

NOTE: The `uploadCalendar` command in the `wsclient.bat` batch file accepts two required parameters and an optional third parameter. The second parameter is a file containing XML that specifies iCal data. The third, optional parameter is an iCal formatted file. Both files must be located in the same directory as `wsclient.bat`. Again, if you want the iCal file to be the only source of data for newly created entries, place an empty XML document in the file specified as the second command parameter.

1.5.7 Binder Pages and `getWorkspaceTreeAsXML`

When you use `getWorkspaceTreeAsXML` to obtain information about the hierarchical workspace tree, ICEcore returns XML formatted information about nodes in the tree, within the levels of the hierarchy you specify. Each node in the tree is a binder, which is typically a place (a workspace or folder). Sometimes, the XML element returned for a node is called a page.

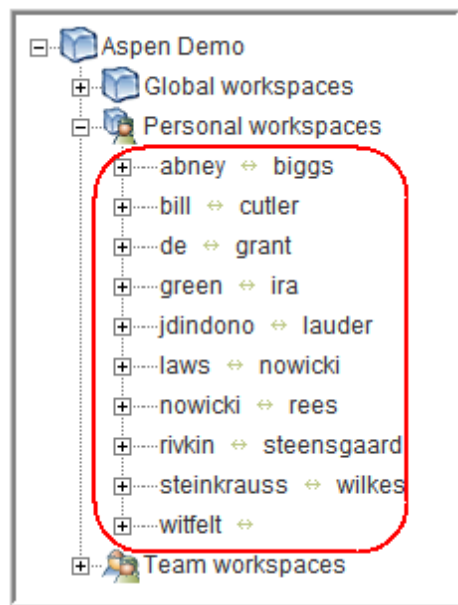
The following page includes a picture of the workspace tree, which is expanded to show three levels of the workspace hierarchy.



Each of the depicted workspaces and folders are nodes in the workspace tree. The “Workspaces” workspace is the only binder at level 1. Level 2 binders include “Global workspaces,” “Personal workspaces,” and “Team workspaces.” The remaining depicted workspaces are at the third level of the hierarchy. Because the binders at level 3 have the plus-sign icon next to their titles, there is at least a fourth level to this workspace’s hierarchy, which is not being shown in the picture.

If you use `getWorkspaceTreeAsXML` to fetch one level of the tree starting at the “Workspaces” node, ICEcore returns information about “Global workspaces,” “Personal workspaces,” and “Team workspaces.”

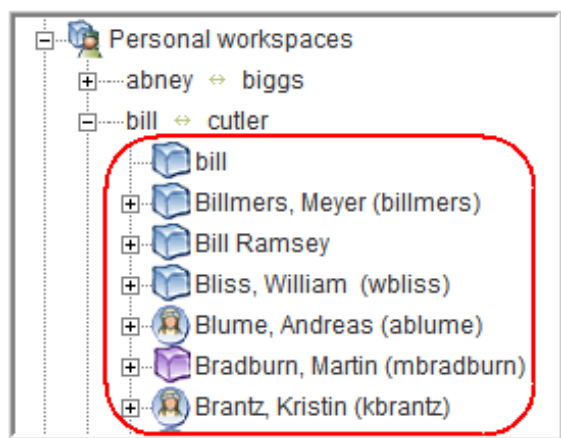
As mentioned, some nodes in the tree are pages, which are shown in this picture:



The `/ssf/web/docroot/WEB-INF/classes/config/ssf.properties` file contains a property called `wsTree.maxBucketSize`, which, by default, is set to 25. This means that the maximum number of subworkspaces or subfolders allowed is 25. If a folder or workspace has more subplaces, ICEcore creates virtual “buckets” called pages. Each line in the previous picture (“abney <--> biggs”, “bill <--> cutler”, and so on) corresponds to a page. The “Personal workspaces” workspace has ten pages.

When you use `getWorkspaceTreeAsXML` to retrieve information about nodes in the workspace tree, it can return more than one hierarchical level as you specified, unless it encounters a page. To expand the tree beyond a page, you must call `getWorkspaceTreeAsXML` again, pass the binder identifier of the page, and pass the number of levels beyond the page you wish to retrieve.

Consider the following picture:



The “bill//cutler” page contains workspaces. The workspaces listed (“bill”, “Billmers, Meyer”, “Bill Ramsey”, and so on) are one level beyond the page.

When you receive page information as a node in the workspace tree, you receive page and tuple attributes. For example, `page="1"` and `tuple="jane_adams//rhonda_hernandez"`. To obtain information about the contents of this page, you need to specify the identifier of the page’s parent, the number of hierarchy levels you want expanded, and a concatenation of the page number and tuple values, as shown in this example:

```
getWorkspaceTreeAsXML 24 3 "1//jane_adams//rhonda_hernandez"
```

This code begins at binder number 24, accesses page number 1, and returns 3 hierarchical levels of data for all users between Jane Adams and Rhonda Hernandez.

Given the structure of the ICEcore pages and how Web services returns tree data, it is easiest to retrieve page data in this way. However, if you choose, you can actually retrieve paged tree data regardless of page number. To do this, specify any page number (ICEcore actually ignores it), and specify a tuple in the correct order in which it appears in the tree, even if the set of users crosses pages. ICEcore returns hierarchical information for all users in between the tuple values. However, if the number of returned nodes exceeds the value specified in the `wsTree.maxBucketSize` property (by default, 25 users), ICEcore pages the data.

Finally, if you want to see all tree information without any page specifications, specify `-1` as the value of the hierarchy levels you want returned.

1.6 Extending ICEcore Web Services

Given that ICEcore is Open Source software, you have the source code that implements our Web services, and you can extend it. However, we invite you to operate within the spirit of an Open Source community by participating in the ICEcore online community, sharing your code with others, and working with the SiteScape Engineers to incorporate your Web services extensions into

the base product. In this way, you make the product and community stronger, and you avoid doing work that may need to be reworked in future versions of ICEcore due to engineering changes.

Of course, whether you participate in the community or upgrade to future versions of the software is up to you. Regardless of your decision, ICEcore includes an example that provides a structure that enables users of either the open or enterprise version of our software to be able to extend our Web services in the most optimal way, minimizing work that you would have to do to maintain the extensions for every upgrade.

ICEcore includes an extended Web services example, which adds the `getBinderTitle` message and the `getBinderTitle` command to the `wsclient.bat` sample client. The source code for the extension is located in this directory and in its subdirectories:

```
/ssf/samples/extendedws
```

This directory contains the `readme.txt` file, which provides cursory directions for establishing the extension. Those instructions are repeated here and elaborated upon.

To implement the extension:

1. Use a command-line window to go to the `/extendedws` directory.
2. Execute this command:

```
> ant deploy
```

3. In the installation directories (not the source code), open this file for editing:

```
/icecore installation dir/webapps/ssf/WEB-INF/server-config.wsdd
```

Replace the string `com.sitescape.team.remoting.ws.JaxRpcFacade` with the string `com.sitescape.team.samples.extendedws.server.JaxRpcFacade2`.

NOTE: If you upgrade to a new version of the software, you need to repeat this step to re-implement your Web services extensions.

4. In the same set of directories, open this file for editing:

```
/install dir/webapps/ssf/WEB-INF/context/applicationContext.xml
```

Replace the string `com.sitescape.team.remoting.ws.FacadeImpl` with the string `com.sitescape.team.samples.extendedws.server.FacadeImpl2`.

Also, replace the string `com.sitescape.team.remoting.Facade` with the string `com.sitescape.team.samples.extendedws.server.Facade2`.

NOTE: If you upgrade to a new version of the software, you need to repeat this step to re-implement your Web services extensions.

5. Restart the ICEcore server.
6. To test the new Web services message, view this page in a browser window:

```
http://local host and port/ssf/ws/Facade?wsdl
```

The `getBinderTitle` message should now appear.

To test the additional command in the `wsclient.bat` file (Windows only), `cd` to this directory:

```
/ssf/samples/remotingclient
```

Execute this command:

```
> ant zip
```

And test the newly enabled batch-file command:

```
> wsclient getBinderTitle 1
```

Both the new message and command accept a binder identifier as a parameter and return the textual title of the binder.

Message Reference

2

This chapter provides alphabetized reference pages for messages provided by ICEcore Web services. The following are conventions used in this reference section:

What you see	What it means
Click the <i>Add a team workspace</i> button.	Items that are clickable on the page, programming variables, or syntax parameters are presented in italic font.
Click the <i>Getting Started</i> link.	
Blog summary - Provides a....	Defined terms in a list, note headers, section headers on a reference page, and list items on a reference page are presented in bold font.
Note: Remember that....	
Type <code>status</code> , then press Enter.	Text that you must type, file names, commands, command options, routines, Web services messages, and parameters are presented in Courier font when occurring in a body of text.
Open the <code>ManagerGuide.pdf</code> file.	
Use the <code>open_db</code> routine with its <code>lock</code> parameter.	
[page]	Optional syntax parameters are enclosed in brackets ([]).
..., paramSyntax1 paramSyntax2,...	Required parameters that accept two or more optional syntaxes are separated by the vertical-line character.
(V1+)	The version of ICEcore that supports the Web services message ("Version 1.0 or later")

NOTE: All examples in this reference section use Apache Axis run-time library methods that specify Web service messages and their argument lists. If you are not using Apache Axis, map the Apache methods to those you are using to implement your Web service calls.

For ICEcore Version 1.0, regarding messages shown in the product source code, the `search` message is under development and subject to change or deletion at any time. Do not use this message in your client applications.

Web service messages contained in this reference section are used by the Windows based clients provided in the ICEcore sources in the `/ssf/samples/remotingclient` folder. With the exception of `uploadCalendar` (see [the reference page for uploadCalendarEntries](#) for more information), use the same parameters for the batch-file command that you use for the corresponding Web service message.

The following table maps the `wsclient.bat` command name to its corresponding Web services message, which is documented in this reference section:

wsclient.bat command	Web services message
<code>addEntry</code>	<code>addFolderEntry</code>
<code>addFolder</code>	<code>addFolder</code>

wsclient.bat command	Web services message
modifyEntry	modifyFolderEntry
printAllPrincipals	getAllPrincipalsAsXML
printDefinition	getDefinitionAsXML
printDefinitionConfig	getDefinitionConfigAsXML
printFolderEntry	getFolderEntryAsXML
printFolderEntries	getFolderEntriesAsXML
printPrincipal	getPrincipalAsXML
printTeamMembers	getTeamMembersAsXML
printTeams	getTeamsAsXML
printWorkspaceTree	getWorkspaceTreeAsXML
uploadCalendar	uploadCalendarEntries
uploadFile	uploadFolderFile

addFolder

Adds a folder to the workspace-tree hierarchy. (V1+)

Syntax

```
public long addFolder( long parentBinderId, long binderConfigId, String title );
```

Description

The `addFolder` message adds a folder to the workspace and folder hierarchy.

Parameters and Return Value

parentBinderId

The identifier of the workspace or folder that is to contain the new folder.

binderConfigId

The identifier that maps to the default configuration for the folder you want to create.

title

A string providing a title for the new entry.

return_value

The binder identifier of the newly created folder.

Example

```
call.setOperationName(new QName("addFolder"));
Object result = call.invoke(new Object[] {new Long(21), new
Long(146), new String("My new folder")});
```

This code creates a new subfolder to the container whose binder identifier is 21, gives the folder a configuration identifier of 146 (on our test installation, this corresponds to a discussion folder), and establishes the title of the new folder as "My new folder". The container whose binder identifier is 21 may be either a workspace or folder.

See Also

- ♦ [The message table for the Windows based `wsclient.bat` program](#)
- ♦ [Section 1.5.1, "Adding Folders and the Binder Configuration Identifier," on page 13](#)

addFolderEntry

Adds an entry to a folder. (V1+)

Syntax

```
public long addFolderEntry( long folderId, String definitionId, String inputDataAsXML,  
String attachedFileName | null );
```

Description

The `addFolderEntry` message adds an entry to a folder.

Parameters and Return Value

folderId

The binder identifier of the folder that is to contain the new entry.

definitionId

The 32-character, hexadecimal identifier that maps to the type of entry you wish to create (for example, some default entry types are topic, file, blog, wiki, and calendar).

The easiest way to work with definition identifiers is to specify `null` for this value. When you specify `null`, ICEcore automatically applies the definition identifier for the default entry type of the folder in which you are creating a new entry. For example, by default, say you want to create an entry in a blog folder. If you pass `null` as the definition identifier, ICEcore automatically applies the definition identifier for a blog entry.

As another option, you can use the `getDefinitionConfigAsXML` message to get information about all definitions. Then, you can parse the XML string for the definition identifier of the type of entry you want.

inputDataAsXML

A string of XML containing the values needed to create an entry of your desired type.

attachedFileName

The name of the file you wish to attach to the new entry. This is an optional parameter. The file must be located in the directory in which the client code executes.

return_value

The entry identifier for the newly created entry.

Examples

```
call.setOperationName(new QName("addFolderEntry"));  
Object result = call.invoke(new Object[] {new Long(21), new  
String("402883b90cc53079010cc539bf260002"), s, filename},  
filename);
```

This code creates a new entry in the folder whose binder identifier is 21; the specified entry-definition identifier maps to a discussion topic. The variable `s` contains XML elements needed

by ICEcore to create the entry. The new entry includes the attached file whose filename is specified by the value of the `filename` variable.

```
call.setOperationName(new QName("addFolderEntry"));
Object result = call.invoke(new Object[] {new Long(21), new
String("402883b90cc53079010cc539bf260002"), s, null});
```

This code produces the same effect as the last example, except that it does not attach a file.

See Also

- ♦ [The message table for the Windows based `wsclient.bat` program](#)
- ♦ [Section 1.5.2, “Adding Entries and the Definition Identifier,” on page 15](#)
- ♦ [Section 1.5.3, “Tips for All Messages that Add and Modify,” on page 15](#)
- ♦ [getDefinitionConfigAsXML \(page 30\)](#)

getAllPrincipalsAsXML

Returns summary information for users and groups. (V1+)

Syntax

```
public String getAllPrincipalsAsXML( int firstRecord, int maxRecords );
```

Description

The `getAllPrincipalsAsXML` message returns XML elements that provide summary information about registered users and defined groups. You can use this message to identify a particular user by name or other data, obtain an identifier for a particular user, and then use the `getPrincipalAsXML` message to gather a finer level of information about that person.

Parameters and Return Value

firstRecord

The index of the first record whose user or group information you want to obtain. The index for the first principal in the system is 1.

maxRecords

The maximum number of user and group records whose information should be returned.

You can use the previous parameter and this parameter in subsequent calls to `getAllPrincipalsAsXML` to process data for sets of users and groups at a time (say, 50 at a time, or 100 at a time).

return_value

A string containing the XML elements providing information about the requested set of users and groups.

Example

```
call.setOperationName(new QName("getAllPrincipalsAsXML")) ;  
Object result = call.invoke(new Object[] {new Integer(100), new  
Integer(50) }) ;
```

This code requests information for users and groups starting with the record number 100 and including up to 50 records.

See Also

- ♦ [The message table for the Windows based `wsclient.bat` program](#)
- ♦ [getPrincipalAsXML \(page 33\)](#)

getDefinitionAsXML

Returns information about one definition. (V1+)

Syntax

```
public String getDefinitionAsXML( String definitionID );
```

Description

The `getDefinitionAsXML` message returns an XML string containing information about one definition. You work with definitions using the designers in the administration UI.

For example, if you pass one of the definition identifiers for an entry type listed in the `addFolderEntry` reference page, ICEcore returns information about the definition for that entry.

As an alternative, you can use the `getDefinitionConfigAsXML` message to obtain all definitions in ICEcore and then parse the larger string for the definition information you want.

Parameter and Return Value

definitionID

The identifier of the definition whose information you want. Definitions are maintained using the designers in the administration UI, and define the components of an object in ICEcore.

return_value

A string of XML whose elements provide information about the components of an object in ICEcore.

Example

```
call.setOperationName(new QName("getDefinitionAsXML"));  
Object result = call.invoke(new Object[] {new  
String("402883b9114739b301114754e8120008")});
```

This code requests XML-formatted information about the definition for a wiki entry.

See Also

- ♦ [The message table for the Windows based `wsclient.bat` program](#)
- ♦ [Section 1.5.2, “Adding Entries and the Definition Identifier,” on page 15](#)
- ♦ [addFolderEntry \(page 26\)](#)
- ♦ [getDefinitionConfigAsXML \(page 30\)](#)

getDefinitionConfigAsXML

Returns information about all configuration definitions. (V1+)

Syntax

```
public String getDefinitionConfigAsXML();
```

Description

The `getDefinitionConfigAsXML` message returns information about all configuration definitions. The configuration information does not include workflow or template definitions. You can use the returned information to extract the definition identifier for a given entry type to use in a subsequent call to `addFolderEntry`.

Return Value

return_value

A string of XML whose elements describe all configuration definitions.

Example

```
call.setOperationName(new QName("getDefinitionConfigAsXML"));  
Object result = call.invoke();
```

This code obtains information about all configuration settings.

See Also

- ♦ [The message table for the Windows based `wsclient.bat` program](#)
- ♦ [addFolderEntry \(page 26\)](#)

getFolderEntriesAsXML

Returns a string containing XML that provides summary information about all of the entries in a folder. (V1+)

Syntax

```
public String getFolderEntriesAsXML( long folderId );
```

Description

The `getFolderEntriesAsXML` message returns XML elements containing summary information about each entry in the specified folder.

Parameter and Return Value

folderId

The binder identifier of the folder containing the entries for which you want information.

return_value

A string containing XML elements containing summary information for each entry in the folder specified by `folderId`.

Example

```
call.setOperationName(new QName("getFolderEntriesAsXML")) ;  
Object result = call.invoke(new Object[] {new Long(21)});
```

This code returns a string containing XML information for all of the entries in the folder whose binder identifier is 21.

See Also

- ♦ [The message table for the Windows based `wsclient.bat` program](#)

getFolderEntryAsXML

Returns information about one entry in a folder. (V1+)

Syntax

```
public String getFolderEntryAsXML( long folderId, long entryId, boolean includeAttachments );
```

Description

The `getFolderEntryAsXML` message returns XML whose elements provide information about one entry in a folder.

Parameters and Return Value

folderId

The binder identifier of the folder containing the entry whose information you want.

entryId

The identifier of the entry whose information you want.

includeAttachments

A boolean value that indicates whether you want ICEcore to return the entry's attachments. The client program is responsible for placement of attachment files on its local system.

return_value

A string containing XML elements for the requested entry.

Example

```
call.setOperationName(new QName("getFolderEntryAsXML")) ;  
Object result = call.invoke(new Object[] {new Long(21), new  
Long(34), new Boolean.FALSE});
```

This code returns XML that includes information contained in entry number 34 in the folder whose identifier is 21. Because of the value of the last parameter, ICEcore does not place the entry's file attachments in the client program's source directory.

See Also

- ♦ [The message table for the Windows based `wsclient.bat` program](#)
- ♦ [Section 1.5.5, "Fetching Attachments," on page 17](#)

getPrincipalAsXML

Returns information about one user or group. (V1+)

Syntax

```
public String getPrincipalAsXML( long binderId, long principalId );
```

Description

The `getPrincipalAsXML` message returns XML whose elements provide information about one registered user or defined group.

Parameters and Return Value

binderId

The binder identifier of the principal's parent workspace. The information returned by `getAllPrincipalsAsXML` includes the binder number of this containing workspace.

principalId

The identifier that maps to the user or group for which you want to gather information.

return_value

A string containing XML elements whose elements provide information about the specified user or group.

Example

```
call.setOperationName(new QName("getPrincipalAsXML"));
Object result = call.invoke(new Object[] {new Long(2), new
Long(25)});
```

This code returns information about a user or group, whose parent workspace has a binder identifier of 2 and whose principal identifier is 25.

See Also

- ♦ [The message table for the Windows based `wsclient.bat` program](#)
- ♦ [getAllPrincipalsAsXML \(page 28\)](#)

getTeamMembersAsXML

Returns information about all team members assigned within a workspace or folder. (V1+)

Syntax

```
public String getTeamMembersAsXML( long binderId );
```

Description

The `getTeamMembersAsXML` message returns XML that names members of a team assigned within the specified workspace or folder.

Parameter and Return Value

binderId

The binder identifier of the workspace or folder for which you want information about team members. The `getTeamsAsXML` message returns information about all workspaces and folders that have assigned teams.

return_value

A string containing XML elements describing team members for the specified place.

Example

```
call.setOperationName(new QName("getTeamMembersAsXML"));  
Object result = call.invoke(new Object[] {new Long(23)});
```

This code returns an XML string whose elements describe all of the team members assigned in the workspace or folder associated with the binder identifier of 23.

See Also

- ♦ [The message table for the Windows based `wsclient.bat` program](#)
- ♦ [getTeamsAsXML \(page 35\)](#)

getTeamsAsXML

Returns information about all workspaces and folders that have assigned teams. (V1+)

Syntax

```
public String getTeamsAsXML();
```

Description

The `getTeamsAsXML` message returns an XML string providing information about all workspaces and folders that have assigned teams. You can use this message to obtain the list of places that have assigned teams, note a binder number of a particular place, and then use the `getTeamMembersAsXML` message to obtain the list of team members for that place.

Return Value

return_value

An XML string whose elements describe workspaces and folders that have assigned teams.

Example

```
call.setOperationName(new QName("getTeamsAsXML"));  
Object result = call.invoke();
```

This code returns information about all places in the ICEcore installation that have assigned teams.

See Also

- ♦ [The message table for the Windows based wsclient.bat program](#)
- ♦ [getTeamMembersAsXML \(page 34\)](#)

getWorkspaceTreeAsXML

Returns information needed to construct the ICEcore workspace and folder tree. (V1+)

Syntax

```
public String getWorkspaceTreeAsXML( long binderId, int levels, String page );
```

Description

The `getWorkspaceTreeAsXML` message returns XML elements needed to construct the requested portion of the ICEcore workspace tree.

Parameters and Return Value

binderId

The binder identifier of the starting node of the returned portion of the hierarchy. The top workspace in the ICEcore tree has a binder ID of 1.

levels

The number of hierarchical levels down from the node specified by `binderId` that you want to include in the returned information. The value `-1` indicates “all subsequent levels.”

page

An parameter used to expand pages of binders. When you specify a valid page identifier, ICEcore expands the page by the levels indicated in the `levels` parameter. (The Web-services overview chapter contains [more detailed information about working with pages.](#))

If you do not want to expand pages using this call, pass `null` as this parameter.

return_value

A string containing XML elements needed to construct each node within the requested levels of the workspace hierarchy.

Example

```
call.setOperationName(new QName("getWorkspaceTreeAsXML"));  
Object result = call.invoke(new Object[] {new Long(1), new  
Integer(3), null});
```

This code returns a string containing XML information for the first three levels of the workspace hierarchy. The following depicts these levels using default workspace titles:

Level 1: Workspaces

Level 2: Global, Personal, and Team workspaces

Level 3: Children of Global, Personal, and Team

The children of “Global workspaces,” “Personal workspaces,” and “Team workspaces” can be either workspaces or folders.

See Also

- ♦ [The message table for the Windows based wsclient.bat program](#)
- ♦ [Section 1.5.7, “Binder Pages and getWorkspaceTreeAsXML,” on page 17](#)

modifyFolderEntry

Modifies a single entry. (V1+)

Syntax

```
public void modifyFolderEntry( long folderId, long entryId, String inputDataAsXML );
```

Description

The `modifyFolderEntry` message modifies one entry in a folder.

Parameters and Return Value

folderId

The binder identifier of the folder that contains the entry to be modified.

entryId

The identifier of the entry to be modified.

inputDataAsXML

A string of XML containing the values needed to modify the entry.

return_value

None.

Example

```
call.setOperationName(new QName("modifyFolderEntry"));  
Object result = call.invoke(new Object[] {new Long(21), new  
Long(43), s});
```

This code modifies entry 43 in the folder whose binder ID is 21. The variable `s` contains XML elements needed by ICEcore to modify the contents of the entry.

See Also

- ♦ [The message table for the Windows based `wsclient.bat` program](#)
- ♦ [Section 1.5.3, “Tips for All Messages that Add and Modify,” on page 15](#)

uploadCalendarEntries

Creates new calendar entries from a file. (V1+)

Syntax

```
public void uploadCalendarEntries( long folderId, String XMLCalendarData );
```

Description

The `uploadCalendarEntries` message uses iCal information in an XML string or in an attachment to add entries to a calendar folder.

NOTE: The `uploadCalendar` command in the `wsclient.bat` batch file accepts two required parameters and an optional third parameter. The second parameter is a file containing XML that specifies iCal data. The third, optional parameter is an iCal formatted file. Both files must be located in the same directory as `wsclient.bat`. Again, if you want the iCal file to be the only source of data for newly created entries, place an empty XML document in the file specified as the second command parameter.

Parameters and Return Value

folderId

The binder identifier of the calendar folder that is to contain the new entries.

XMLCalendarData

A string containing XML formatted calendar data (`<doc><entry>iCal data</entry>...</doc>`). If you wish to specify all of your calendar data in an iCal file attached to the message, pass an empty document for this string (`<doc></doc>`).

return_value

None.

Example

```
call.setOperationName(new QName("uploadCalendarEntries"));
Object result = call.invoke(new Object[] {new Long(21), s});
```

This code creates new entries in the calendar folder whose binder ID is 21. ICEcore uses XML-formatted iCal information contained in the `s` variable to create the new calendar entries.

See Also

- ♦ [The message table for the Windows based `wsclient.bat` program](#)
- ♦ [Section 1.5.6, “Adding Calendar Entries,” on page 17](#)

uploadFolderFile

Attaches a file to an entry to a folder. (V1+)

Syntax

```
public void uploadFolderFile( long folderId, String entryId, String fileUploadDataItemName,  
String attachedFileName );
```

Description

The `uploadFolderFile` message attaches a file to an entry in a folder. You can attach only one file at a time; call this message multiple times to attach more than one file to the entry. Files to be attached must be located in the same directory as the executing client.

Parameters and Return Value

folderId

The binder identifier of the folder that contains the entry to which you want to attach a file.

entryId

The identifier of the entry to which you want to attach a file.

fileUploadDataItemName

A string containing the internal identifier for the part of the entry that contains attached files. This identifier maps the `name` attribute of an `input` HTML tag on a form to the ICEcore database; a `hidden` HTML tag communicates this mapping to the server.

The name value for the standard entry element containing attached files is `ss_attachFile`. If you want to upload a file into a custom form element you defined using the designers, you need to look up the name identifier for that form element (see also `getDefinitionConfigAsXML` or `getFolderEntryAsXML`).

attachedFileName

The name of the file you wish to attach to the new entry. This client is responsible for locating on its local system the file to be used as an attachment.

return_value

None.

Example

```
call.setOperationName(new QName("uploadFolderFile"));
Object result = call.invoke(new Object[] {new Long(21), new
Long(43), new String("ss_attachFile"), filename}, filename);
```

This code attaches a file to entry 43 in the folder whose binder ID is 21. The name of the file to be attached to the entry is contained in the variable `filename`.

See Also

- ♦ [The message table for the Windows based `wsclient.bat` program](#)
- ♦ [Section 1.5.4, “Attaching Files,” on page 16](#)
- ♦ [getDefinitionConfigAsXML \(page 30\)](#)
- ♦ [getFolderEntryAsXML \(page 32\)](#)