

## Developer Guide

# Kablink Teaming

**2.0**

November 25, 2009

## Legal Notices

Novell, 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, Novell, 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, Novell, 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, Novell, Inc. reserves the right to make changes to any and all parts of Novell 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. See the [Novell International Trade Services Web page \(http://www.novell.com/info/exports/\)](http://www.novell.com/info/exports/) for more information on exporting Novell software. Novell assumes no responsibility for your failure to obtain any necessary export approvals.

Copyright © 2009 Novell, Inc. All rights reserved. Permission is granted to copy, distribute, and/or modify this document under the terms of the GNU Free Documentation License (GFDL), Version 1.2 or any later version, published by the Free Software Foundation with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the GFDL can be found at <http://www.fsf.org/licenses/fdl.html>.

THIS DOCUMENT AND MODIFIED VERSIONS OF THIS DOCUMENT ARE PROVIDED UNDER THE TERMS OF THE GNU FREE DOCUMENTATION LICENSE WITH THE FURTHER UNDERSTANDING THAT:

1. THE DOCUMENT IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, WARRANTIES THAT THE DOCUMENT OR MODIFIED VERSION OF THE DOCUMENT IS FREE OF DEFECTS, MERCHANTABLE, FIT FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THE ENTIRE RISK AS TO THE QUALITY, ACCURACY, AND PERFORMANCE OF THE DOCUMENT OR MODIFIED VERSION OF THE DOCUMENT IS WITH YOU. SHOULD ANY DOCUMENT OR MODIFIED VERSION PROVE DEFECTIVE IN ANY RESPECT, YOU (NOT THE INITIAL WRITER, AUTHOR OR ANY CONTRIBUTOR) ASSUME THE COST OF ANY NECESSARY SERVICING, REPAIR OR CORRECTION. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS LICENSE. NO USE OF ANY DOCUMENT OR MODIFIED VERSION OF THE DOCUMENT IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER; AND

2. UNDER NO CIRCUMSTANCES AND UNDER NO LEGAL THEORY, WHETHER IN TORT (INCLUDING NEGLIGENCE), CONTRACT, OR OTHERWISE, SHALL THE AUTHOR, INITIAL WRITER, ANY CONTRIBUTOR, OR ANY DISTRIBUTOR OF THE DOCUMENT OR MODIFIED VERSION OF THE DOCUMENT, OR ANY SUPPLIER OF ANY OF SUCH PARTIES, BE LIABLE TO ANY PERSON FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF GOODWILL, WORK STOPPAGE, COMPUTER FAILURE OR MALFUNCTION, OR ANY AND ALL OTHER DAMAGES OR LOSSES ARISING OUT OF OR RELATING TO USE OF THE DOCUMENT AND MODIFIED VERSIONS OF THE DOCUMENT, EVEN IF SUCH PARTY SHALL HAVE BEEN INFORMED OF THE POSSIBILITY OF SUCH DAMAGES.

Novell, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more of the U.S. patents listed on the [Novell Legal Patents Web page \(http://www.novell.com/company/legal/patents/\)](http://www.novell.com/company/legal/patents/) and one or more additional patents or pending patent applications in the U.S. and in other countries.

Novell, Inc.

404 Wyman Street, Suite 500  
Waltham, MA 02451  
U.S.A.  
[www.novell.com](http://www.novell.com)

*Online Documentation:* To access the latest online documentation for this and other Novell products, see  
[the Novell Documentation Web page \(http://www.novell.com/documentation\)](http://www.novell.com/documentation).

## **Novell Trademarks**

For Novell trademarks, see [the Novell Trademark and Service Mark list \(http://www.novell.com/company/legal/trademarks/tmlist.html\)](http://www.novell.com/company/legal/trademarks/tmlist.html).

## **Third-Party Materials**

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

# Contents

<b>About This Manual</b>	<b>9</b>
<b>1 Web Services Overview</b>	<b>11</b>
1.1 Teaming Web Services Terminology	11
1.2 Web Services Implementation	12
1.2.1 Sample Clients	12
1.3 Authentication	14
1.3.1 HTTP Basic Authentication Access (ssr)	14
1.3.2 Web Services Security Access (ssf)	14
1.4 Server Endpoints	15
1.5 Categories of Operations	15
1.6 Client Stubs	16
1.7 Managing Data	16
1.7.1 Working with Java Objects	17
1.7.2 Adding Folders and the Binder Configuration Identifier	18
1.7.3 Attaching Files	19
1.7.4 Fetching Attachments	20
1.7.5 Adding Calendar Entries	20
1.7.6 Binder Pages and search_getWorkspaceTreeAsXML	20
1.8 Extending Teaming Web Services	22
<b>A Web Service Operations</b>	<b>23</b>
admin_destroyApplicationScopedToken	26
admin_getApplicationScopedToken	27
binder_addBinder	28
binder_copyBinder	29
binder_deleteBinder	30
binder_deleteTag	31
binder_getBinder	32
binder_getBinderByPathName	33
binder_getFileVersions	34
binder_getFolders	35
binder_getSubscription	36
binder_getTags	37
binder_getTeamMembers	38
binder_indexBinder	39
binder_indexTree	40
binder_modifyBinder	41
binder_moveBinder	42
binder_removeFile	43
binder_setDefinitions	44
binder_setFunctionMembership	45
binder_setFunctionMembershipInherited	46
binder_setOwner	47
binder_setSubscription	48
binder_setTag	49

binder_setTeamMembers	50
binder_uploadFile	51
definition_getDefinitionAsXML	52
definition_getDefinitionByName	53
definition_getDefinitions	54
definition_getLocalDefinitionByName	55
definition_getLocalDefinitions	56
folder_addEntry	57
folder_addEntryWorkflow	58
folder_addMicroBlog	59
folder_addReply	60
folder_copyEntry	61
folder_deleteEntry	62
folder_deleteEntryTag	63
folder_deleteEntryWorkflow	64
folder_getEntries	65
folder_getEntry	66
folder_getEntryByFileName	67
folder_getEntryTags	68
folder_getFileVersions	69
folder_getSubscription	70
folder_modifyEntry	71
folder_modifyWorkflowState	72
folder_moveEntry	73
folder_removeFile	74
folder_reserveEntry	75
folder_setEntryTag	76
folder_setRating	77
folder_setSubscription	78
folder_setWorkflowResponse	79
folder_synchronizeMirroredFolder	80
folder_unreserveEntry	81
folder_uploadFile	82
folder_uploadFileStaged	83
ical_uploadCalendarEntriesWithXML	85
ldap_synchAll	86
ldap_synchUser	87
license_getExternalUsers	88
license_getRegisteredUsers	89
license_updateLicense	90
migration_addBinder	91
migration_addBinderWithXML	92
migration_addEntryWorkflow	94
migration_addFolderEntry	95
migration_addFolderEntryWithXML	96
migration_addReply	98
migration_addReplyWithXML	99
migration_uploadFolderFile	101
migration_uploadFolderFileStaged	103
profile_addGroup	105
profile_addGroupMember	106
profile_addUser	107

profile_addUserWorkspace	108
profile_deletePrincipal	109
profile_getFileVersions	110
profile_getGroup	111
profile_getGroupByName	112
profile_getGroupMembers	113
profile_getPrincipals	114
profile_getUser	115
profile_getUserByName	116
profile_getUsers	117
profile_getUserTeams	118
profile_modifyGroup	119
profile_modifyUser	120
profile_removeFile	121
profile_removeGroupMember	122
profile_uploadFile	123
search_getFolderEntries	124
search_getHotContent	125
search_getTeams	126
search_getWorkspaceTreeAsXML	127
search_search	128
template_addBinder	130
template_getTemplates	131
zone_addZone	132
zone_deleteZone	133
zone_modifyZone	134

## **B Deprecated Web Service Operations 135**

addFolder	137
addFolderEntry	138
addReply	140
addUserWorkspace	142
getAllPrincipalsAsXML	143
getDefinitionAsXML	144
getDefinitionConfigAsXML	145
getDefinitionListAsXML	146
getFolderEntriesAsXML	147
getFolderEntryAsXML	148
getPrincipalAsXML	149
getTeamMembersAsXML	150
getTeamsAsXML	151
getWorkspaceTreeAsXML	152
indexFolder	154
migrateBinder	155
migrateEntryWorkflow	157
migrateFolderEntry	159
migrateFolderFile	161
migrateFolderFileStaged	163
migrateReply	165
modifyFolderEntry	167
setDefinitions	168

setFunctionMembership . . . . .	169
setFunctionMembershipInherited . . . . .	171
setOwner . . . . .	172
setTeamMembers . . . . .	173
synchronizeMirroredFolder . . . . .	174
uploadCalendarEntries . . . . .	175
uploadFolderFile . . . . .	176

## **C Migrating from Forum to Kablink Teaming 179**

C.1 Sequence of Migration Operations . . . . .	179
C.2 Migration Overwrite Operations . . . . .	180
C.3 Migrating Users . . . . .	180
C.4 Migrating Files . . . . .	180
C.5 Migrating Custom Commands and Workflow . . . . .	181



# About This Manual

The *Kablink Teaming 2.0 Developer Guide* present ways to extend the functionality of Teaming 2.0. The guide is divided into the following sections:

- ♦ [Chapter 1, “Web Services Overview,” on page 11](#)
- ♦ [Appendix A, “Web Service Operations,” on page 23](#)
- ♦ [Appendix B, “Deprecated Web Service Operations,” on page 135](#)
- ♦ [Appendix C, “Migrating from Forum to Kablink Teaming,” on page 179](#)

## Audience

This guide is intended for programmers who want to write extensions for Teaming 2.0.

## Feedback

We want to hear your comments and suggestions about this manual and the other documentation included with this product. Please use the User Comments feature at the bottom of each page of the online documentation, or go to [www.novell.com/documentation/feedback.html](http://www.novell.com/documentation/feedback.html) and enter your comments there.

## Documentation Updates

For the most recent version of the *Kablink Teaming 2.0 Developer Guide* and other documentation, visit the [Kablink Teaming 2.0 Documentation Web site](http://www.novell.com/documentation/kablinkteaming2) (<http://www.novell.com/documentation/kablinkteaming2>).

## Additional Documentation

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

- ♦ Kablink Teaming Help system
- ♦ *Kablink Teaming Quick Start*
- ♦ *Kablink Teaming User Guide*
- ♦ *Kablink Teaming Advanced User Guide*
- ♦ *Kablink Teaming Installation Guide*
- ♦ *Kablink Teaming Administration Guide*

To access the Kablink Teaming Help system, log in to the Teaming site, then click the *Help* icon (question mark), then click a yellow Help spot for context-sensitive help.

To access the Kablink Teaming guides from within Teaming, click the *Help* icon (question mark), then click *Teaming Manuals*.

## Documentation Conventions

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

A trademark symbol (<sup>®</sup>, <sup>™</sup>, etc.) denotes a Novell 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.

# Web Services Overview

# 1

Novell® offers a set of operations that you can use in client programs to exchange information with a server that is running an installation of Kablink Teaming 2.0 or later.

In addition to the overview information in this chapter, see [Appendix A, “Web Service Operations,” on page 23](#), for reference information about the latest operations for the new interface. For reference information about earlier Web Services operations that have been superseded by the current release, see [Appendix B, “Deprecated Web Service Operations,” on page 135](#).

- ♦ [Section 1.1, “Teaming Web Services Terminology,” on page 11](#)
- ♦ [Section 1.2, “Web Services Implementation,” on page 12](#)
- ♦ [Section 1.3, “Authentication,” on page 14](#)
- ♦ [Section 1.4, “Server Endpoints,” on page 15](#)
- ♦ [Section 1.5, “Categories of Operations,” on page 15](#)
- ♦ [Section 1.6, “Client Stubs,” on page 16](#)
- ♦ [Section 1.7, “Managing Data,” on page 16](#)
- ♦ [Section 1.8, “Extending Teaming Web Services,” on page 22](#)

## 1.1 Teaming Web Services Terminology

The following are Kablink Teaming definitions that can assist you when using the Teaming 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. This number represents a set of information that Teaming uses to establish configuration settings, such as the default view, allowable views, allowable workflow, and workflow associations.
- ♦ **binder ID:** A unique number that identifies a specific workspace or folder.
- ♦ **data item name:** A tag value that maps an HTML form element to a value stored in the Teaming 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 by using the designers in the administration portlet.) You need to specify this value when creating a new entry in a folder.
- ♦ **page:** A 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 in the user interface (UI). [Section 1.7.6, “Binder Pages and search\\_getWorkspaceTreeAsXML,” on page 20](#) 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.
- ♦ **stub:** A proxy on the client. The stub code performs SOAP calls to the server.

## 1.2 Web Services Implementation

- ♦ [Section 1.2.1, “Sample Clients,” on page 12](#)

Teaming implements Java Web services, which provide a set of operations that client programs can use to exchange information with Teaming. The alphabetized reference section in this documentation provides syntax for these operations ([Appendix B, “Deprecated Web Service Operations,” on page 135](#)).

You can view a list of available operations online:

```
http://localhost:8080/ssf/ws
```

The latest operations are listed under the *TeamingServiceVI* header, and the deprecated operations are listed under the *Facade* header.

You can also access the Teaming Web Services Description Language (WSDL) file:

```
http://localhost:8080/ssf/ws/TeamingServiceV1?wsdl
```

In the previous two examples, replace the `localhost` specification with the host and port for your Teaming installation.

---

**NOTE:** Teaming does not currently publish its WSDL file with Universal Description, Discovery, and Integration (UDDI) or the Web Services Inspection Language (WSIL). Use the alphabetized reference section in this manual ([Appendix A, “Web Service Operations,” on page 23](#)) or the URL-generated WSDL file to understand the Teaming operation interface. For reference information about earlier Web Services operations that have been superseded by the current release, see [Appendix B, “Deprecated Web Service Operations,” on page 135](#).

---

When you make calls to Teaming Web services, there are two ways that you can implement lower-level Simple Object Access Protocol (SOAP) calls:

- ♦ Unzip client-side routines on the system running your application. These routines are Java classes and other files that produce a stub. Your application can use an interface with these stub routines, which make the SOAP calls to and from the server. See [Section 1.6, “Client Stubs,” on page 16](#), for more information about implementing these client-side routines on your application’s system.
- ♦ Have your application perform the SOAP calls by using, for example, routines from the Apache Axis toolkit.

Teaming Web services accepts and provides data by using Java objects and methods defined in the Teaming source code. (Visit the [Open Source Community page \(http://www.kablink.org\)](http://www.kablink.org) for more information about downloading the source code.) Although this section provides tips for locating object and method definitions, you might want to apply a tool such as Javadoc to the sources, so that you have reference pages to assist you in working with the Teaming objects and methods.

The primary method of learning to use Teaming Web services is by reviewing sample clients and their source code, which are provided in the Teaming sources.

### 1.2.1 Sample Clients

Teaming provides sample clients in its product code base that can assist you in learning how to use its Web services. The sample clients are located within the source code:

`/ssf/samples/wsclient`

The following sample clients are provided. They are listed in the order of how helpful they are in learning how to make Web service calls:

- ♦ **teaming-service-client-with-stub.bat (Teaming 2.0+)**: Uses client-side routines to implement a Windows batch file for simple operations; this is the recommended method. Using this batch file requires the installation of the client-side routines.
- ♦ **teaming-service-client-with-call.bat (Teaming 2.0+)**: Uses the Axis Call object when making Web service calls, as a way to implement a Windows batch file for simple operations.
- ♦ **facade-client.bat (V1+)**: Uses the deprecated Web services interface.
- ♦ **wsExport.bat and wsImport.bat (Teaming 2.0+)**: Takes data from a portion of the workspace and folder hierarchy and reproduces it on another file system. These tools are not a complete import and export facility, because they do not retain the workflow states, access-control settings, and history of the original objects.

You can find the source files for the sample clients here:

`/ssf/samples/wsclient/src/org/kablink/teaming/samples/wsclient`

The `TeamingServiceClientWithCall.java` file extends the `WSClientBase.java` file, which is also located in the `/ssf/samples/wsclient/src/org/kablink/teaming/samples/wsclient` directory.

### Enabling the .bat clients (Windows systems only)

Before executing the sample .bat programs on a Windows system, you need to do some work in your build to enable them.

- 1 Execute the build Ant target in `/ssf/samples/wsclient/build.xml` by entering `ant` from the command line.

To use one of the batch files:

- 1 Use a command line window to `cd` to the `/ssf/samples/wsclient` directory.
- 2 Type the filename for the batch file you want to execute.  
To see a list of legal commands and arguments for one of the `teaming-service` or `facade` batch files, type only the filename of the batch file, then press the Return key.
- 3 On the same line, just after the name of the batch file, type a command name and desired arguments.
- 4 Press the Return key.

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

## 1.3 Authentication

Before determining how to connect your client application to the server, it is important to decide on the authentication method that you want to use. Teaming and its Web services support three types of authentication:

- ♦ [Section 1.3.1, “HTTP Basic Authentication Access \(ssr\),” on page 14](#)
- ♦ [Section 1.3.2, “Web Services Security Access \(ssf\),” on page 14](#)

### 1.3.1 HTTP Basic Authentication Access (ssr)

For basic authentication, use calls from your client application to pass a username and password as you establish an HTTP session. Then, perform SOAP calls or calls using the client-side routines. If you want to use basic authentication, you must use the `/ssr/secure/ws` endpoint when connecting to the server.

HTTP Basic Authentication is the existing transport authentication to authenticate the Web services client. HTTP Basic Authentication uses a username and password to authenticate a service client to a secure endpoint. To use this authentication mechanism, use `/ssr/secure/ws` endpoint. To enable this service on the Teaming side, select the *Enable Basic Authentication (recommended)* check box during product installation.

See [Section 1.4, “Server Endpoints,” on page 15](#), for more information about connecting to the server.

### 1.3.2 Web Services Security Access (ssf)

For WSS authentication, you need to place the authentication information (username and password) in the SOAP calls. If you want to use this method of authentication, use the `/ssf/ws` endpoint to connect to the server.

Web Services Security (WSS) is a standard protocol from Oasis\* that provides a means for applying security to Web services. Unlike security mechanisms that rely on the use of transport layer services, WSS provides authentication at the message layer by using a SOAP header. To use this authentication mechanism, use `/ssf/ws` endpoint. The deprecated Web services operation is accessed only through this mechanism. This service is enabled on the Teaming side by selecting the *Enable WSS Authentication (recommended)* check box during product installation.

If you choose to use WSS authentication instead of HTTP basic authentication:

- ♦ Use the `teamingservices-client-with-call.bat` client and its sources to see an example of this type of authentication.
- ♦ You must use the `/ssf/ws` endpoint (see [Section 1.4, “Server Endpoints,” on page 15](#), for more information).
- ♦ You must use password-text methods.

Password-digest is still supported in Teaming 2.0 and earlier but support is dropped with Teaming 2.1. We strongly recommend that you use only the password-text method.

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

On the server side, the security framework allows Teaming to retrieve the clear-text password from the operation by using an application programming interface (API) call. Teaming applies its internal password encryptor and compares the result with the password stored in the database for the user when the password is retrieved.

Although it is easy to code, this method is not secure, because the password is transmitted in plain text. Systems requiring a higher level of security should connect to Teaming over SSL.

To use this service with the `teamingservice-client-with-call.bat`, edit the script and set the value of the `-Dauthmethod` switch to `wss_text`.

See [Section 1.4, “Server Endpoints,” on page 15](#), for more information about connecting to the server.

## 1.4 Server Endpoints

An endpoint is the URL that you use to connect your client application to the Teaming server. Depending on the authentication method you want to use and other factors, you must choose one of the following four endpoints to specify in your client application:

- ♦ **/ssf/ws/TeamingServiceV1:** Use this endpoint if you want to use WSS authentication with the latest Web services operations. See [Section 1.3, “Authentication,” on page 14](#).
- ♦ **/ssf/ws/Facade:** Use this endpoint if you want to use the deprecated Web services operation. This endpoint requires WSS authentication.
- ♦ **/ssr/secure/ws/TeamingServiceV1:** Use this endpoint only if you are using HTTP Basic Authentication with the latest Web services operations.
- ♦ **/ssr/token/ws/TeamingServiceV1:** Use this endpoint when you are making a Web services call as a remote application.
- ♦ **/ssr/ws/TeamingServiceV1:** Use this endpoint when you want to access Teaming as an anonymous user (not specifying any username or password).

## 1.5 Categories of Operations

To assist you in locating the operation you need to perform, the name of each operation is prefaced with its category name. For example, one category is called `folder`, and one operation within that category is `folder_getEntry`.

The following categories of Web services operations are available:

- ♦ **binder:** Operations that are specific to workspaces, common to workspaces and folders, or that are to be applied to the workspace tree beginning at a specific node in the tree.
- ♦ **definition:** Operations for obtaining and using definitions. Definitions are created by using the designers within the Teaming UI.
- ♦ **folder:** Operations that affect only folders and their contents (entries and comments).
- ♦ **ical:** The operation that adds a calendar entry.
- ♦ **ldap:** Operations that work with LDAP data.
- ♦ **license:** Operations used for license compliance.
- ♦ **migration:** Operations that assist migration from the SiteScape Forum<sup>®</sup> product to Teaming. See [Appendix C, “Migrating from Forum to Kablink Teaming,” on page 179](#).

- ♦ **profile:** Operations affecting users and groups.
- ♦ **search:** Operations that assist in locating information based on criteria other than the defined type.
- ♦ **template:** Operations that create workspaces and folders, or that get lists of available templates. (To create a completely configured folder, use `template_addBinder` and not `binder_addBinder`.)
- ♦ **zone:** Operations that work with different Teaming starting points within the same installation. Each starting point contains its own unique workspace hierarchy.

## 1.6 Client Stubs

A stub is a proxy on the client. The stub code performs SOAP calls to the server. Teaming provides pre-generated Java stub classes that are included in the Kablink Teaming Web Services Java client library. To obtain the Kablink Teaming Web Services Java client library, see [Section 1.7.1, “Working with Java Objects,” on page 17](#).

The following example is the `deleteFolderEntry` method defined in the sample Java class `TeamingServiceClientWithStub.java` file. The `TeamingServiceClientWithStub.java` file makes SOAP calls to Teaming through the use of the pre-generated Java stub classes mentioned above. This method uses the `folder_deleteEntry` Web services operation to delete an entry from Teaming. This code assumes that your client is running on the same machine that is running the Teaming server (localhost). It uses the Basic Authentication mechanism for authentication.

Consider the following code:

```
private static final String TEAMING_SERVICE_ADDRESS_BASIC = "http://
localhost:8080/ssr/secure/ws/TeamingServiceV1";

private static final String USERNAME = "admin";
private static final String PASSWORD = "test";
.
.
.

public static void deleteFolderEntry(long entryId) throws Exception {
    TeamingServiceSoapServiceLocator locator = new
TeamingServiceSoapServiceLocator();
    locator.setTeamingServiceEndpointAddress(TEAMING_SERVICE_ADDRESS_BASIC);
    TeamingServiceSoapBindingStub stub = (TeamingServiceSoapBindingStub)
locator.getTeamingService();
    WebServiceClientUtil.setUserCredentialBasicAuth(stub, USERNAME, PASSWORD);

    stub.folder_deleteEntry(null, entryId);

    System.out.println("ID of the deleted entry: " + entryId);
}
```

## 1.7 Managing Data

Some operations are less intuitive than others for messages. This section provides additional information for those operations and includes the following subsections:

- ♦ [Section 1.7.1, “Working with Java Objects,” on page 17](#)
- ♦ [Section 1.7.2, “Adding Folders and the Binder Configuration Identifier,” on page 18](#)



- ♦ [Section 1.7.3, “Attaching Files,” on page 19](#)
- ♦ [Section 1.7.4, “Fetching Attachments,” on page 20](#)
- ♦ [Section 1.7.5, “Adding Calendar Entries,” on page 20](#)
- ♦ [Section 1.7.6, “Binder Pages and search\\_getWorkspaceTreeAsXML,” on page 20](#)

## 1.7.1 Working with Java Objects

The Web services operations often pass and return data within model objects as defined within the Kablink® Teaming software. This is beneficial because it cuts down on the amount of code required to prepare, send, receive, and interpret data. For example, parsing XML strings requires more coding. For users who develop Web services client applications in Java, Kablink Teaming provides a client-side library that they can use directly for added convenience. Users who develop Web services client applications in a language other than Java must rely on their own tools for understanding and coding the Kablink Teaming Web interfaces that have been defined and exposed by the corresponding WSDL.

Regardless of the language and tools that are used to develop Web services applications, it is helpful to familiarize yourself with some of the Teaming source code in order to understand the model objects and methods that are used to pass parameters and receive returned data.

To obtain the Kablink Teaming Web Services Java client library, download the Kablink Teaming product distribution tar/zip file, and expand it in a directory. This product distribution tar/zip file contains `teaming-2.*.*-wsclient.zip`. This file contains:

- ♦ The Axis generated Java source and class files for the client side stubs and model classes.

`kablink-teaming-wsclient.jar`

- ♦ Search utility classes that aid in building search queries.

`kablink-teaming-util-search.jar`

- ♦ All third-party libraries needed on the client side to run generated stubs.

The `kablink-teaming-wsclient.jar` file contains the Java source that defines model objects that are passed between the Web services client and the Teaming server as either input arguments to or return values from various Web service operations. These model classes are located in the `org/kablink/teaming/client/ws/model` Java package. A significant number of the model classes build upon the base class `DefinableEntity`. The `TeamingServiceSoapBindingStub.java` class is the main stub class that application programs need to interact with in order to invoke various Web service operations.

To access Java sample programs that use the Kablink Teaming Web Services Java client library, download the Teaming source code from the [Open Community Source page \(http://www.kablink.org/\)](http://www.kablink.org/) and examine the source code and scripts located in the `/ssf/samples/wsclient` directory. For example, the `TeamingServiceClientWithStub.java` class in `/ssf/samples/wsclient/src/org/kablink/teaming/samples/wsclient` demonstrates how to use the supplied stub and model classes to invoke Teaming Web services operations with minimum coding effort.

The `kablink-teaming-wsclient.jar` is also found within the source tree in the `/ssf/ws-client` directory. To implement a client-side application of your own, all of the necessary libraries must be defined as being in your class path. When the sample program is run in `/ssf/samples/wsclient`, the accompanying `build.xml` Ant build script performs this function for you. It can be viewed as a template.

The names of the Web services operations use categories to organize the operations so they are easier for you to locate and understand. In general, the categories describe an item within Teaming that is the focus of the operation, such as folder, entry, binder, or attachments.

## 1.7.2 Adding Folders and the Binder Configuration Identifier

When you add a fully configured folder ([template\\_addBinder \(page 130\)](#)), 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:

- 1 Log in to Teaming as the Teaming administrator.
- 2 Click *Manage > Site Administration*.
- 3 Click *Manage Workspace and Folder Templates*.
- 4 In the *Standard Templates* section, click *Blog*.
- 5 Click *Manage This Target > Configure*.

The Configure Default Settings page is displayed.

**Configure Default Settings**

Current Folder: **Blog** Close

**Blog**

Define a simple URL for this folder or workspace ⓘ

**Currently defined URLs**

Delete the selected URLs

**Define URL**

http://pen.oddteaming/ admin ▼

Add

**Definition inheritance ⓘ**

**Not inheriting definition settings.**

Inherit definitions?

☐ yes ☒ no Apply

**Allowed Views ⓘ**

- ☒ Blog (\_blogFolder)
- ☐ Calendar (\_calendarFolder)
- ☒ Discussion Folder - Movable Columns (\_discussionFolderTable)
- ☒ Discussion Folder - Standard View (\_discussionFolderList)
- ☐ File Folder (\_libraryFolder)
- ☐ Guestbook (\_guestbookFolder)
- ☐ Micro-Blog (\_miniBlogFolder)
- ☐ Milestone Folder (\_milestoneFolder)
- ☐ Mixed File Folder (\_mixedFileFolder)

The following configuration settings are available in the template:

- ♦ Definition inheritance

- ◆ Allowed Views
- ◆ Default View
- ◆ Default Entry Types
- ◆ Workflow Associations
- ◆ Allowed Workflows

At the time of this writing, Teaming does not provide a message that you can use to retrieve the binder configuration identifier for a particular type of folder. Use the following procedure to obtain the binder configuration identifier for the folder you want to create:

- 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, or calendar).
- 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.

The following figure shows an example of the binder configuration information for a blog folder, as found in the HTML source for the *Add new folder* page:

**Figure 1-1** *The Binder Configuration Identifier in Source Code*

```
<tr><td valign="top" nowrap><input type="radio" name="binderConfigId" value="147"
onClick="ss_showAddBinderOptions()"
* Blog&nbsp; <td>
<td valign="top" style="padding-bottom:6px;"><span class="ss_smallprint ss_light">
A blog folder is a forum where entire entries are displayed in reverse chronolog
, based on when they were created. Blogs typically provide information on a particu
r from an individual or small group of authors. Optionally, the blog folder can be
tured so that a larger group can make comments on the entries posted by the origina
```

### 1.7.3 Attaching Files

In Kablink Teaming, attachments are files that are associated with an entry. An entry can have more than one attached file.

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

To attach more than one file to an entry in Teaming, you must use one of the upload operations multiple times. For example, to attach 17 files to an entry in Teaming, you must use `folder_uploadFile` 17 times. Your client source code establishes where in the file system it finds or places files used as attachments to messages.

The `folder_uploadFile` operation 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 Teaming database.

To upload a file into the standard form element used to contain attachments, specify `ss_attachFile1` as the data item name. If you are uploading files into a custom form element, create an instance of that custom entry, use an operation to get the name of the hidden field, then use the name when attaching files to the entry you actually want to affect.

## 1.7.4 Fetching Attachments

When you use `folder_getEntry` 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 places the attached files.

## 1.7.5 Adding Calendar Entries

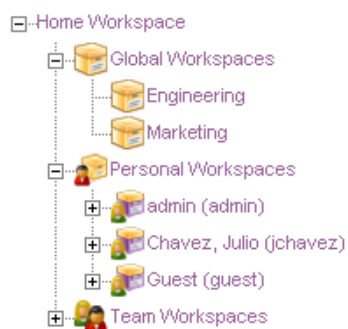
When you pass the `ical_uploadCalendarEntriesWithXML` operation to the server, the Web services framework uses an XML formatted string of iCal data passed as the second parameter to the operation (`<doc><entry>iCal data</entry></doc>`).

## 1.7.6 Binder Pages and `search_getWorkspaceTreeAsXML`

When you use `search_getWorkspaceTreeAsXML` to obtain information about the hierarchical workspace tree, Kablink Teaming 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 graphic shows the workspace tree, which is expanded to show five levels of the workspace hierarchy:

**Figure 1-2** *Workspace Hierarchy Levels as Seen in the UI*



In the graphic, each of the 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 only binder shown at level 3 is the Corporate web site binder. Level 4 binders include folders and the December 2008 redesign workspace. The Calendar binder is located at level 5. If a binder has a plus sign next to it (for example, both the Global workspaces and Personal workspaces binders are preceded by plus signs), it means that there are hierarchy levels of binders that are not displayed in the UI.

If you use `search_getWorkspaceTreeAsXML` to get one level of the tree starting at the Workspaces node, Teaming returns information about Global workspaces, Personal workspaces, and Team workspaces.

As mentioned, some nodes in the tree are pages:

**Figure 1-3** Pages as They Appear in the UI

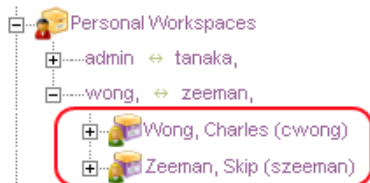


The `/ssf/web/docroot/WEB-INF/classes/config/ssf.properties` file contains a property called `wsTree.maxBucketSize`, which, by default, is set to 25. This means that the maximum number of sub-workspaces allowed is 25. If a folder or workspace has more subplaces, Teaming creates virtual buckets called pages. Each line in [Figure 1-3](#) corresponds to a page. The Personal workspaces workspace has two pages.

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

Consider the following:

**Figure 1-4** A Page Containing Sub-Workspaces



The `wong//zeeman` page contains workspaces. The workspaces listed (*Wong, Charles (cwong)*, and *Zeeman, Skip (szeeman)*) 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="2"` and `pageTuple="charles_wong (cwong)//skip_zeeman (szeeman)"`. 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:

```
search_getWorkspaceTreeAsXML 24 3 "2//charles_wong//skip_zeeman"
```

This code begins at binder number 24, accesses page number 2, and returns two hierarchical levels of data for all users between Charles Wong and Skip Zeeman.

Given the structure of the Teaming 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 (Teaming 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. Teaming 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), Teaming 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.

## 1.8 Extending Teaming Web Services

Because Kablink Teaming 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 Kablink Teaming [online community](http://www.kablink.org) (<http://www.kablink.org>), sharing your code with others, and working with the Novell 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 might need to be redone in future versions of Kablink Teaming because of 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, Kablink Teaming includes an example that provides a structure that enables users of all versions of our software to be able to extend our Web services in the most optimal way, minimizing work that you might need to do to maintain the extensions for every upgrade.

Kablink Teaming includes an extended Web services example, which adds the `folder_getFolderTitle` operation to the base Teaming web services, and also adds the `getFolderTitle` command to the `teamingservice-client-with-call.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 simple directions for establishing the extension.

# Web Service Operations

# A

This section provides alphabetized reference pages for the Web services operations of Kablink® Teaming 2.0 and later.

---

**NOTE:** All examples in this reference section use the Kablink Teaming client library. See [Section 1.6, “Client Stubs,” on page 16](#), for more information about the client library and other ways to call Teaming Web services operations.

---

- ♦ [“admin\\_destroyApplicationScopedToken” on page 26](#)
- ♦ [“admin\\_getApplicationScopedToken” on page 27](#)
- ♦ [“binder\\_addBinder” on page 28](#)
- ♦ [“binder\\_copyBinder” on page 29](#)
- ♦ [“binder\\_deleteBinder” on page 30](#)
- ♦ [“binder\\_deleteTag” on page 31](#)
- ♦ [“binder\\_getBinder” on page 32](#)
- ♦ [“binder\\_getBinderByPathName” on page 33](#)
- ♦ [“binder\\_getFileVersions” on page 34](#)
- ♦ [“binder\\_getFolders” on page 35](#)
- ♦ [“binder\\_getSubscription” on page 36](#)
- ♦ [“binder\\_getTags” on page 37](#)
- ♦ [“binder\\_getTeamMembers” on page 38](#)
- ♦ [“binder\\_indexBinder” on page 39](#)
- ♦ [“binder\\_indexTree” on page 40](#)
- ♦ [“binder\\_modifyBinder” on page 41](#)
- ♦ [“binder\\_moveBinder” on page 42](#)
- ♦ [“binder\\_removeFile” on page 43](#)
- ♦ [“binder\\_setDefinitions” on page 44](#)
- ♦ [“binder\\_setFunctionMembership” on page 45](#)
- ♦ [“binder\\_setFunctionMembershipInherited” on page 46](#)
- ♦ [“binder\\_setOwner” on page 47](#)
- ♦ [“binder\\_setSubscription” on page 48](#)
- ♦ [“binder\\_setTag” on page 49](#)
- ♦ [“binder\\_setTeamMembers” on page 50](#)
- ♦ [“binder\\_uploadFile” on page 51](#)
- ♦ [“definition\\_getDefinitionAsXML” on page 52](#)
- ♦ [“definition\\_getDefinitionByName” on page 53](#)
- ♦ [“definition\\_getDefinitions” on page 54](#)

- ◆ “definition\_getLocalDefinitionByName” on page 55
- ◆ “definition\_getLocalDefinitions” on page 56
- ◆ “folder\_addEntry” on page 57
- ◆ “folder\_addEntryWorkflow” on page 58
- ◆ “folder\_addMicroBlog” on page 59
- ◆ “folder\_addReply” on page 60
- ◆ “folder\_copyEntry” on page 61
- ◆ “folder\_deleteEntry” on page 62
- ◆ “folder\_deleteEntryTag” on page 63
- ◆ “folder\_deleteEntryWorkflow” on page 64
- ◆ “folder\_getEntries” on page 65
- ◆ “folder\_getEntry” on page 66
- ◆ “folder\_getEntryByFileName” on page 67
- ◆ “folder\_getEntryTags” on page 68
- ◆ “folder\_getFileVersions” on page 69
- ◆ “folder\_getSubscription” on page 70
- ◆ “folder\_modifyEntry” on page 71
- ◆ “folder\_modifyWorkflowState” on page 72
- ◆ “folder\_moveEntry” on page 73
- ◆ “folder\_removeFile” on page 74
- ◆ “folder\_reserveEntry” on page 75
- ◆ “folder\_setEntryTag” on page 76
- ◆ “folder\_setRating” on page 77
- ◆ “folder\_setSubscription” on page 78
- ◆ “folder\_setWorkflowResponse” on page 79
- ◆ “folder\_synchronizeMirroredFolder” on page 80
- ◆ “folder\_unreserveEntry” on page 81
- ◆ “folder\_uploadFile” on page 82
- ◆ “folder\_uploadFileStaged” on page 83
- ◆ “ical\_uploadCalendarEntriesWithXML” on page 85
- ◆ “ldap\_synchAll” on page 86
- ◆ “ldap\_synchUser” on page 87
- ◆ “license\_getExternalUsers” on page 88
- ◆ “license\_getRegisteredUsers” on page 89
- ◆ “license\_updateLicense” on page 90
- ◆ “migration\_addBinder” on page 91
- ◆ “migration\_addBinderWithXML” on page 92
- ◆ “migration\_addEntryWorkflow” on page 94



- ◆ “migration\_addFolderEntry” on page 95
- ◆ “migration\_addFolderEntryWithXML” on page 96
- ◆ “migration\_addReply” on page 98
- ◆ “migration\_addReplyWithXML” on page 99
- ◆ “migration\_uploadFolderFile” on page 101
- ◆ “migration\_uploadFolderFileStaged” on page 103
- ◆ “profile\_addGroup” on page 105
- ◆ “profile\_addGroupMember” on page 106
- ◆ “profile\_addUser” on page 107
- ◆ “profile\_addUserWorkspace” on page 108
- ◆ “profile\_deletePrincipal” on page 109
- ◆ “profile\_getFileVersions” on page 110
- ◆ “profile\_getGroup” on page 111
- ◆ “profile\_getGroupByName” on page 112
- ◆ “profile\_getGroupMembers” on page 113
- ◆ “profile\_getPrincipals” on page 114
- ◆ “profile\_getUser” on page 115
- ◆ “profile\_getUserByName” on page 116
- ◆ “profile\_getUsers” on page 117
- ◆ “profile\_getUserTeams” on page 118
- ◆ “profile\_modifyGroup” on page 119
- ◆ “profile\_modifyUser” on page 120
- ◆ “profile\_removeFile” on page 121
- ◆ “profile\_removeGroupMember” on page 122
- ◆ “profile\_uploadFile” on page 123
- ◆ “search\_getFolderEntries” on page 124
- ◆ “search\_getHotContent” on page 125
- ◆ “search\_getTeams” on page 126
- ◆ “search\_getWorkspaceTreeAsXML” on page 127
- ◆ “search\_search” on page 128
- ◆ “template\_addBinder” on page 130
- ◆ “template\_getTemplates” on page 131
- ◆ “zone\_addZone” on page 132
- ◆ “zone\_deleteZone” on page 133
- ◆ “zone\_modifyZone” on page 134

# admin\_destroyApplicationScopedToken

Destroys an application-scoped token.

## Syntax

```
public void admin_destroyApplicationScopedToken( String accessToken, String token );
```

## Description

The `admin_destroyApplicationScopedToken` operation destroys a previously acquired application scoped token.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **token**

The string representation of the previously acquired application-scoped token that you want to destroy.

### **return\_value**

None.

# admin\_getApplicationScopedToken

Requests an application-scoped token on behalf of the user.

## Syntax

```
public String admin_getApplicationScopedToken( String accessToken, long applicationId, long userId );
```

## Description

The `admin_getApplicationScopedToken` operation requests the system to create and return an application-scoped token on behalf of the user. The token is subsequently utilized by the application.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **applicationId**

The identifier of the application set up with the Teaming system.

### **userId**

The identifier of the user on whose behalf you want the token to be created.

### **return\_value**

A string representation of the requested token.

# binder\_addBinder

Adds an unconfigured binder to the workspace tree hierarchy.

## Syntax

```
public long binder_addBinder( String accessToken, Binder binder );
```

## Description

The `binder_addBinder` operation adds either a workspace or folder to the hierarchy.

To add a fully configured binder, use the `template_addBinder` operation instead.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part implementing a remote application, or the null value.

### **binder**

Data and methods for the Java `Binder` object, defined in the Teaming source code.

### **return\_value**

The identifier of the new binder.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))
- ♦ [template\\_addBinder](#) (page 130)

# binder\_copyBinder

Creates a new binder identical to an existing one.

## Syntax

```
public long binder_copyBinder( String accessToken, long sourceId, long destinationId,  
boolean cascade);
```

## Description

The `binder_copyBinder` operation copies an existing workspace or folder, and creates a new one.

Teaming automatically copies all non-binder content (entries and comments). As an option, you can replicate the source binder's sub-binders (sub-workspaces or sub-folders).

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **sourceId**

The identifier of the binder you want to copy.

### **destinationId**

The binder identifier of the parent for the new workspace or folder.

### **cascade**

A Boolean value indicating whether you want to copy the source binder's sub-binders (sub-workspaces and sub-folders).

### **return\_value**

The identifier of the new binder.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# binder\_deleteBinder

Deletes a binder.

## Syntax

```
public void binder_deleteBinder( String accessToken, long binderId, boolean deleteMirroredSource );
```

## Description

The `binder_deleteBinder` operation deletes a workspace or folder.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **binderId**

The binder identifier of the workspace or folder you want to delete.

### **deleteMirroredSource**

Deletes the source directory, if the folder being deleted is a mirrored folder.

### **return\_value**

None.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# binder\_deleteTag

Removes a tag from a binder.

## Syntax

```
public void binder_deleteTag( String accessToken, long binderId, String tagId );
```

## Description

The `binder_deleteTag` operation removes a tag from a workspace or folder.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **binderId**

The identifier of the binder that applies the tag you want to remove.

### **tagId**

The tag you want to remove.

### **return\_value**

None.

## Example

```
public static void checkTags(long binderId) throws Exception { ...  
    Tag[] tags = setupTags(binderId);  
    for (int i=0; i<tags.length; ++i) { stub.binder_setTag(null, tags[i]); }  
    tags = stub.binder_getTags(null, binderId);  
    validateTags(tags);  
    stub.binder_deleteTag(null, binderId, tags[0].getId());  
}
```

This code is taken from the source code for the `teaming-service-client-with-stub.bat` file.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# binder\_getBinder

Accepts a binder identifier to get information about a binder.

## Syntax

```
public Binder binder_getBinder( String accessToken, long binderId, boolean includeAttachments );
```

## Description

The `binder_getBinder` operation gets information about a workspace or folder.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **binderId**

The binder identifier for the workspace or folder for which you want information.

### **includeAttachments**

A Boolean value that indicates whether you want Teaming to return attached files.

By default, workspaces do not include attached files. However, users can use the designers to define workspaces that do include attached files.

### **return\_value**

A Binder Java object that contains data and methods for the requested binder.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))



# binder\_getBinderByPathName

Accepts a directory specification to get information about a binder.

## Syntax

```
public Binder binder_getBinderByPathName( String accessToken, String pathName,  
boolean includeAttachments );
```

## Description

The `binder_getBinderByPathName` operation uses a workspace-hierarchy path name to get information about a workspace or folder.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **pathName**

The titles of the binder for which you want information, preceded by the titles of all of its parents, separated by slashes:

`Workspaces / Global workspaces / wsOrfolderTitle / ... / titleTargetWS`

### **includeAttachments**

A Boolean value that indicates whether you want Teaming to return attached files.

By default, workspaces do not include attached files. However, users can use the designers to define workspaces that do include attached files.

### **return\_value**

A Binder Java object that contains data and methods for the requested binder.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,”](#) on [page 17](#))

# binder\_getFileVersions

Returns information about the versions of a file.

## Syntax

```
public void binder_getFileVersions( String accessToken, long binderID, String fileName );
```

## Description

The `binder_getFileVersion` operation retrieves information about the versions of a file associated with a workspace or folder.

By default, workspaces and folders do not contain files, but users can alter definitions by using the designers in the user interface so that a custom workspace or folder can include one or more files.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **binderID**

The binder identifier for the workspace or folder.

### **filename**

The filename of the file you want to retrieve version information about.

### **return\_value**

A File Version Java object that contains information about the file versions.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# binder\_getFolders

Returns a folder collection for a binder's sub-folders.

## Syntax

```
public FolderCollection binder_getFolders( String accessToken, long binderId,int firstRecord, int maxRecords );
```

## Description

The `binder_getFolders` operation returns a folder collection, which contains information about the sub-folders of a specified binder.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **binderId**

The binder identifier for the workspace or folder for which you want information about its sub-folders.

### **firstRecord**

The index of the first record whose folder information you want to obtain. The index is 0-based.

### **maxRecord**

The maximum number of folders whose information should be returned. Specify -1 for unlimited.

### **return\_value**

A FolderCollection Java object containing information about the sub-folders.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# binder\_getSubscription

Obtains subscription information about a binder.

## Syntax

```
public Subscription binder_getSubscription( String accessToken, long binderId );
```

## Description

The `binder_getSubscription` operation returns subscription information for a specified binder. When a user subscribes to a binder, that person receives e-mail notifications.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **binderId**

The binder identifier for the workspace or folder for which you want subscription information.

### **return\_value**

A Subscription Java object containing subscription information.

## Example

```
public static void checkBinderSubscriptions(long binderId) throws Exception {
    ...
    Subscription subscription = setupSubscription(binderId);
    stub.binder_setSubscription(null, binderId, subscription);
    subscription = stub.binder_getSubscription(null, binderId);
    validateSubscription(subscription);
}
```

This code is taken from the source code for the `teamingservice-client-with-stub.bat` file.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# binder\_getTags

Obtains tags applied to a binder.

## Syntax

```
public Tag[] binder_getTags( String accessToken, long binderId );
```

## Description

The `binder_getTags` operation gets tag information for a specified binder.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **binderId**

The binder identifier for the workspace or folder whose tag information you want.

### **return\_value**

An array of Tag Java objects, each containing information about one of the tags applied to the binder.

## Example

```
public static void checkTags(long binderId) throws Exception { ...
    Tag[] tags = setupTags(binderId);
    for (int i=0; i<tags.length; ++i) { stub.binder_setTag(null, tags[i]); }
    tags = stub.binder_getTags(null, binderId);
    validateTags(tags);
    stub.binder_deleteTag(null, binderId, tags[0].getId());
}
```

This code is taken from the source code for the `teamingservice-client-with-stub.bat` file.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# binder\_getTeamMembers

Obtains information about the members of a team assigned to a specified binder.

## Syntax

```
public TeamMemberCollection binder_getTeamMembers( String accessToken, long binderId );
```

## Description

The `binder_getTeamMembers` operation obtains information about the members of a team assigned to a specified binder.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **binder**

The binder identifier for the workspace or folder for which you want information about team members.

### **return\_value**

A `TeamMemberCollection` Java object containing information about the members of a team assigned to the specified binder.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# binder\_indexBinder

Indexes a binder and its content.

## Syntax

```
public void binder_indexBinder( String accessToken, long binderId );
```

## Description

The `binder_indexBinder` operation indexes a workspace or folder (and its contents), optimizing the ability of Teaming to search its contents. This operation does not index sub-binders.

To index sub-binders, use the `binder_indexTree` operation.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **binderId**

The binder identifier for the workspace or folder that you want to index.

### **return\_value**

None.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))
- ♦ [binder\\_indexTree](#) (page 40)

# binder\_indexTree

Indexes a binder's sub-binders.

## Syntax

```
public Long binder_indexTree( String accessToken, long binderId );
```

## Description

The `binder_indexTree` operation indexes the specified workspace or folder, all sub-binders, and all content in all those binders.

If you want to index a binder and its contents without indexing sub-binders, use the `binder_indexBinder` operation.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **binderId**

The binder identifier for the workspace or folder that indicates the node where you want to begin indexing within the workspace hierarchy.

### **return\_value**

An array of integers, with each integer being the identifier of a binder successfully indexed.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))
- ♦ [binder\\_indexBinder \(page 39\)](#)



# binder\_modifyBinder

Modifies a binder.

## Syntax

```
public void binder_modifyBinder( String accessToken, Binder binder );
```

## Description

The `binder_modifyBinder` operation modifies a workspace or folder.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **binder**

Data and methods for the Binder Java object, defined in the Teaming source code.

### **return\_value**

None.

## Example

```
public static Binder modifyBinder(Binder binder) throws Exception { ...
    binder.setTitle(binder.getTitle() + " (Modified)");
    binder.getDescription().setText(binder.getDescription().getText() + "
    (Modified)"); stub.binder_modifyBinder(null, binder);
    stub.binder_getBinder(null, binder.getId(), true);
    System.out.println("ID of the modified binder: " + binder.getId());
    return binder;
}
```

This code is taken from the source code for the `teamingservice-client-with-stub.bat` file.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# binder\_moveBinder

Moves a binder within the workspace tree hierarchy.

## Syntax

```
public void binder_moveBinder( String accessToken, long binderId, long newParentBinderId );
```

## Description

The `binder_moveBinder` operation moves either a workspace or folder within the workspace hierarchy.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **binderId**

The binder identifier for the workspace or folder that you want to move.

### **newParentBinderId**

The binder identifier of the binder under which you want *binderId* to appear as a sub-binder.

### **return\_value**

None.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# binder\_removeFile

Removes a file from a binder.

## Syntax

```
public void binder_removeFile( String accessToken, long binderId, String fileName );
```

## Description

The `binder_removeFile` operation removes a file from a workspace or folder.

By default, workspaces do not contain files, but users can alter definitions by using the designers in the UI so that a custom workspace can include one or more files.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **binderId**

The binder identifier for the workspace or folder from which you want to remove a file.

### **fileName**

The file name of the file you want to remove from the binder.

### **return\_value**

None.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# binder\_setDefinitions

Associates workflow definitions with entry definitions.

## Syntax

```
public void binder_setDefinitions( String accessToken, long binderId, String[] entryDefinitionIds,  
String [] workflowDefinitionIds );
```

## Description

The `binder_setDefinitions` operation associates entries within the specified binder with workflow processes. (Teaming associates identifiers in the first element of both arrays, the second element of both arrays, the third, and so on.)

When an entry is associated with a workflow process, creation of an entry of that type automatically places the entry into the initial state of the workflow process. By default, workspaces do not contain entries that can be associated with workflow processes. However, users can alter definitions by using the designers in the UI so that a custom workspace can include one or more files.

---

**NOTE:** This operation is an overwrite operation, setting all workflow associations for the folder; you cannot use repeated calls to this operation to set associations incrementally. Set all of the workflow associations for the folder with one call.

---

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **binderId**

The binder identifier for the folder or custom workspace within which you want entries associated with workflow processes.

### **entryDefinitionIds**

An array of definition identifiers for each type of entry to which you want to associate a workflow process.

### **workflowDefinitionIds**

An array of workflow identifiers in the order in which you want them applied to the entry-definition identifiers in *entryDefinitionIds*.

### **return\_value**

None.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# binder\_setFunctionMembership

Applies access-control settings to a binder.

## Syntax

```
public void binder_setFunctionMembership( String accessToken, long binderId,  
FunctionMembership[] functionMemberships);
```

## Description

The `binder_setFunctionMembership` operation provides access-control settings for a folder or workspace. The term function is analogous to an access-control role in the UI.

---

**NOTE:** This operation is an overwrite operation, that sets all function memberships for the folder or workspace; you cannot use repeated calls to this operation to set memberships incrementally. Set all memberships for the workspace or folder with one call.

---

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **binderId**

The binder identifier for the workspace or folder whose access control you want to set.

### **functionMemberships**

An array of `FunctionMembership` Java objects.

### **return\_value**

None.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))
- ♦ [binder\\_setFunctionMembershipInherited](#) (page 46)

# binder\_setFunctionMembershipInherited

Establishes inheritance as the access-control mechanism for a folder or workspace.

## Syntax

```
public void binder_setFunctionMembershipInherited( String accessToken, long binderId,  
boolean inherit );
```

## Description

The `binder_setFunctionMembershipInherited` establishes whether a specified workspace or folder inherits access-control settings from its parent binder.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **binderId**

The binder identifier for the workspace or folder for which you want to establish the inheritance setting for access control.

### **inherit**

A true or false value that establishes whether the binder inherits its access-control settings.

### **return\_value**

None.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))
- ♦ [binder\\_setFunctionMembership](#) (page 45)

# binder\_setOwner

Establishes the owner of the binder.

## Syntax

```
public long binder_setOwner( String accessToken, long binderId, long userId);
```

## Description

The `binder_setOwner` operation establishes the specified user as the owner of a workspace or folder.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **binderId**

The binder identifier for the workspace or folder whose owner you want to establish.

### **userId**

The identifier for the user whom you want to be the owner of the binder.

### **return\_value**

None.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# binder\_setSubscription

Establishes e-mail settings for a binder.

## Syntax

```
public void binder_setSubscription( String accessToken, long binderId, Subscription subscription );
```

## Description

The `binder_setSubscription` operation establishes subscription settings for a workspace or folder. When a user subscribes to a binder, that person receives e-mail notifications.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **binderId**

The binder identifier for the workspace or folder whose subscription you want to set.

### **subscription**

A Subscription Java object containing information used to establish e-mail notification settings for the specified binder.

### **return\_value**

None.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))



# binder\_setTag

Applies a tag for a binder.

## Syntax

```
public void binder_setTag( String accessToken, Tag tag );
```

## Description

The `binder_setTag` operation applies a tag to a workspace or folder.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **tag**

A Tag Java object that contains information applying the tag to a workspace or folder.

### **return\_value**

None.

## Example

```
public static void checkTags(long binderId) throws Exception { ...  
    Tag[] tags = setupTags(binderId);  
    for (int i=0; i<tags.length; ++i) { stub.binder_setTag(null, tags[i]); }  
    tags = stub.binder_getTags(null, binderId);  
    validateTags(tags);  
    stub.binder_deleteTag(null, binderId, tags[0].getId());  
}
```

This code is taken from the source code for the `teamingservice-client-with-stub.bat` file.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# binder\_setTeamMembers

Establishes members of a team for a binder.

## Syntax

```
public void binder_setTeamMembers( String accessToken, long binderId, String[] teamMembers );
```

## Description

The `binder_setTeamMembers` operation establishes members of the team for a specified workspace or folder.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **binder**

The binder identifier for the workspace or folder for which you want to establish team membership.

### **return\_value**

None.

### **teamMembers**

Names of the team members.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# binder\_uploadFile

Uploads a file into a binder.

## Syntax

```
public void binder_uploadFile( String accessToken, long binderId,  
String formDataItemName, String fileName );
```

## Description

The `binder_uploadFile` operation performs an action equivalent to using the UI to upload a file to either a workspace or folder. You can attach only one file at a time; call this operation multiple times to attach more than one file to the binder.

By default, workspaces do not include attached files. However, users can use the designers to define workspaces that do include attached files.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **binderId**

The binder identifier for the workspace or folder into which you want to upload a file.

### **formDataItemname**

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 data in the Teaming database; a `hidden` HTML tag communicates this file mapping to the server.

The `name` value for the standard entry element containing attached files is `ss_attachFile1`. 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.

If you are uploading to a folder file, specify `upload` as an argument to this parameter to make this attachment the primary file for the entry.

### **fileName**

A string containing the filename of the file you want to upload to the binder.

### **return\_value**

None.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# definition\_getDefinitionAsXML

Obtains information about a definition.

## Syntax

```
public String definition_getDefinitionAsXML( String accessToken, String definitionId );
```

## Description

The `definition_getDefinitionAsXML` operation returns a string of XML containing information about a specified definition.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **definitionId**

The definition identifier of the item about which you want information.

### **return\_value**

An XML string containing information about the definition. This XML is free form; it does not have a firm, established schema.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))
- ♦ [definition\\_getDefinitionByName](#) (page 53)
- ♦ [definition\\_getDefinitions](#) (page 54)
- ♦ [definition\\_getDefinitionByName](#) (page 53)
- ♦ [definition\\_getLocalDefinitions](#) (page 56)

# definition\_getDefinitionByName

Obtains information about a global definition.

## Syntax

```
public DefinitionBrief definition_getDefinitionByName( String accessToken,  
String definitionName );
```

## Description

The `definition_getDefinitionByName` operation obtains information about a global definition by using the definition name. To get information about a local definition, use the `definition_getLocalDefinitionByName` operation.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **definitionName**

The descriptive word or phrase used to name the global definition.

### **return\_value**

A `DefinitionBrief` Java object containing information about the global definition.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))
- ♦ [definition\\_getDefinitions](#) (page 54)
- ♦ [definition\\_getLocalDefinitionByName](#) (page 55)

# definition\_getDefinitions

Obtains all global definitions in the installation.

## Syntax

```
public DefinitionCollection definition_getDefinitions( String accessToken );
```

## Description

The `definition_getDefinitions` operation obtains information about all global definitions in the installation. To get information about local definitions, use the `definition_getLocalDefinitions` operation.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **return\_value**

A `DefinitionCollection` Java object containing information about all global definitions in the installation.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))
- ♦ [definition\\_getLocalDefinitionByName](#) (page 55)
- ♦ [definition\\_getLocalDefinitions](#) (page 56)

# definition\_getLocalDefinitionByName

Obtains information about a local definition.

## Syntax

```
public DefinitionBrief definition_getLocalDefinitionByName( String accessToken, long binderId,  
String name, boolean includeAncestors );
```

## Description

The `definition_getLocalDefinitionByname` operation obtains information about a local definition by using a name. To get information about a global definition, use the `definition_getDefinitionByname` operation.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **binderId**

The binder identifier for the workspace or folder whose local definition you want.

### **name**

The word or phrase used to name the local definition.

### **includeAncestors**

A Boolean value that indicates whether Teaming should check local definitions inherited from ancestor workspaces and folders, which are located higher in the hierarchy than the specified binder. If you specify false, Teaming checks only the local definitions created within the specified binder.

### **return\_value**

A DefinitionBrief Java object containing information about the definition that matches *name*.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))
- ♦ [definition\\_getLocalDefinitionByName](#) (page 55)
- ♦ [definition\\_getLocalDefinitions](#) (page 56)

# definition\_getLocalDefinitions

Obtains information about all local definitions.

## Syntax

```
public DefinitionCollection definition_getLocalDefinitions( String accessToken, long binderId,  
boolean includeAncestors);
```

## Description

The `definition_getLocalDefinitions` operation obtains information about the local definitions for a specified binder. If you want information about all global definitions in the installation, use the `definition_getDefinitions` operation.

If you want to add a fully configured binder, use `template_addBinder` instead.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **binderId**

The binder identifier for the workspace or folder whose local definitions you want.

### **includeAncestors**

A Boolean value that indicates whether Teaming should include local definitions inherited from ancestor workspaces and folders, which are located higher in the hierarchy than the specified binder. If you specify false, Teaming includes only the local definitions created within the specified binder.

### **return\_value**

A `DefinitionCollection` Java object that contains information about the binder's local definitions.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))
- ♦ [definition\\_getDefinitions](#) (page 54)
- ♦ [definition\\_getLocalDefinitionByName](#) (page 55)



# folder\_addEntry

Adds an entry to a folder.

## Syntax

```
public long folder_addEntry( String accessToken, FolderEntry entry, String attachedFileName );
```

## Description

The `folder_addEntry` operation adds an entry to a folder.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **entry**

A FolderEntry Java object containing information that Teaming uses to create the new entry.

### **attachedFileName**

(Optional) A string containing the filename of a file to attach to the new entry. If you are not attaching a file, specify the null value for this argument.

### **return\_value**

The entry identifier of the newly created entry.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,”](#) on [page 17](#))

# folder\_addEntryWorkflow

Initiates a workflow process for a folder entry.

## Syntax

```
public void folder_addEntryWorkflow( String accessToken, long entryId, String  
workflowDefinitionId );
```

## Description

The `folder_addEntryWorkflow` operation initiates a workflow process for a folder entry.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **entryId**

The entry identifier of the folder entry with which you want to initiate a workflow process.

### **workflowDefinitionId**

The definition identifier of the workflow process that you want to initiate for the specified folder entry.

### **return\_value**

None.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# folder\_addMicroBlog

Adds a micro-blog entry to a folder.

## Syntax

```
public long folder_addMicroBlog( String accessToken, string text );
```

## Description

The `folder_addMicroBlog` operation adds a micro-blog entry to a folder.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **text**

A string containing the text of the micro-blog to create.

### **return\_value**

The entry identifier of the newly created micro-blog entry.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# folder\_addReply

Adds a comment to a folder entry.

## Syntax

```
public long folder_addReply( String accessToken, long parentEntryId, FolderEntry reply,  
String attachedFileName );
```

## Description

The `folder_addReply` operation adds a comment to a folder entry.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **parentEntryId**

The entry identifier of the entry or comment that is to be the parent of the comment you are adding.

### **reply**

A `FolderEntry` Java object containing information that yyyy uses to create the new comment.

### **attachedFileName**

The filename of a file you are attaching to the comment. If you are not attaching a file, specify the null value for this argument.

### **return\_value**

The entry identifier of the newly created comment.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# folder\_copyEntry

Copies a folder entry.

## Syntax

```
public long folder_copyEntry( String accessToken, long entryId, long parentFolderId );
```

## Description

The `folder_copyEntry` operation copies a folder entry.

---

**NOTE:** This operation does not copy workflow information for an entry.

---

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **entryId**

The entry identifier of the folder entry you want to copy.

### **parentFolderId**

The folder identifier of the folder you want to contain the copied entry.

### **return\_value**

The entry identifier of the new entry created by copying the existing entry.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# folder\_deleteEntry

Deletes a folder entry.

## Syntax

```
public void folder_deleteEntry( String accessToken, long entryId );
```

## Description

The `folder_deleteEntry` operation deletes a folder entry.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **entryId**

The entry identifier of the folder entry you want to delete.

### **return\_value**

None.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# folder\_deleteEntryTag

Removes a tag from a folder entry.

## Syntax

```
public void folder_deleteEntryTag( String accessToken, long entryId, String tagId );
```

## Description

The `folder_deleteEntryTag` operation removes a tag from a folder entry.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **entryId**

The entry identifier of the entry from which you want to remove a tag.

### **tagId**

A string identifying the tag you want to remove from the entry.

### **return\_value**

None.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# folder\_deleteEntryWorkflow

Removes a workflow from an entry.

## Syntax

```
public void folder_deleteEntryWorkflow( String accessToken, long entryId,  
String workflowDefinitionId );
```

## Description

The `folder_deleteEntryWorkflow` operation removes a workflow process from a folder entry.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **entryId**

The entry identifier of the entry for which you want to remove a workflow process.

### **workflowDefinitionId**

A string containing the definition identifier for the workflow process you want to remove from the entry.

### **return\_value**

None.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))



# folder\_getEntries

Obtains information about the entries within a specified folder.

## Syntax

```
public FolderEntryCollection folder_getEntries( String accessToken, long binderID, int firstRecord,  
int maxRecords );
```

## Description

The `folder_getEntries` operation obtains information about the entries contained in a folder.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **binderId**

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

### **firstRecord**

The index of the first record whose information you want to obtain. The index is 0-based.

### **maxRecords**

The maximum number of entries whose information should be returned. Specify -1 for unlimited.

### **return\_value**

A `FolderEntryCollection` Java object containing information about the entries contained within the folder.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# folder\_getEntry

Accepts an entry identifier to get information about an entry in a folder.

## Syntax

```
public FolderEntry folder_getEntry( String accessToken, long entryId,  
boolean includeAttachments );
```

## Description

The `folder_getEntry` operation obtains information about an entry in a folder.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **entryId**

The entry identifier of the entry about which you want information.

### **includeAttachments**

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

### **return\_value**

A `FolderEntry` Java object that contains information about the specified entry.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# folder\_getEntryByFileName

Accepts a filename to get information about an entry.

## Syntax

```
public FolderEntry folder_getEntryByFileName( String accessToken, long binderId,  
String fileName, boolean includeAttachments );
```

## Description

The `folder_getEntryByFileName` operation obtains information about an entry in a folder by using the entry's file name.

Although this operation is most useful for file folders, it works for any folder that requires that all filenames within the folder to be unique.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **binderId**

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

### **fileName**

The name of the file that corresponds with the entry for which you want information.

### **includeAttachment**

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

### **return\_value**

A FolderEntry Java object that contains information about the specified entry.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# folder\_getEntryTags

Obtains information about an entry's tags.

## Syntax

```
public Tag[] folder_getEntryTags( String accessToken, long entryId );
```

## Description

The `folder_getEntryTags` operation gets information about each of the tags applied to a folder entry.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **entryId**

The identifier of the entry about whose tags you want information.

### **return\_value**

An array of Tag Java objects, where each object contains information about one tag applied to the entry.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# folder\_getFileVersions

Returns information about the versions of a file.

## Syntax

```
public void folder_getFileVersions( String accessToken, long entryId, String fileName );
```

## Description

The `folder_getFileVersions` operation retrieves information about the versions of a file associated with an entry.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **entryId**

The entry identifier of the entry.

### **fileName**

The filename of the file you want to retrieve version information about.

### **return\_value**

A File Versions Java object containing information about the file versions.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,”](#) on [page 17](#))

# folder\_getSubscription

Obtains subscription information for a specified folder.

## Syntax

```
public Subscription folder_getSubscription( String accessToken, long entryId );
```

## Description

The `folder_getSubscription` operation gets information about the e-mail notification settings for a specified folder.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **entryId**

The identifier for the entry whose subscription information you want.

### **return\_value**

A Subscription Java object that contains information about e-mail notification settings for the specified folder.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# folder\_modifyEntry

Modifies an entry in a folder.

## Syntax

```
public void folder_modifyEntry( String accessToken, FolderEntry entry );
```

## Description

The `folder_modifyEntry` operation modifies the contents of a folder entry.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **entry**

A `FolderEntry` Java object containing the information to apply to the existing folder entry.

### **return\_value**

None.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# folder\_modifyWorkflowState

Changes the workflow state of an entry.

## Syntax

```
public void folder_modifyWorkflowState( String accessToken, long entryId, long StateId, String  
toState );
```

## Description

The `folder_modifyWorkflowState` operation changes the workflow state of a folder entry.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **entryId**

The identifier of the entry whose workflow state you want to change.

### **stateID**

The token ID of the current workflow state from which you want the entry to transition to the new state.

### **toState**

A string identifying your desired workflow state.

### **return\_value**

None.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))



# folder\_moveEntry

Moves an entry within the folder-tree hierarchy.

## Syntax

```
public void folder_moveEntry( String accessToken, long entryId, long parentId );
```

## Description

The `folder_moveEntry` operation moves an entry to be under a new parent within the folder hierarchy.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **entryId**

The identifier of the entry you want to move.

### **parentId**

The identifier of the folder that is to be the new parent of the specified entry.

### **return\_value**

None.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,”](#) on [page 17](#))

# folder\_removeFile

Removes a file attachment from an entry.

## Syntax

```
public void folder_removeFile( String accessToken, long entryId, String fileName );
```

## Description

The `folder_removeFile` operation removes a file attachment from an entry in a folder.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **entryId**

The identifier of the entry that includes the attachment you want to remove.

### **fileName**

A string containing the filename of the attachment you want to remove from the entry.

### **return\_value**

None.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# folder\_reserveEntry

Reserves an entry.

## Syntax

```
public void folder_reserveEntry( String accessToken, long entryId );
```

## Description

The `folder_reserveEntry` operation reserves an entry in a folder, preventing others from modifying it.

Users reserve and release an entry in the UI using the *Reserve* and *Unreserve* menu items.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **entryId**

The identifier of the entry you want to reserve.

### **return\_value**

None.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# folder\_setEntryTag

Applies a tag to a folder entry.

## Syntax

```
public void folder_setEntryTag( String accessToken, Tag tag );
```

## Description

The `folder_setEntryTag` operation applies a tag to a folder entry.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **tag**

A Tag Java object containing information about the tag you want to apply.

### **return\_value**

None.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# folder\_setRating

Sets a rating for a folder entry.

## Syntax

```
public void folder_setRating( String accessToken, long entryId, long value );
```

## Description

The `folder_setRating` operation applies a “star” rating to an entry.

In the UI, entries can have ratings that range from a lowest value of 1 star to the highest value of 5 stars.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **entryId**

The identifier of the entry for which you want to apply a rating.

### **ratingValue**

An integer indicating how many stars you want to set as the rating.

### **return\_value**

None.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# folder\_setSubscription

Establishes subscription settings for an entry.

## Syntax

```
public void folder_setSubscription( String accessToken, long entryId, Subscription subscription );
```

## Description

The `folder_setSubscription` operation establishes settings for e-mail notifications for a specified entry.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **entryId**

The identifier of the entry for which you want to set subscription information.

### **subscription**

A Subscription Java object that contains subscription information to be applied to the specified entry.

### **return\_value**

None.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# folder\_setWorkflowResponse

Applies an answer to a workflow question for a specified entry.

## Syntax

```
public void folder_setWorkflowResponse( String accessToken, long entryId,  
long stateId, String question, String response );
```

## Description

The `folder_setWorkflowResponse` operation establishes an answer for a workflow question for a specified entry.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **entryId**

The identifier of the entry that is currently in the workflow state within which you want to apply an answer to a question.

### **stateId**

The token ID of the current workflow state that defines the question that you want to affect.

### **question**

A string identifying the question that you are providing an answer to.

### **response**

A string identifying the response you want to apply to the workflow question.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,”](#) on [page 17](#))

# folder\_synchronizeMirroredFolder

Synchronizes a mirrored folder with its source folder.

## Syntax

```
public void folder_synchronizeMirroredFolder( String accessToken, long binderId );
```

## Description

The `folder_synchronizeMirroredFolder` operation synchronizes a mirrored folder with its source folder.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **binderId**

The identifier of the mirrored folder that you want to synchronize.

### **return\_value**

None.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))



# folder\_unreserveEntry

Releases a locked entry.

## Syntax

```
public void folder_unreserveEntry( String accessToken, long entryId );
```

## Description

The `folder_unreserveEntry` operation releases a locked entry.

Users reserve and release an entry in the UI by using the *Reserve* and *Unreserve* menu items.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **entryId**

The identifier of the entry that you want to release from its lock.

### **return\_value**

None.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# folder\_uploadFile

Uploads a file as an attachment to an entry.

## Syntax

```
public void file_uploadFile( String accessToken, long entryId, String formDataItemName,  
String fileName );
```

## Description

The `file_uploadFile` operation uploads a file as an attachment to an entry. You can attach only one file at a time; call this operation multiple times to attach more than one file to the entry.

Because transferring files across the Internet can be time-consuming, you can upload files that have already been moved to a staging area on the Teaming server by using the `folder_uploadFileStaged` operation.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **entryId**

The identifier of the entry that is to include the new attached file.

### **formDataItemName**

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 data in the Teaming database; a `hidden` HTML tag communicates this file mapping to the server.

The `name` value for the standard entry element containing attached files is `ss_attachFile1`. If you want to upload a file into a custom form element you defined by using the designers, you need to look up the `name` identifier for that form element.

If you are uploading to a folder file, specify `upload` as an argument to this parameter to make this attachment the primary file for the entry.

### **fileName**

A string containing the filename of the file you want to attach to the entry.

### **return\_value**

None.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))
- ♦ [folder\\_uploadFileStaged](#) (page 83)

# folder\_uploadFileStaged

Locates a locally stored file and attaches it to an entry.

## Syntax

```
public void file_uploadFileStaged( String accessToken, long entryId, String formDataItemName,  
String fileName, String stagedFileRelativePath );
```

## Description

As a way to streamline the transfer of files, the `file_uploadFileStaged` operation accesses a file that has been copied locally to the Teaming server, avoiding transferring them over the Internet. The operation then attaches the file to a folder entry in Teaming. In order for the Web services client to utilize this operation, the Teaming administrator must first configure the server to allow this operation by specifying `staging.upload.files.enable` and `staging.upload.files.rootpath` configuration settings in `ssf-ext.properties` file. Because it involves Teaming administrator access to the server environment, this operation is reserved only for major migration projects where individual file uploads through the HTTP protocol do not meet the performance requirements of the project.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **entryId**

An identifier for the entry to which you want to attach a file.

### **formDataItemName**

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 data in the Teaming database; a `hidden` HTML tag communicates this file mapping to the server.

The `name` value for the standard entry element containing attached files is `ss_attachFile1`. If you want to upload a file into a custom form element you defined by using the designers, you need to look up the `name` identifier for that form element.

If you are uploading to a folder file, specify `upload` as an argument to this parameter to make this attachment the primary file for the entry.

### **fileName**

A string containing the filename of the file you want to attach to the entry.

### **stagedFileRelativePath**

A pathname of the file relative to the staging area on the server side. On the Teaming server, the staging directory is designated by the value of the `staging.uploads.files.rootpath` configuration setting. This relative pathname is resolved against the staging directory of the Teaming server to identify the input file.

**return\_value**

None.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))
- ♦ [folder\\_uploadFile](#) (page 82)

# ical\_uploadCalendarEntriesWithXML

Adds a calendar entry to a folder.

## Syntax

```
public void ical_uploadCalendarEntriesWithXML( String accessToken, long folderId, String iCalDataAsXML );
```

## Description

The `ical_uploadCalendarEntriesWithXML` adds a calendar entry using iCal information in an XML string.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **folderId**

The identifier of the folder where you want to add a calendar entry.

### **iCalDataAsXML**

A string containing XML formatted calendar data (`<doc><entry>iCal data</entry>...</doc>`).

### **return\_value**

None.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# ldap\_synchAll

Synchronizes all users with the current information that is in LDAP.

## Syntax

```
public void ldap_synchAll( String accessToken );
```

## Description

The `ldap_synchAll` operation synchronizes all users with the current information that is in LDAP.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **return\_value**

None.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# ldap\_synchUser

Synchronizes one user with the latest information in LDAP for that person.

## Syntax

```
public void ldap_synchUser( String accessToken, long userId );
```

## Description

The `ldap_synchUser` operation synchronizes one user with the latest information in LDAP for that person.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **userId**

The identifier of the user whose information you want synchronized with that person's LDAP data.

### **return\_value**

None.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# license\_getExternalUsers

Obtains a count of external users.

## Syntax

```
public long license_getExternalUsers( String accessToken );
```

## Description

The `license_getExternalUsers` operation obtains a count of legal external users for the current license.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **return\_value**

An integer indicating the number of allowed external users.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))



# license\_getRegisteredUsers

Obtains a count of registered Teaming users.

## Syntax

```
public long license_getRegisteredUsers( String accessToken );
```

## Description

The `license_getRegisteredUsers` operation obtains a count of the current number of registered users on the system.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **return\_value**

An integer that is the count of users currently registered on the system.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# license\_updateLicense

Updates the Teaming license.

## Syntax

```
public void license_updateLicense( String accessToken );
```

## Description

The `license_updateLicense` operation updates the Teaming license.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **return\_value**

None.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# migration\_addBinder

Accepts a Java object to add a binder, allowing preservation of SiteScape Forum<sup>®</sup> data.

## Syntax

```
public long migration_addBinder( String accessToken, Binder binder );
```

## Description

The `migration_addBinder` operation adds either a workspace or folder to the hierarchy, allowing you to specify SiteScape Forum data (such as the person who created the workspace or folder in Forum, the Forum creation date, the user who last modified the workspace or folder in Forum, and the date of the last modification in Forum).

If you prefer to use XML to specify data, use the `migration_addBinderWithXML` operation.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **binder**

Data and methods for the Java Binder object, defined in the Teaming source code. Edit the information in the Binder object to reflect the Forum values.

### **return\_value**

The identifier of the newly created binder.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))
- ♦ [migration\\_addBinderWithXML \(page 92\)](#)

# migration\_addBinderWithXML

Accepts XML to add a binder, allowing preservation of SiteScape Forum data.

## Syntax

```
public long migration_addBinderWithXML( String accessToken, long parentId, String definitionId,  
String inputDataAsXML, String creator, Calendar creationDate, String modifier,  
Calendar modificationDate );
```

## Description

The `migration_addBinderWithXML` operation adds either a workspace or folder to the hierarchy, allowing you to specify SiteScape Forum data (such as the person who created the workspace or folder in Forum, the Forum creation date, the user who last modified the workspace or folder in Forum, and the date of the last modification in Forum).

If you prefer to use a Java object to specify data, use the `migration_addBinder` operation.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **parentId**

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

### **definitionID**

A string that identifies the definition used to create the new binder.

### **inputDataAsXML**

An XML string that provides the data needed to construct the workspace or folder.

### **creator**

A string containing the username of the person who created the workspace or folder in Forum.

### **creationDate**

A Calendar Java object that contains the creation date of the workspace or folder in Forum.

### **modifier**

A string containing the username of the person who last modified the workspace or folder in Forum.

### **modificationDate**

A Calendar Java object that contains the modification date of the workspace or folder in Forum.

### **return\_value**

The identifier of the newly created binder.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,”](#) on [page 17](#))
- ♦ [migration\\_addBinder](#) ([page 91](#))

# migration\_addEntryWorkflow

Associates an entry with a workflow process, allowing preservation of SiteScape Forum data.

## Syntax

```
public void migration_addEntryWorkflow( String accessToken, long binderId, long entryId,  
String definitionId, String startState, String modifier, Calendar modificationDate );
```

## Description

The `migration_addEntryWorkflow` operation associates an entry with a workflow process, while preserving values from a SiteScape Forum installation.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **binderId**

The identifier of the folder containing the entry with which you want to associate a workflow process.

### **entryId**

The identifier of the entry with which you want to associate the workflow process.

### **definitionId**

A string containing the definition identifier for the workflow process you want to associate with the entry.

### **startState**

A string containing the name of the state of the entry as it was last set in the Forum installation.

### **modifier**

A string containing the username of the person who last modified the workflow state in the Forum installation.

### **modificationDate**

A Calendar Java object that contains the date that the workflow state was last modified in the Forum installation.

### **return\_value**

None.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# migration\_addFolderEntry

Accepts a Java object to add an entry to a folder, allowing preservation of SiteScape Forum data.

## Syntax

```
public long migration_addFolderEntry( String accessToken, FolderEntry entry, boolean subscribe );
```

## Description

The `migration_addFolderEntry` operation adds an entry to a folder.

If you prefer to use an XML string to create the new entry, use the `migration_addFolderEntryWithXML` operation.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **entry**

A `FolderEntry` Java object that contains information used to create the new entry, including information from the entry in the Forum installation.

### **subscribe**

A Boolean value that implements the Forum *notify me when someone replies to this entry* feature by establishing a subscription for the entry owner.

### **return\_value**

The identifier of the newly created entry.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))
- ♦ [migration\\_addFolderEntryWithXML](#) (page 96)

# migration\_addFolderEntryWithXML

Accepts XML to add an entry to a folder, allowing preservation of SiteScape Forum data.

## Syntax

```
public long migration_addFolderEntryWithXML( String accessToken, long binderId,  
String definitionId, String inputDataAsXML, String creator, Calendar creationDate,  
String modifier, Calendar modificationDate, boolean subscribe );
```

## Description

The `migration_addFolderEntry` operation adds an entry to a folder, allowing you to preserve data from the entry as it last existed in an installation of SiteScape Forum.

If you prefer to create the entry by using a Java object, use the `migrate_addFolderEntry` operation.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **binderId**

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

### **definitionId**

A string containing the definition identifier for the new entry.

### **inputDataAsXML**

A string containing the XML elements used to construct the new entry.

### **creator**

A string containing the username of the person who created the entry in the Forum installation.

### **creationDate**

A Calendar Java object containing the date the entry was created in the Forum installation.

### **modifier**

A string containing the username of the person who last modified the entry in the Forum installation.

### **modificationDate**

A Calendar Java object containing the date the entry was last modified in the Forum installation.

### **subscribe**

A boolean value that implements the Forum feature “notify me when someone replies to this entry” by establishing a subscription for the entry owner.



**return\_value**

The identifier of the binder for the newly created entry.

**See Also**

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))
- ♦ [migration\\_addFolderEntry](#) (page 95)

# migration\_addReply

Accepts a Java object to add a comment, allowing preservation of SiteScape Forum data.

## Syntax

```
public long migration_addReply( String accessToken, long parentEntryId, FolderEntry reply );
```

## Description

The `binder_addReply` operation adds a comment to an entry or a reply, and allows you to preserve data from the reply as it last appeared in a Forum installation.

If you prefer to add the comment by using XML, use the `migrate_addReplyWithXML` operation.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **parentEntryId**

The identifier of the entry or comment that is the parent of the comment you want to create.

### **reply**

A `FolderEntry` Java object that contains information used to construct the new comment, including data reflecting the reply as it last appeared in the Forum installation.

### **return\_value**

The identifier of the newly created comment.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))
- ♦ [migration\\_addReplyWithXML](#) (page 99)

# migration\_addReplyWithXML

Accepts XML to add a comment, allowing preservation of SiteScape Forum data.

## Syntax

```
public long migration_addReplyWithXML( String accessToken, long binderId, long parentId,  
String definitionId, String inputDataAsXML, String creator, Calendar creationDate, String modifier,  
Calendar modificationDate );
```

## Description

The `migration_addReplyWithXML` operation adds a comment to an entry or to another comment, allowing you to preserve data from the reply as it last appeared in the SiteScape Forum installation.

If you prefer to add the comment by using a Java object, use the `migration_addReply` operation.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **binderId**

The identifier of the folder that contains the entry to which you want to add the comment.

### **parentId**

The identifier of the entry or comment that is to be the parent of the newly created comment.

### **definitionId**

A string containing the definition identifier for the comment you want to create.

### **inputDataAsXML**

An XML string whose elements are used to create the new comment.

### **creator**

A string containing the username of the person who created the reply in the Forum installation.

### **creationDate**

A Calendar Java object containing the date that the reply was created in the Forum installation.

### **modifier**

A string containing the username of the person who last modified the reply in the Forum installation.

### **modificationDate**

A Calendar Java object that contains the date that the reply was last modified in the Forum installation.

**return\_value**

The identifier of the newly created comment.

**See Also**

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))
- ♦ [migration\\_addReply](#) (page 98)

# migration\_uploadFolderFile

Uploads an entry attachment, allowing preservation of SiteScape Forum data.

## Syntax

```
public void migration_uploadFolderFile( String accessToken, long binderId, long entryId,  
String formDataItemName, String fileName, String modifier, Calendar modificationDate );
```

## Description

The `migration_uploadFolderFile` operation attaches a file to an entry, allowing you to preserve data from the attachment as it last appeared in a SiteScape Forum installation. You can attach only one file at a time; call this operation multiple times to attach more than one file to the entry.

Because moving files across the Internet can be time-consuming, you can create attachments from Forum files that have already been copied to a staging area on the Teaming server by using the `migration_uploadFolderFileStaged` operation.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **binderId**

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

### **entryId**

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

### **formDataItemName**

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 data in the Teaming database; a `hidden` HTML tag communicates this file mapping to the server.

The `name` value for the standard entry element containing attached files is `ss_attachFile1`. If you want to upload a file into a custom form element you defined by using the designers, you need to look up the `name` identifier for that form element.

If you are uploading to a folder file, specify `upload` as an argument to this parameter to make this attachment the primary file for the entry.

### **fileName**

A string containing the name of the file you want to upload.

### **modifier**

A string containing the username of the last person in the Forum installation to modify the file.

**modificationDate**

A Calendar Java object containing the date that the file was last modified in the Forum installation.

**return\_value**

None.

**See Also**

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))
- ♦ [migration\\_uploadFolderFileStaged](#) (page 103)

# migration\_uploadFolderFileStaged

Uploads a local copy of an entry attachment, allowing preservation of SiteScape Forum data.

## Syntax

```
public void migration_uploadFolderFileStaged( String accessToken, long binderId, long entryId,  
String formDataItemName, String fileName, String stagedFileRelativePath, String modifier,  
Calendar modificationDate );
```

## Description

The `migration_uploadFolderFileStaged` operation accesses a file that has been copied locally to the Teaming server as a way to streamline the transfer of files, avoiding transferring them over the Internet. The operation then attaches the file to a folder entry in Teaming.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **binderId**

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

### **entryId**

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

### **formDataItemName**

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 data in the Teaming database; a `hidden` HTML tag communicates this file mapping to the server.

The `name` value for the standard entry element containing attached files is `ss_attachFile1`. If you want to upload a file into a custom form element you defined by using the designers, you need to look up the `name` identifier for that form element.

If you are uploading to a folder file, specify `upload` as an argument to this parameter to make this attachment the primary file for the entry.

### **fileName**

A string containing the name of the file you want to attach to an entry.

**stagedFileRelativePath**

A pathname of the file relative to the staging area on the server side. On the Teaming server, the staging directory is designated by the value of the `staging.upload.files.rootpath` configuration setting. This relative pathname is resolved against the staging directory of the Teaming server to identify the input file.

Although the files can be present in any folder structure within the staging area, one streamlined way to approach this task is to unzip the Forum hidden directory into the staging area. Then, use this parameter to specify the relative path through the hidden folder structure to the location of the file to be attached to the entry in Teaming.

**modifier**

A string containing the username of the person who last modified the file in the Forum installation.

**modificationDate**

A Calendar Java object that contains the date that the file was last modified in the Forum installation.

**return\_value**

None.

**See Also**

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))
- ♦ [migration\\_uploadFolderFile](#) (page 101)



# profile\_addGroup

Adds a group.

## Syntax

```
public long profile_addGroup( String accessToken, Group group );
```

## Description

The `profile_addGroup` operation adds a new group to Teaming.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **group**

A Group Java object containing information needed to create the new group in Teaming.

### **return\_value**

The identifier of the newly created group.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# profile\_addGroupMember

Adds a user to a group.

## Syntax

```
public void profile_addGroupMember( String accessToken, String groupName, String userName );
```

## Description

The `profile_addGroupMember` operation adds a user to a group.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **groupName**

A string containing the name of the group.

### **userName**

A string containing the name of the user to be added to the group.

### **return\_value**

None.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# profile\_addUser

Adds a user profile.

## Syntax

```
public long profile_addUser( String accessToken, User user );
```

## Description

The `profile_addUser` operation adds a profile for a new Teaming user.

After you add a user profile, you can add a user workspace for the new user by using the `profile_addUserWorkspace` operation.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **user**

A User Java object containing the information needed to create a new user.

### **return\_value**

The identifier of the newly created user.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))
- ♦ [profile\\_addUserWorkspace](#) (page 108)

# profile\_addUserWorkspace

Adds a user workspace for an existing user.

## Syntax

```
public long profile_addUserWorkspace( String accessToken, long userId );
```

## Description

The `profile_addUserWorkspace` operation adds a user workspace for an existing user.

To create a new user before using this operation, use the `profile_addUser` operation, which creates a profile for a new user.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **userId**

The identifier of the user for whom you want to create a user workspace.

### **return\_value**

The binder identifier of the newly created user workspace.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))
- ♦ [profile\\_addUser](#) (page 107)

# profile\_deletePrincipal

Removes a group or user.

## Syntax

```
public void profile_deletePrincipal( String accessToken, long principalId,  
boolean deleteWorkspace );
```

## Description

The `profile_deletePrincipal` operation removes a group or user.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **principalId**

The identifier of the group or user you want to delete.

### **deleteWorkspace**

When you delete a user, this Boolean value indicates whether Teaming should delete the corresponding user workspace.

### **return\_value**

None.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# profile\_getFileVersions

Returns information about the versions of a file.

## Syntax

```
public void profile_getFileVersions( String accessToken, long principalId, string fileName );
```

## Description

The `profile_getFileVersions` operation retrieves information about the versions of a file associated with a user or group.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **principalId**

The identifier for the principal (a user or group).

### **fileName**

The filename of the file you want to retrieve version information about.

### **return\_value**

A File Version Java object containing information about the file versions.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,”](#) on [page 17](#))

# profile\_getGroup

Accepts a group identifier to obtain the title and the description of the group.

## Syntax

```
public Group profile_getGroup( String accessToken, long groupId, boolean includeAttachments );
```

## Description

The `profile_getGroup` operation obtains the title and the description of the group.

If you want to get information about the members of a group, use the `profile_getGroupMembers` operation.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **groupId**

The identifier of the group about which you want information.

### **includeAttachments**

A Boolean value that indicates whether you want files that are attached to the group.

By default, you cannot attach files to a group. However, a site administrator can use the designers in the UI to customize a group to be able to include files.

### **return\_value**

A Group Java object containing information about all of the group members.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))
- ♦ [profile\\_getGroupName](#) (page 112)
- ♦ [profile\\_getGroupMembers](#) (page 113)

# profile\_getGroupName

Accepts a group name to obtain the title and the description of the group.

## Syntax

```
public Group profile_getGroupName( String accessToken, String groupName,  
boolean includeAttachments )
```

## Description

The `profile_getGroupName` operation obtains the title and the description of a group.

If you want to get information about the members of a group, use the `profile_getGroupMembers` operation.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **groupName**

A string containing the name of the group.

### **includeAttachments**

A Boolean value that indicates whether you want files attached to the group.

By default, you cannot attach files to a group. However, a site administrator can use the designers in the UI to customize a group to be able to include files.

### **return\_value**

A Group Java object containing information about all of the group members.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))
- ♦ [profile\\_getGroup \(page 111\)](#)
- ♦ [profile\\_getGroupMembers \(page 113\)](#)



# profile\_getGroupMembers

Obtains information about the members of a group.

## Syntax

```
public PrincipalCollection profile_getGroupMembers( String accessToken, String groupName );
```

## Description

The `profile_getGroupMembers` operation obtains information about members of a group.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **groupName**

A string containing the name of the group whose members you want information about.

### **return\_value**

A `PrincipalCollection` Java object containing information about the members of the specified group.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# profile\_getPrincipals

Gets information for users and groups in the installation.

## Syntax

```
public PrincipalCollection profile_getPrincipals( String accessToken, int firstRecord,  
int maxRecords );
```

## Description

The `profile_getPrincipals` operation gets information for users and groups in the installation. Because the set of information is potentially very large, you can use successive calls to this operation to receive manageable subsets of information for each call.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **firstRecord**

The number of the record (information about one user or group) to begin returning. Use this parameter to page the returned list of principals.

The number of the first record 0.

### **maxRecords**

The largest number of records you want returned in this call. For an unlimited number specify -1.

### **return\_value**

A `PrincipalCollection` Java objection containing information about the set of users and groups you requested.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# profile\_getUser

Accepts a user identifier to get information about a user.

## Syntax

```
public User profile_getUser( String accessToken, long userId, boolean includeAttachments );
```

## Description

The `profile_getUser` operation accepts a user identifier and returns information about a Teaming user.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **userId**

The identifier of the user about whom you want information.

### **includeAttachments**

A Boolean value that specifies whether Teaming should return attachments to the user's profile.

By default, the only attached files are the users' pictures. However, the site administrator can customize the profile to include other files by using the designer tools in the UI.

### **return\_value**

A `User` Java object that contains information about the requested user.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))
- ♦ [profile\\_getUserByName](#) (page 116)

# profile\_getUserByName

Accepts a username to get information about a user.

## Syntax

```
public User profile_getUserByName( String accessToken, String userName,  
boolean includeAttachments );
```

## Description

The `profile_getUserByName` operation accepts a username as a parameter and returns information about a Teaming user.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **userName**

A string containing the username of the user for whom you want information.

### **includeAttachments**

A Boolean value that indicates whether Teaming should return attached files.

By default, the only attached files are the users' pictures. However, the site administrator can customize the profile to include other files by using the designer tools in the UI.

### **return\_value**

A User Java object containing information about the requested user.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))
- ♦ [profile\\_getUser](#) (page 115)

# profile\_getUsers

Obtains information for users in the installation.

## Syntax

```
public UserCollection profile_getUsers( String accessToken, boolean captive, int firstRecord, int maxRecords );
```

## Description

The `profile_getUsers` operation gets information for users in the installation. Because the set of information is potentially very large, you can use successive calls to this operation to receive manageable subsets of information for each call.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **captive**

Set this to `true` if you want the permalink URL returned for each user workspace to represent captive mode. When a user workspace is viewed in captive mode, the master heading and the sidebar are removed from the display, which allows the page to fit better in a small screen. The default is `false`.

### **firstRecord**

The number of the record to begin returning. Use this parameter to page the returned list of users.

The number of the first record is 0.

### **maxRecord**

The largest number of records you want to return in this call. Specify -1 for unlimited.

### **return\_value**

A `UserCollection` Java object that contains information about the entries contained within the folder.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))
- ♦ [profile\\_getUserByName](#) (page 116)

# profile\_getUserTeams

Obtains information about all teams that the specified user is a member of.

## Syntax

```
public TeamCollection search_getUserTeams( String accessToken, long userId );
```

## Description

The `search_getUserTeams` operation obtains information about all teams that the user is a member of.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **userId**

The identifier of the user about whom you want information.

### **return\_value**

A `UserCollection` Java object that contains information about the entries contained within the folder.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# profile\_modifyGroup

Modifies a group.

## Syntax

```
public void profile_modifyGroup( String accessToken, Group group );
```

## Description

The `profile_modifyGroup` operation modifies information associated with a group.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **group**

A Group Java object containing modified information about a group.

### **return\_value**

None.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,”](#) on [page 17](#))

# profile\_modifyUser

Modifies a user.

## Syntax

```
public void profile_modifyUser( String accessToken, User user );
```

## Description

The `profile_modifyUser` operation modifies information associated with a user.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **user**

A User Java object containing modified information about a user.

### **return\_value**

None.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))



# profile\_removeFile

Removes a file from the user profile.

## Syntax

```
public void profile_removeFile( String accessToken, long principalId, String fileName );
```

## Description

The `profile_removeFile` operation removes a file from a user profile.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **principalId**

The identifier for the principal (by default, a user) from which you want to remove a file.

By default, only user profiles contain files. However, it is possible for site administrators to customize groups by using the designer tools in the UI.

### **fileName**

A string containing the name of the file you want to remove.

### **return\_value**

None.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# profile\_removeGroupMember

Removes a user from a group.

## Syntax

```
public void profile_removeGroupMember( String accessToken, String groupName,  
String userName );
```

## Description

The `profile_removeGroupMember` operation removes a user from membership in a group.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **groupName**

A string containing the name of the group from which you want to remove a member.

### **userName**

A string containing the name of the user you want to remove from the specified group.

### **return\_value**

None.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# profile\_uploadFile

Uploads a file as an attachment to a user or group.

## Syntax

```
public void profile_uploadFile( String accessToken, long principalID, String formDataItemName,  
String fileName );
```

## Description

The `profile_uploadFile` operation performs an action similar to using the user interface to upload a picture to user profiles. Files are attached one at a time; call this operation multiple times to attach more than one file to the binder.

By default, only user profiles contain files. However, it is possible for site administrators to customize groups by using the designer tools in the user interface,

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **principalId**

The identifier for the user or group to which you want to attach a file.

### **formDataItemName**

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

The name value for the standard entry element containing attached files is `ss_attachFile1`. To upload a file into the custom forms element you defined by using the designer, you need to look up the name identifier for that form element.

To upload a picture for a user profile, specify `picture` as an argument to this parameter to make this attachment one of the pictures associated with the user profile.

### **fileName**

A string containing the filename of the file you want to upload to the principal.

### **return\_value**

None.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# search\_getFolderEntries

Obtains information about the entries that match the specified search query.

## Syntax

```
public String search_getFolderEntries( String accessToken, String query, int offset, int maxResults );
```

## Description

The `search_getFolderEntries` operation obtains information about the entries matching the specified search query. Because the list of each result can be lengthy, this operation lets you make multiple calls, receiving a subset of the search results each time.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **query**

A search query represented in XML.

### **offset**

An integer indicating at which result you want to begin receiving information. The first result is numbered 0.

### **maxResults**

An integer indicating the number of results you want returned. The value of -1 indicates unlimited.

### **return\_value**

A `FolderEntryCollection` Java object containing information about the entries contained within the folder.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# search\_getHotContent

Returns information about binders and entries active during the last two weeks.

## Syntax

```
public String search_getHotContent( String accessToken, String limitType, Long binderId );
```

## Description

The `search_getHotContent` operation returns an XML document about all items that were viewed, downloaded, or modified in the past two weeks.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **limitType**

One of the following values:

- ♦ **activity:** Returns items that had any activity.
- ♦ **download:** Returns items that included at least one attachment that was downloaded.
- ♦ **modify:** Returns items that were modified.
- ♦ **view:** Returns items that were viewed.

### **binderId**

The identifier of the binder whose descendants you want to evaluate for activity. If you want to evaluate all items in a zone, pass the null value to this parameter.

### **return\_value**

An XML string containing information about the entries that had activity.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# search\_getTeams

Obtains information about the teams that the calling user is a member of.

## Syntax

```
public TeamCollection search_getTeams( String accessToken );
```

## Description

The `search_getTeams` operation obtains information about the teams that the calling member is a user of.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **return\_value**

A `TeamCollection` Java object that contains information about the teams that the calling user is a member of.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# search\_getWorkspaceTreeAsXML

Obtains information needed to construct the Teaming workspace and folder tree.

## Syntax

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

## Description

The `search_getWorkspaceTreeAsXML` operation obtains information needed to construct the Teaming workspace and folder tree.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **binderId**

The identifier of the binder whose descendants you want to include in the workspace and folder tree information.

The top workspace in the Teaming tree has a binder identifier 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 that you want all subsequent levels.

### **page**

A parameter used to expand pages of binders. When you specify a valid page identifier, Teaming expands the page by the number of levels indicated in the *levels* parameter.

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

See [Section 1.7.6, “Binder Pages and search\\_getWorkspaceTreeAsXML,” on page 20](#) for more detailed information about working with pages.

### **return\_value**

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

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# search\_search

Returns XML for results of a search query.

## Syntax

```
public String search_search( String accessToken, String query, int offset, int maxResults );
```

## Description

The `search_search` operation returns XML for the results of a search query represented in XML. Because the list of each results can be lengthy, this operation is designed so that you can make multiple calls, receiving a subset of the search results each time.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **query**

A search query represented in XML.

### **offset**

An integer indicating at which result you want to begin receiving information. The first result is numbered 0.

### **maxResults**

An integer indicating the number of results you want returned.

### **return\_value**

A string of XML containing information about the search results that match your specified criteria.

## Example

The following input query string in XML matches all users whose first name begins with the letter J or the last name is Smith.

```
<QUERY>
  <AND>
    <FIELD fieldname="_entityType" exactphrase="true">
      <TERMS>user</TERMS>
    </FIELD>
    <FIELD> fieldname="_docType" exactphrase="true">
      <TERMS>entry</TERMS>
    </FIELD>
  <OR>
    <FIELD fieldname="_lastName">
      <TERMS>Smith</TERMS>
    </FIELD>
  </OR>
</AND>
</QUERY>
```



```
        <FIELD fieldname="_firstName">
            <TERMS>J*</TERMS>
        </FIELD>
    </OR>
</AND>
</QUERY>
```

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# template\_addBinder

Adds a fully configured workspace or folder to the workspace hierarchy.

## Syntax

```
public long template_addBinder( String accessToken, long parentId, long binderConfigId, String title );
```

## Description

The `template_addBinder` operation adds a fully configured workspace or folder to the workspace hierarchy.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **parentId**

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

### **binderConfigId**

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

You can use the `template_getTemplates` information to get a configuration identifier from a binder that has a configuration you want for your new binder. Or, you can get a binder configuration identifier from the Teaming user interface. See [Section 1.7.2, “Adding Folders and the Binder Configuration Identifier,” on page 18](#), for information about getting a configuration identifier from the user interface.

### **title**

A string containing the title of the new binder.

### **return\_value**

The binder identifier of the newly created workspace or folder.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))
- ♦ [template\\_getTemplates \(page 131\)](#)

# template\_getTemplates

Obtains information about all defined templates in the installation.

## Syntax

```
public TemplateCollection template_getTemplates( String accessToken );
```

## Description

The `template_getTemplates` operation obtains information about all defined templates in the installation.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **return\_value**

A `TemplateCollection` Java object that contains information about all templates in the installation.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# zone\_addZone

Adds a zone to the installation.

## Syntax

```
public Long zone_addZone( String accessToken, String zoneName, String virtualHost,  
String mailDomain );
```

## Description

The `zone_addZone` operation adds a zone to the installation.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **zoneName**

A string containing the name of the new zone.

### **virtualHost**

A string specifying the virtual host. (See the installation guide for more information.)

### **mailDomain**

This parameter is not used.

### **return\_value**

The zone identifier, which is the binder identifier of the top workspace in the new zone.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# zone\_deleteZone

Deletes a zone.

## Syntax

```
public void zone_deleteZone( String accessToken, String zoneName );
```

## Description

The `zone_deleteZone` operation deletes a zone.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **zoneName**

A string containing the name of the zone you want to delete.

### **return\_value**

None.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,”](#) on [page 17](#))

# zone\_modifyZone

Modifies a zone.

## Syntax

```
public void zone_modifyZone( String accessToken, String zoneName, String virtualHost,  
String mailDomain );
```

## Description

The `zone_modifyZone` operation changes a zone's virtual host specification.

## Parameters and Return Value

### **accessToken**

Either the security token passed to your application by Teaming as part of implementing a remote application, or the null value.

### **zoneName**

A string containing the name of the new zone.

### **virtualHost**

A string specifying the virtual host. (See the installation guide for more information.)

### **mailDomain**

This parameter is not used

### **return\_value**

None.

## See Also

- ♦ Java objects in the Teaming sources (see [Section 1.7.1, “Working with Java Objects,” on page 17](#))

# Deprecated Web Service Operations

# B

This topic provides alphabetized reference pages for deprecated Web services operations provided by Kablink Teaming.

**NOTE:** Novell recommends that you do not use these Web services operations in new applications. Instead, use the operations documented in [Appendix A, “Web Service Operations,” on page 23](#). Novell continues to support the operations in this appendix for backward compatibility for applications written to interoperate with Teaming 1.03 or earlier.

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 <i>italic</i> font.
Click the <i>Getting Started</i> link.	
<b>Blog summary</b> - 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.
<b>Note:</b> 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 <code>Courier</code> 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.	
<b>[page]</b>	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—V1.0.3)	The versions of Teaming that support the Web services operation (“all versions between Version 1.0 through Version 1.0.3”)

**NOTE:** All examples in this reference section use Apache Axis run-time library methods that specify Web service operations 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.

The `search` operation is under development and subject to change or deletion at any time. Do not use this operation in your client applications.

Web service operations contained in this reference section are used by this Windows based client: `/ssf/samples/wsclient/facade-client.bat`. With the exception of `uploadCalendarEntries`, use the same parameters for the batch-file command that you use for the corresponding Web service message.

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

<b>facade-client.bat command</b>	<b>Web services message</b>
<code>addEntry</code>	<a href="#">addFolderEntry</a>
<code>addFolder</code>	<a href="#">addFolder</a>
<code>addReply</code>	<a href="#">addReply</a>
<code>[none]</code>	<a href="#">addUserWorkspace</a>
<code>indexBinder</code>	<a href="#">indexFolder</a>
<code>listDefinitions</code>	<a href="#">getDefinitionListAsXML</a>
<code>migrateBinder</code>	<a href="#">migrateBinder</a>
<code>migrateEntry</code>	<a href="#">migrateFolderEntry</a>
<code>migrateReply</code>	<a href="#">migrateReply</a>
<code>migrateFile</code>	<a href="#">migrateFolderFile</a>
<code>migrateFileStaged</code>	<a href="#">migrateFolderFileStaged</a>
<code>migrateWorkflow</code>	<a href="#">migrateEntryWorkflow</a>
<code>modifyEntry</code>	<a href="#">modifyFolderEntry</a>
<code>printAllPrincipals</code>	<a href="#">getAllPrincipalsAsXML</a>
<code>printDefinition</code>	<a href="#">getDefinitionAsXML</a>
<code>printDefinitionConfig</code>	<a href="#">getDefinitionConfigAsXML</a>
<code>printFolderEntry</code>	<a href="#">getFolderEntryAsXML</a>
<code>printFolderEntries</code>	<a href="#">getFolderEntryAsXML</a>
<code>printPrincipal</code>	<a href="#">getPrincipalAsXML</a>
<code>printTeamMembers</code>	<a href="#">getTeamMembersAsXML</a>
<code>printTeams</code>	<a href="#">getTeamsAsXML</a>
<code>printWorkspaceTree</code>	<a href="#">getWorkspaceTreeAsXML</a>
<code>setDefinitions</code>	<a href="#">setDefinitions</a>
<code>setFunctionMembership</code>	<a href="#">setFunctionMembership</a>
<code>setFunctionMembershipInherited</code>	<a href="#">setFunctionMembershipInherited</a>
<code>setOwner</code>	<a href="#">setOwner</a>
<code>setTeamMembers</code>	<a href="#">setTeamMembers</a>
<code>synchronize</code>	<a href="#">synchronizeMirroredFolder</a>
<code>uploadCalendar</code>	<a href="#">uploadCalendarEntries</a>
<code>uploadFile</code>	<a href="#">uploadFolderFile</a>



# addFolder

Adds a folder to the workspace-tree hierarchy. (V1—V1.0.3)

## Syntax

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

## Description

The `addFolder` operation 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 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 can be either a workspace or folder.

## See Also

- ♦ The operation table for the Windows based `facade-client.bat` program ([Appendix B, “Deprecated Web Service Operations,” on page 135](#))
- ♦ [Section 1.7.1, “Working with Java Objects,” on page 17](#)

# addFolderEntry

Adds an entry to a folder. (V1—V1.0.3)

## Syntax

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

## Description

The `addFolderEntry` operation 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 to be created (for example, some default entry types are topic, file, blog, wiki, and calendar).

The easiest way to work with definition identifiers for entries is to specify `null` for this value. When you specify `null`, Teaming 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, you want to create an entry in a blog folder. If you pass `null` as the definition identifier, Teaming automatically applies the definition identifier for a blog entry.

As another option, you can use the `getDefinitionConfigAsXML` operation 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.

Use the Teaming UI to create a complete entry of the type you want this Web services operation to create, note the entry identifier, and then use the `getFolderEntryAsXML` operation to return XML for the entry. Then, use the returned XML as a template for this parameter.

### **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 Teaming 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 operation table for the Windows based `facade-client.bat` program ([Appendix B, “Deprecated Web Service Operations,” on page 135](#))
- ♦ [getFolderEntryAsXML](#) (page 148)
- ♦ [getDefinitionConfigAsXML](#) (page 145)

# addReply

Adds a new comment to an entry or comment. (V1.0.3)

## Syntax

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

## Description

The `addReply` operation adds a new comment to an entry or to an existing comment.

## Parameters and Return Value

### **folderId**

The binder identifier of the folder containing the entry or comment to which you want to apply the new comment.

### **parentEntryId**

The entry identifier for the entry or comment to which you want to apply the comment.

### **definitionId**

The 32-character, hexadecimal identifier that maps to the type of comment to be created.

You can use the `getDefinitionListAsXML` operation to get metadata for all definitions. Then, you can parse the XML string for the definition identifier of the type of comment you want.

### **inputDataAsXML**

A string of XML containing the values needed to create a comment of your desired type.

Use the Teaming UI to create a complete comment of the type you want this Web services operation to create, note the entry identifier, and then use the `getFolderEntryAsXML` operation to return XML for the entry. Then, use the returned XML as a template for this parameter.

### **attachedFileName**

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

### **return\_value**

The entry identifier of the newly created comment.

## Example

```
call.setOperationName(new QName("addReply"));
Object result = call.invoke(new Object[] {new Long(21), new Long(45), null, s,
null});
```

This code creates a new comment in the folder whose binder identifier is 21, and applies it to the entry or comment whose entry identifier is 45. The first `null` value instructs Teaming to use the default comment type for the folder. The variable `s` contains XML elements needed by Teaming to create the comment. Because of the final `null` value, the new comment does not include an attached file.

## See Also

- ♦ The operation table for the Windows based `facade-client.bat` program ([Appendix B, “Deprecated Web Service Operations,”](#) on page 135)
- ♦ [getFolderEntryAsXML](#) (page 148)
- ♦ [getDefinitionListAsXML](#) (page 146)

# addUserWorkspace

Adds a new personal workspace. (V1.0.3)

## Syntax

```
public long addUserWorkspace ( long userId );
```

## Description

The `addUserWorkspace` operation adds a new personal workspace to the workspace hierarchy.

The primary purpose of this operation is to assist with migrating data from SiteScape Forum to Teaming. By default using Teaming, the creation of the personal workspace occurs when someone first uses the portal software to sign in with a username and password. If you want to migrate Forum information as sub-content to a personal workspace in Teaming, use this operation before creating the sub-content.

## Parameters and Return Value

### **userId**

The identifier for the user for whom you want to create the personal workspace

### **return\_value**

The binder identifier of the newly created personal workspace.

## Example

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

This code creates a new personal workspace.

## See Also

- ♦ [Section C.1, “Sequence of Migration Operations,” on page 179](#)
- ♦ [Section C.3, “Migrating Users,” on page 180](#)

# getAllPrincipalsAsXML

Returns summary information for users and groups. (V1—V1.0.3)

## Syntax

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

## Description

The `getAllPrincipalsAsXML` operation returns XML elements that provide summary information about registered users and defined groups. You can use this operation to identify a particular user by name or other data, obtain an identifier for a particular user, and then use the `getPrincipalAsXML` operation 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 (for example, 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 operation table for the Windows based `facade-client.bat` program ([Appendix B, “Deprecated Web Service Operations,” on page 135](#))
- [getAllPrincipalsAsXML \(page 143\)](#)

# getDefinitionAsXML

Returns information about one definition. (V1—V1.0.3)

## Syntax

```
public String getDefinitionAsXML( String definitionId );
```

## Description

The `getDefinitionAsXML` operation 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, Teaming returns information about the definition for that entry.

As an alternative, you can use the `getDefinitionConfigAsXML` operation to obtain all definitions in Teaming 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 Teaming.

### **return\_value**

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

## 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 operation table for the Windows based `facade-client.bat` program ([Appendix B, “Deprecated Web Service Operations,” on page 135](#))
- ♦ [addFolderEntry](#) (page 138)
- ♦ [getDefinitionConfigAsXML](#) (page 145)



# getDefinitionConfigAsXML

Returns information about all configuration definitions. (V1—V1.0.3)

## Syntax

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

## Description

The `getDefinitionConfigAsXML` operation 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 operation table for the Windows based `facade-client.bat` program ([Appendix B, “Deprecated Web Service Operations,” on page 135](#))
- ♦ [addFolderEntry \(page 138\)](#)

# getDefinitionListAsXML

Returns metadata for all definitions in the installation. (V1.0.3)

## Syntax

```
public String getDefinitionListAsXML ();
```

## Description

The `getDefinitionListAsXML` operation returns metadata for all definitions in the installation. This metadata includes information such as the definition name and identifier.

When using other Web services operations that require a definition identifier, you can use this message, parse the XML for the name (discussion, blog, calendar, comment), and obtain the 32-character, hexadecimal identifier that maps to the desired object.

## Return Value

### **return\_value**

A string of XML whose elements contain metadata for all definitions in the installation.

## Example

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

This code obtains metadata for all definitions in the installation.

## See Also

- ♦ The operation table for the Windows based `facade-client.bat` program ([Appendix B, “Deprecated Web Service Operations,” on page 135](#))

# getFolderEntriesAsXML

Returns a string containing XML providing summary information about entries in a folder.  
(V1—V1.0.3)

## Syntax

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

## Description

The `getFolderEntriesAsXML` operation 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 operation table for the Windows based `facade-client.bat` program ([Appendix B, “Deprecated Web Service Operations,”](#) on page 135)

# getFolderEntryAsXML

Returns information about one entry in a folder. (V1—V1.0.3)

## Syntax

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

## Description

The `getFolderEntryAsXML` operation 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 Teaming 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, Teaming does not place the entry's file attachments in the client program's source directory.

## See Also

- ♦ The operation table for the Windows based `facade-client.bat` program ([Appendix B, "Deprecated Web Service Operations,"](#) on page 135)
- ♦ [Section 1.7.4, "Fetching Attachments,"](#) on page 20

# getPrincipalAsXML

Returns information about one user or group. (V1—V1.0.3)

## Syntax

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

## Description

The `getPrincipalAsXML` operation 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 operation table for the Windows based `facade-client.bat` program ([Appendix B, “Deprecated Web Service Operations,”](#) on page 135)
- ♦ `getAllPrincipalsAsXML` (page 143)

# getTeamMembersAsXML

Returns information about all team members assigned within a workspace or folder. (V1—V1.0.3)

## Syntax

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

## Description

The `getTeamMembersAsXML` operation 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` operation 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 operation table for the Windows based `facade-client.bat` program ([Appendix B, “Deprecated Web Service Operations,”](#) on page 135)
- ♦ [getTeamsAsXML](#) (page 151)

# getTeamsAsXML

Returns information about all workspaces and folders that have assigned teams. (V1—V1.0.3)

## Syntax

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

## Description

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

## See Also

- ♦ The operation table for the Windows based `facade-client.bat` program ([Appendix B, “Deprecated Web Service Operations,” on page 135](#))
- ♦ `getTeamMembersAsXML` (page 150)

# getWorkspaceTreeAsXML

Returns information needed to construct the Teaming workspace and folder tree. (V1—V1.0.3)

## Syntax

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

## Description

The `getWorkspaceTreeAsXML` operation returns XML elements needed to construct the requested portion of the Teaming 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 Teaming tree has a binder identifier 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 that you want all subsequent levels.

### **page**

A parameter used to expand pages of binders. When you specify a valid page identifier, Teaming expands the page by the levels indicated in the `levels` parameter.

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

The Web-services overview topic contains more detailed information about working with pages ([Section 1.7.6, “Binder Pages and search\\_getWorkspaceTreeAsXML,” on page 20](#)).

### **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 operation table for the Windows based `facade-client.bat` program ([Appendix B, “Deprecated Web Service Operations,” on page 135](#))
- ♦ [Section 1.7.6, “Binder Pages and search\\_getWorkspaceTreeAsXML,” on page 20](#)

# indexFolder

Indexes a folder. (V1.0.3)

## Syntax

```
public void indexFolder( long folderId );
```

## Description

The `indexFolder` operation indexes a folder.

The primary use of this operation is to index data after you migrate it from SiteScape Forum into Teaming. (The migration operations transfer the data but do not index it.)

## Parameter

### **folderId**

The binder identifier of the folder you want to index.

## Example

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

This indexes the folder whose binder identifier is 21.

## See Also

- ♦ The operation table for the Windows based `facade-client.bat` program ([Appendix B, “Deprecated Web Service Operations,”](#) on page 135)
- ♦ [Section C.1, “Sequence of Migration Operations,”](#) on page 179

# migrateBinder

Creates a new workspace or folder while preserving SiteScape Forum data. (V1.0.3)

## Syntax

```
public long migrateBinder ( long parentId, String definitionId, String inputDataAsXML,  
String creator, Calendar creationDate, String modifier, Calendar modificationDate );
```

## Description

The `migrateBinder` operation creates a workspace or folder in Teaming that preserves values from a SiteScape Forum installation (for example, the name of the person who created the item in Forum, the Forum creation date, the person who last modified the item in Forum, and the date of the last modification in Forum).

## Parameters and Return Value

### **parentId**

The binder identifier of the parent of the newly created workspace or folder.

### **definitionId**

The 32-character, hexadecimal identifier that maps to the type of workspace or folder to be created.

You can use the `getDefinitionListAsXML` operation to get metadata for all definitions. Then, you can parse the XML string for the definition identifier of the type of workspace or folder you want to create.

### **inputDataAsXML**

A string of XML supplying the elements and values needed to construct the workspace or folder you want to create.

### **creator**

A string containing the username of the person who created the corresponding workspace or folder in the Forum installation.

### **creationDate**

Calendar data specifying the date when the corresponding workspace or folder was created in Forum.

### **modifier**

A string containing the username of the person who last modified the corresponding workspace or folder in Forum.

### **modificationDate**

Calendar data specifying the date when the corresponding workspace or folder was modified in Forum.

## return\_value

The binder identifier of the newly created workspace or folder.

## Example

```
call.setOperationName(new QName("migrateBinder"));
Object result = call.invoke(new Object[] {new Long(21), def, input, new
String("JSmith"), createcal, new String("JGarces"), modcal});
```

This code creates a new binder determined by the definition in the `def` variable (use the `getDefinitionListAsXML` operation to obtain the correct string for your binder type), and the binder will be a child of the binder whose identifier is 21. The `input` variable contains an XML string, properly formatted for your binder type, which Teaming uses to create binder content. The remaining four parameters provide names (literals) and dates (the `createcal` and `modcal` variables) for the creation and last modification of the corresponding item in the Forum installation.

## See Also

- ♦ The operation table for the Windows based `facade-client.bat` program ([Appendix B, “Deprecated Web Service Operations,” on page 135](#))
- ♦ [Section C.1, “Sequence of Migration Operations,” on page 179](#)
- ♦ [getDefinitionListAsXML \(page 146\)](#)

# migrateEntryWorkflow

Associates an entry with a workflow process while preserving SiteScape Forum data. (V1.0.3)

## Syntax

```
public void migrateEntryWorkflow ( long binderId, long entryId, String definitionId,  
String startState, String modifier, Calendar modificationDate );
```

## Description

The `migrateEntryWorkflow` operation associates a workflow process with an entry in Teaming, while preserving values from a SiteScape Forum installation (for example, the state to which the entry should be set, the person who last changed workflow state in Forum, and the date of the last state change in Forum).

## Parameters and Return Value

### **binderId**

The binder identifier of the folder that contains the entry to which you want to associate a workflow process.

### **entryId**

The entry identifier of the entry to which you want to associate a workflow process.

### **definitionId**

The 32-character, hexadecimal identifier that maps to the workflow-process definition.

Before using this message, you must replicate the Forum workflow processes in Teaming.

### **startState**

The current state of the Teaming entry (which would reflect its last state in Forum).

### **modifier**

A string containing the username of the person who last changed the workflow process in Forum.

### **modificationDate**

Calendar data specifying the date when the workflow process was last changed in Forum.

## Example

```
call.setOperationName(new QName("migrateEntryWorkflow"));  
Object result = call.invoke(new Object[] {new Long(21), new Long(45),  
String("ptoProcess"), String("PTO Request"), new String("JGarces"), modcal});
```

This code associates the `ptoProcess` workflow process with the entry whose identifier is 45 and which is located in a folder whose binder identifier is 21. The entry should be placed in the `PTO Request` state. The operation also provides the name of the person who last changed the workflow state in Forum and the date when that state change occurred.

## See Also

- ♦ The operation table for the Windows based `facade-client.bat` program ([Appendix B, “Deprecated Web Service Operations,” on page 135](#))
- ♦ [Section C.1, “Sequence of Migration Operations,” on page 179](#)
- ♦ [Section C.5, “Migrating Custom Commands and Workflow,” on page 181](#)

# migrateFolderEntry

Creates a new folder entry while preserving SiteScape Forum data. (V1.0.3)

## Syntax

```
public void migrateFolderEntry ( long binderId, String definitionId, String inputDataAsXML,  
String creator, Calendar creationDate, String modifier, Calendar modificationDate );
```

## Description

The `migrateFolderEntry` operation creates a folder entry in Teaming that preserves values from a SiteScape Forum installation (for example, the name of the person who created the item in Forum, the Forum creation date, the person who last modified the item in Forum, and the date of the last modification in Forum).

When creating entries within a file folder in Teaming, use this operation to create the entry, and then use either `migrateFolderFile` or `migrateFolderFileStaged` to attach the file to the entry.

## Parameters and Return Value

### **binderId**

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

### **definitionId**

The 32-character, hexadecimal identifier that maps to the type of entry to be created.

The easiest way to work with definition identifiers for entries is to specify `null` for this value. When you specify `null`, Teaming 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, you want to create an entry in a blog folder. If you pass `null` as the definition identifier, Teaming automatically applies the definition identifier for a blog entry.

As another option, you can use the `getDefinitionListAsXML` operation to get metadata for all definitions. Then, you can parse the XML string for the definition identifier of the type of workspace or folder you want to create.

### **inputDataAsXML**

A string of XML supplying the elements and values needed to construct the type of entry you want to create.

### **creator**

A string containing the username of the person who created the corresponding entry in the Forum installation.

### **creationDate**

Calendar data specifying the date when the corresponding entry was created in Forum.

### **modifier**

A string containing the username of the person who last modified the corresponding entry in Forum.

**modificationDate**

Calendar data specifying the date when the corresponding entry was modified in Forum.

**return\_value**

The entry identifier of the newly created entry.

## Example

```
call.setOperationName(new QName("migrateFolderEntry"));
Object result = call.invoke(new Object[] {new Long(21), def, input, new
String("JSmith"), createcal, new String("JGarces"), modcal});
```

This code creates a new entry of the type determined by the definition in the `def` variable (use the `getDefinitionListAsXML` operation to obtain the correct string for your entry type), and the new entry is to be located in the binder whose identifier is 21. The `input` variable contains an XML string, properly formatted for your entry type, which Teaming uses to create entry content. The remaining four parameters provide names (literals) and dates (the `createcal` and `modcal` variables) for the creation and last modification of the corresponding entry in the Forum installation.

## See Also

- ♦ The operation table for the Windows based `facade-client.bat` program ([Appendix B, “Deprecated Web Service Operations,”](#) on page 135)
- ♦ [Section C.1, “Sequence of Migration Operations,”](#) on page 179
- ♦ [getDefinitionListAsXML](#) (page 146)
- ♦ [migrateFolderEntry](#) (page 159)
- ♦ [migrateFolderFileStaged](#) (page 163)



# migrateFolderFile

Attaches a file to an entry while preserving SiteScape Forum data. (V1.0.3)

## Syntax

```
public void migrateFolderFile ( long binderId, long entryId, String fileUploadDataItemName, String  
filename, String modifier, Calendar modificationDate );
```

## Description

The `migrateFolderFile` operation attaches a file to a folder entry in Teaming that preserves values from a SiteScape Forum installation (for example, the person who last modified the item in Forum, and the date of the last modification in Forum).

## Parameters and Return Value

### **binderId**

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

### **entryId**

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

### **fileUploadDataItemName**

The internal-use name used by the database to identify the file as an element of an entry.

For example, a Forum custom command allowed for uploading different files into a single entry that served different functions, such as an expense report, a meeting presentation, and so on. These custom file uploads have associated internal-use names that are different than the reserved internal-use name applied to standard file entries or standard attachments.

If you are migrating to a folder file, specify `upload` as an argument to this parameter to make this attachment the primary file for the entry.

### **filename**

The name of the file to be attached to the entry.

### **modifier**

A string containing the username of the person who last modified the corresponding file in Forum.

### **modificationDate**

Calendar data specifying the date when the corresponding file was modified in Forum.

## Example

```
call.setOperationName(new QName("migrateFolderFile"));
Object result = call.invoke(new Object[] {new Long(21), new Long(45),
String("_budgetReport"), String("budget-report.xls"), new String("JGarces"),
modcal});
```

This code attaches the `budget-report.xls` file to the entry whose identifier is 45 and is located in a folder whose binder identifier is 21. The internal-use name that maps to the file as an element in the entry is `_budgetReport`. The operation also provides the name of the person who modified the file in Forum and the date when that modification occurred.

## See Also

- ♦ The operation table for the Windows based `facade-client.bat` program ([Appendix B, “Deprecated Web Service Operations,”](#) on page 135)
- ♦ [Section C.1, “Sequence of Migration Operations,”](#) on page 179
- ♦ [Section C.4, “Migrating Files,”](#) on page 180

# migrateFolderFileStaged

Locates a locally stored file, and attaches it to an entry while preserving Forum data. (V1.0.3)

## Syntax

```
public void migrateFolderFileStaged ( long binderId, long entryId,  
String fileUploadDataItemName, String filename, String stagedFileRelativePath, String modifier,  
Calendar modificationDate );
```

## Description

The `migrateFolderFileStaged` accesses a Forum file that has been copied locally on the Teaming server as a way to streamline the transfer of files, avoiding transferring them over the Internet. The operation then attaches the file to a folder entry in Teaming that preserves values from a SiteScape Forum installation (for example, the person who last modified the item in Forum, and the date of the last modification in Forum).

## Parameters and Return Value

### **binderId**

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

### **entryId**

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

### **fileUploadDataItemName**

The internal-use name used by the database to identify the file as an element of an entry.

For example, a Forum custom command allowed for uploading different files into a single entry that served different functions, such as an expense report, a meeting presentation, and so on. These custom file uploads have associated internal-use names that are different than the reserved internal-use name applied to standard file entries or standard attachments.

If you are migrating to a folder file, specify `upload` as an argument to this parameter to make this attachment the primary file for the entry.

### **filename**

The name of the file to be attached to the entry.

### **stagedFileRelativePath**

The relative path specification, beginning with the staging area designated in the `ssf.properties` and `ssf-ext.properties` files on the Teaming server. (See the installation guide for more information about these files.)

Although the files can be present in any folder structure within the staging area, one streamlined way to approach this task is to unzip the Forum hidden directory into the staging area. Then, use this parameter to specify the relative path through the hidden folder structure to the location of the file to be attached to the entry in Teaming.

**modifier**

A string containing the full name of the person who last modified the corresponding file in Forum.

**modificationDate**

Calendar data specifying the date when the corresponding file was modified in Forum.

## Example

```
call.setOperationName(new QName("migrateFolderFileStaged"));
Object result = call.invoke(new Object[] {new Long(21), new Long(45),
String("_budgetReport"), String("budget-report.xls"), String("hidden/ssf/
myworkspace/myforum/4567849"), new String("JGarces"), modcal});
```

To locate the file, Teaming begins with the defined staging folder and then applies the relative path `hidden/ssf/myworkspace/myforum/456789`. This code attaches the `budget-report.xls` file to the entry whose identifier is 45 and is located in a folder whose binder identifier is 21. The internal-use name that maps to the file as an element in the entry is `_budgetReport`. The operation also provides the name of the person who modified the file in Forum and the date when that modification occurred.

## See Also

- ♦ The operation table for the Windows based `facade-client.bat` program ([Appendix B, “Deprecated Web Service Operations,” on page 135](#))
- ♦ [Section C.1, “Sequence of Migration Operations,” on page 179](#)
- ♦ [Section C.4, “Migrating Files,” on page 180](#)

# migrateReply

Creates a new comment while preserving SiteScape Forum data. (V1.0.3)

## Syntax

```
public void migrateReply ( long binderId, long parentId, String definitionId,  
String inputDataAsXML, String creator, Calendar creationDate, String modifier,  
Calendar modificationDate );
```

## Description

The `migrateReply` operation creates a comment in Teaming that preserves values from a SiteScape Forum installation (for example, the name of the person who created the item in Forum, the Forum creation date, the person who last modified the item in Forum, and the date of the last modification in Forum).

## Parameters and Return Value

### **binderId**

The binder identifier of the folder that will contain the new comment.

### **parentId**

The binder identifier of the entry or comment to which you want to apply the new comment.

### **definitionId**

The 32-character, hexadecimal identifier that maps to the type of comment to be created.

You can use the `getDefinitionListAsXML` operation to get metadata for all definitions. Then, you can parse the XML string for the definition identifier of the type of comment you want to create.

### **inputDataAsXML**

A string of XML supplying the elements and values needed to construct the type of comment you want to create.

### **creator**

A string containing the username of the person who created the corresponding reply in the Forum installation.

### **creationDate**

Calendar data specifying the date when the corresponding reply was created in Forum.

### **modifier**

A string containing the username of the person who last modified the corresponding reply in Forum.

### **modificationDate**

Calendar data specifying the date when the corresponding reply was modified in Forum.

## **return\_value**

The entry identifier of the newly created comment.

## **Example**

```
call.setOperationName(new QName("migrateReply"));
Object result = call.invoke(new Object[] {new Long(21), new Long(45), def,
input, new String("JSmith"), createcal, new String("JGarces"), modcal});
```

This code creates a new comment of the type determined by the definition in the `def` variable (use the `getDefinitionListAsXML` operation to obtain the correct string for your comment type). The new comment is to be located in the binder whose identifier is 21, and applied to an entry or comment whose identifier is 45. The `input` variable contains an XML string, properly formatted for your comment type, that Teaming uses to create comment content. The remaining four parameters provide names (literals) and dates (the `createcal` and `modcal` variables) for the creation and last modification of the corresponding reply in the Forum installation.

## **See Also**

- ♦ The operation table for the Windows based `facade-client.bat` program ([Appendix B, “Deprecated Web Service Operations,” on page 135](#))
- ♦ [Section C.1, “Sequence of Migration Operations,” on page 179](#)

# modifyFolderEntry

Modifies a single entry. (V1—V1.0.3)

## Syntax

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

## Description

The `modifyFolderEntry` operation 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 Teaming to modify the contents of the entry.

## See Also

- ♦ The operation table for the Windows based `facade-client.bat` program ([Appendix B, “Deprecated Web Service Operations,” on page 135](#))

# setDefinitions

Establishes workflow-entry associations for a folder. (V1.0.3)

## Syntax

```
public void migrateEntryWorkflow ( long binderId, String[] definitionIds,  
String[] workflowAssociations );
```

## Description

The `setDefinitions` operation uses two arrays to associate workflow identifiers with entry identifiers for a folder. (Teaming associates identifiers in the first element of both arrays, the second element of both arrays, the third, and so on.)

When an entry is associated with a workflow process, creation of an entry of that type automatically places the entry into the initial state of the workflow process.

---

**NOTE:** This operation is an overwrite operation, setting all workflow associations for the folder; you cannot use repeated calls to this operation to set associations incrementally. So, set all of the workflow associations for the folder with one call.

---

## Parameters and Return Value

### **binderId**

The binder identifier of the folder in which you want to associate entry and workflow identifiers.

### **definitionIds**

An array of entry identifiers.

### **workflowAssociations**

An array of workflow identifiers.

Before using this message, you must replicate the Forum workflow processes in Teaming.

## Example

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

This code passes two array variables, `entries` and `workflows`. Teaming uses the corresponding elements in both arrays to create entry-workflow associations for the folder whose binder identifier is 21.

## See Also

- ♦ The operation table for the Windows based `facade-client.bat` program ([Appendix B, “Deprecated Web Service Operations,” on page 135](#))



# setFunctionMembership

Applies access-control settings to a folder or workspace. (V1—V1.0.3)

## Syntax

```
public void setFunctionMembership( long binderId, String inputDataAsXML );
```

## Description

The `setFunctionMembership` operation provides access-control settings for folder or a workspace. The term function is analogous to a role in the user interface (UI).

The primary use of this operation is to establish access-control settings when migrating workspaces and folders from Forum to Teaming. You must ensure that you have migrated Forum user and group names to Teaming that are required for your access-control settings.

---

**NOTE:** This operation is an overwrite operation, setting all function memberships for the folder or workspace; you cannot use repeated calls to this operation to set memberships incrementally. So, set all memberships for the workspace or folder with one call.

---

## Parameters and Return Value

### **binderId**

The binder identifier of the folder or workspace for which you want to set access control.

### **inputDataAsXML**

A string of XML containing the values needed to set access control. Here is an example of XML that sets the visitor function:

```
<workAreaFunctionMemberships>
  <workAreaFunctionMembership>
    <property name="functionName">__role.visitor</property>
    <property name="memberName">jGarces</property>
    <property name="memberName">sChen</property>
    <property name="members">1,2,3</property>
  </workAreaFunctionMembership>
  .
  .
  .
</workAreaFunctionMemberships>
```

To obtain the `functionName` value:

1. Sign in as a site administrator for Teaming.
2. In the administration portlet, click *Configure role definitions*.

3. Click any item (for example, *Participant*).
4. Note or copy the identifier in the *Role Name* text box (for example, `__role.participant`). This identifier begins with a double underscore (`_`).

You can pass either user or group names (for example, `jGarces` or `sChen`) or user or group identifiers (for example, `1`, `2`, `3`). Teaming reserves the identifiers `-1` for the workspace or folder owner, and `-2` for a team member.

## Example

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

This code uses the content of the XML string `s` to establish access-control settings for the folder or workspace whose binder identifier is `21`.

## See Also

- ♦ The operation table for the Windows based `facade-client.bat` program ([Appendix B, “Deprecated Web Service Operations,”](#) on page 135)
- ♦ [Section C.1, “Sequence of Migration Operations,”](#) on page 179

# setFunctionMembershipInherited

Establishes inheritance as the access-control mechanism for a folder or workspace. (V1.0.3)

## Syntax

```
public void setFunctionMembershipInherited( long binderId, boolean inherit );
```

## Description

The `setFunctionMembershipInherited` operation allows you to establish that a folder or workspace is to inherit its access-control settings from the parent binder. The primary purpose of this operation is to set inheritance for folders and workspaces that you migrate from Forum.

## Parameters and Return Value

### **binderId**

The binder identifier of the folder or workspace for which you want to establish inheritance for its access-control settings.

### **inherit**

A boolean value that determines whether the folder or workspace uses inheritance to establish its access settings.

## Example

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

This code establishes inheritance as the access-control mechanism for the folder or workspace whose binder identifier is 21.

## See Also

- ♦ The operation table for the Windows based `facade-client.bat` program ([Appendix B, “Deprecated Web Service Operations,” on page 135](#))
- ♦ [Section C.1, “Sequence of Migration Operations,” on page 179](#)

# setOwner

Establishes the owner of a folder or workspace. (V1.0.3)

## Syntax

```
public void setOwner( long binderId, long userId );
```

## Description

The `setOwner` operation allows you to establish an owner for a folder or workspace. The primary purpose of this operation is to mirror Forum ownership as you migrate folders and workspaces.

## Parameters and Return Value

### **binderId**

The binder identifier of the folder or workspace for which you want to establish ownership.

### **userId**

The user identifier of the person whom you want to be the owner of a folder or workspace.

## Example

```
call.setOperationName(new QName("setOwner"));
Object result = call.invoke(new Object[] {new Long(21), new Long(345)});
```

This code establishes the user whose identifier is 345 as the owner of the folder or workspace whose binder identifier is 21.

## See Also

- ♦ The operation table for the Windows based `facade-client.bat` program ([Appendix B, “Deprecated Web Service Operations,” on page 135](#))
- ♦ [Section C.1, “Sequence of Migration Operations,” on page 179](#)

# setTeamMembers

Establishes the membership of a team for a folder or workspace. (V1.0.3)

## Syntax

```
public void setTeamMembers( long binderId, String[] memberNames );
```

## Description

The `setTeamMembers` operation establishes the members of a team for a folder or workspace.

## Parameters and Return Value

### **binderId**

The binder identifier of the folder or workspace for which you want to establish team membership.

### **memberNames**

An array containing the names of all team members for the folder or workspace.

## Example

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

This code establishes each username in the array `users` as team members for the folder or workspace whose binder identifier is 21.

## See Also

- ♦ The operation table for the Windows based `facade-client.bat` program ([Appendix B, “Deprecated Web Service Operations,” on page 135](#))
- ♦ [Section C.1, “Sequence of Migration Operations,” on page 179](#)

# synchronizeMirroredFolder

Synchronizes the mirrored folder with the folder on the external drive. (V1.0.3)

## Syntax

```
public void synchronizeMirroredFolder( long binderId );
```

## Description

The `synchronizeMirroredFolder` operation synchronizes a mirrored folder with the corresponding file on the external drive. A new mirrored folder does not synchronize with its external drive until a synchronization occurs manually in the user interface (UI) or using this message.

## Parameters and Return Value

### **binderId**

The binder identifier of the mirrored file that you want to synchronize with its external drive.

## Example

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

This code synchronizes with its external drive the mirrored folder whose binder identifier is 21.

## See Also

- ♦ The operation table for the Windows based `facade-client.bat` program ([Appendix B, “Deprecated Web Service Operations,” on page 135](#))
- ♦ [Section C.1, “Sequence of Migration Operations,” on page 179](#)

# uploadCalendarEntries

Creates new calendar entries from a file. (V1—V1.0.3)

## Syntax

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

## Description

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

---

**NOTE:** The `uploadCalendar` command in the `facade-client.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 `facade-client.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. Teaming uses XML-formatted iCal information contained in the `s` variable to create the new calendar entries.

## See Also

- ♦ The operation table for the Windows based `facade-client.bat` program ([Appendix B, “Deprecated Web Service Operations,” on page 135](#))
- ♦ [Section 1.7.5, “Adding Calendar Entries,” on page 20](#)

# uploadFolderFile

Attaches a file to an entry to a folder. (V1—V1.0.3)

## Syntax

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

## Description

The `uploadFolderFile` operation attaches a file to an entry in a folder. You can attach only one file at a time; call this operation 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 Teaming database; a `hidden` HTML tag communicates this mapping to the server.

The `name` value for the standard entry element containing attached files is `ss_attachFile1`. 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_attachFile1"), 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 operation table for the Windows based `facade-client.bat` program ([Appendix B, “Deprecated Web Service Operations,” on page 135](#))
- ♦ [Section 1.7.3, “Attaching Files,” on page 19](#)
- ♦ [getDefinitionConfigAsXML \(page 145\)](#)
- ♦ [getFolderEntryAsXML \(page 148\)](#)



# Migrating from Forum to Kablink Teaming



Kablink Teaming is the ongoing path from the legacy SiteScape Forum product. To assist with migrating data from SiteScape Forum to an installation of Kablink Teaming, Novell developed a set of Web services.

Although this section provides guidance about migrating, the task is complex and requires the active assistance of the Kablink Teaming support team. This is especially true for workflow migration. For more information, please contact the support team and arrange to receive consultation as you perform this task.

- ♦ [Section C.1, “Sequence of Migration Operations,” on page 179](#)
- ♦ [Section C.2, “Migration Overwrite Operations,” on page 180](#)
- ♦ [Section C.3, “Migrating Users,” on page 180](#)
- ♦ [Section C.4, “Migrating Files,” on page 180](#)
- ♦ [Section C.5, “Migrating Custom Commands and Workflow,” on page 181](#)

## C.1 Sequence of Migration Operations

Some operations require the previous execution of other operations. For example, migrating an entry requires that you have already migrated the folder. As another example, a workflow process requires that you have already migrated user and group names, so that these names can be applied to its access control.

Here are notes regarding the sequence of operations:

- ♦ Migrate users and groups, and create personal workspaces early in the process.

Use LDAP to establish the Forum users in Kablink Teaming.

You need the existence of personal workspaces to be able to migrate sub-workspaces and child folders. Also, migrating workflow and some types of custom commands requires that your users be established in Kablink Teaming first.

The Forum term “custom command” maps to “custom view and form” in Kablink Teaming.

- ♦ Generally, migrate parents before children you wish to create.

Examples include migrating parent workspaces before its child folders, and migrating entries before migrating attached files.

Using SiteScape Forum, a “forum” maps to a “folder” in Kablink Teaming, a “reply” maps to a “comment” in Kablink Teaming, and the process of “attaching a file” maps to the Web services phrase “adding a folder file.”

- ♦ Migrate binders before setting their ownership, team members, and access control.

The Forum items “workspaces and folders” map to the Kablink Teaming Web services term of “binders.” The Forum term “access control” maps to “membership.” Also, the Web services term “function” is equivalent to the term “roles” in the UI for Kablink Teaming.

- ♦ Migrate custom commands before creating entries.

The custom command migration process cannot be done using only Web services (see [Section C.5, “Migrating Custom Commands and Workflow,” on page 181](#), for more information).

- ♦ Migrate workflow processes before migrating entries.

First, the workflow-migration process cannot be done using only Web services (see [Section C.5, “Migrating Custom Commands and Workflow,” on page 181](#), for more information). Second, any entry that is currently in a workflow state requires the presence of the workflow definition in Kablink Teaming.

- ♦ After migrating custom commands and workflow, you can migrate workflow associations for specific folders.
- ♦ Finalizing operations include indexing folders and synchronizing any mirrored folders that you created.

Remember that migrated entries do not appear in the UI until you index the folders containing these entries.

The Kablink Teaming UI does not begin to mirror the files on the drive until someone manually synchronizes them. The Web services call is equivalent to a manual synchronization in the UI.

## C.2 Migration Overwrite Operations

Two operations require that you perform the operation for all items using one call to the message; they do not allow you to perform the operation incrementally on subsets of items, using multiple calls to the message. If you call these operations sequentially for subsets of the information, each successive call erases the established data from the previous call.

The operations that require you to perform the operation for all items using only one call are:

- ♦ **binder\_setDefinitions:** Associates entry types with workflow processes (see [setDefinitions \(page 168\)](#)).
- ♦ **binder\_setFunctionMembership:** Sets access control for a workspace or folder (see [setFunctionMembership \(page 169\)](#)).

## C.3 Migrating Users

Migrating users requires two steps:

- 1 Use LDAP to add your Forum users to Kablink Teaming.
- 2 Use the `migration_addBinder` operation to add personal workspaces for the new Kablink Teaming users.

Migrating custom commands and workflow involve additional work in regard to users. See [Section C.5, “Migrating Custom Commands and Workflow,” on page 181](#), for more information.

## C.4 Migrating Files

If you have a small number of files to migrate to Kablink Teaming, you can use the `migration_uploadFolderFile` operation.

However, most Forum installations include a significant number of files, and those files might be large. To improve performance, you should strongly consider using the `migration_uploadFolderFileStaged` message.

Staging involves moving all of the files from SiteScape Forum to the server running the Kablink Teaming installation. Although the files can be located using any folder hierarchy on the server, a convenient way to migrate files is to unzip the Forum hidden directory onto the Kablink Teaming server machine and to work within that existing folder hierarchy from Forum. After placing the files on the Kablink Teaming server, the `migration_uploadFolderFileStaged` operation takes files from the staging area and migrates them into the Kablink Teaming installation.

Here are the steps needed to migrate files:

- 1 Establish a directory on the Kablink Teaming server machine where you want to place the Forum files.
- 2 Make the three required changes to the `ssf.properties` and `ssf-ext.properties` files. This action indicates the location of the staging directory. (See the installation guide for more information about these files.)

Multiple Forum file versions are separate files in the staged area. Call the `migration_uploadFolderFileStaged` operation once for each version of the file, using the same filename for each call but specifying a different path. This method creates versioned files in Kablink Teaming.

- 3 Copy the Forum files onto the Kablink Teaming server, using the specified staging directory as your top directory.
- 4 Use the `migration_uploadFolderFileStaged` operation to migrate the files into the Kablink Teaming installation.

This command attaches files to an existing entry. Also, it accepts as one of its arguments a relative path, which traverses the `s` beneath the designated staging directory.

See [migrateFolderFileStaged \(page 163\)](#), for more information.

## C.5 Migrating Custom Commands and Workflow

Migrating custom commands and workflow require tasks beyond the scope of using only Web services calls. It is highly recommended that you work closely with the Kablink Teaming support team while completing these tasks.

These are the general steps needed to migrate custom commands and workflow processes:

- 1 Migrate your Forum users to Kablink Teaming.
- 2 Use the `profile_getPrincipals` operation to get a list of the user identifiers for the newly created Kablink Teaming users.
- 3 Create a mapping file that maps Kablink Teaming user identifiers to Forum usernames.
- 4 Run a Tcl script—which uses the mapping file—to generate an XML file of workflow information.
- 5 Import the workflow XML file into Kablink Teaming.
- 6 Create another mapping file, which maps workflow identifiers in Kablink Teaming to Forum workflow names.

- 7 Run a Tcl script—which uses the second mapping file—to generate an XML file of custom command information.

Some custom commands are associated with workflow processes. Because of this, the mapping file of workflow information is necessary.

- 8 Import the custom command XML into Kablink Teaming.

---

**NOTE:** This process migrates custom commands created using Forum's UI. It does not migrate template-based custom commands. To migrate template-based custom commands, use the Kablink Teaming entry designer and any necessary JSPs to recreate the command.

---