CollabraSuite, BEA WebLogic Edition™

Integration Guide

Version 3.8
January 2006

Copyright

Restricted Rights Legend

Trademarks and Service Marks

Contents

Overview

An effective collaborative environment not only brings people together, it provides access to important data by tightly integrating with other business critical systems. CollabraSuite provides an Application Programming Interface (API) to facilitate this integration. This API allows developers to seamlessly bring existing data and systems into their collaborative environment, tailoring it to their specific requirements.

For example, documents can be created in a room or user's briefcase using real-time data such as an RSS feed. The document's subscriber list can then be modified to automatically notify users of the new information. Another example might involve dynamically creating rooms or sessions to deal with a situation in real-time, such as an intrusion detection system. When a pre-defined event occurs, the API could be used to create a new collaborative session and bring a set of online users into that new session.

Getting Started

Compiling

In order to begin compiling code using the Integration API, the following CollabraSuite JARs are required: csuite-i9n.jar, csuite-ejb-client.jar and common-server.jar. These JARs are located under the CollabraSuite installation in the lib directory. Additionally, the standard J2EE classes are required. These can usually be found bundled with your J2EE application server. For example, WebLogic includes these classes in weblogic.jar. Below is an example of compiling a client using the Integration API:

```
% javac -classpath csuite-i9n.jar:common-server.jar:csuite-ejb-client.java:
${WL_HOME}/server/lib/weblogic.jar:. CSuiteIntegrationClient.java
```

Running

The Integration API uses Log4j for logging, so running the client code requires all of the JARs mentioned above plus log4j.jar. The following command illustrates running a stand-alone client that connects to a WebLogic application server:

```
% java –classpath csuite-i9n.jar:common-server.jar:
csuite-ejb-client.java:log4j.jar: ${WL_HOME}/server/lib/weblogic.jar:.
CSuiteIntegrationClient
```

Running client code inside the web tier of an application server requires access to the same list of JAR files. These can be made available by placing them in the WEB-INF/lib directory of a WAR. Additionally, the following changes must be made to *web.xml* and the application specific deployment descriptor such as *weblogic.xml*.

Figure 1: Additions to web.xml

```
<ejb-ref>
   <ejb-ref-name>ejb/CSuiteCollaboration</ejb-ref-name>
   <ejb-ref-type>Session</ejb-ref-type>
   <home>
com.collabraspace.csuite.server.i9n.interfaces.CSuiteCollaborationRemoteHome
   </home>
   <remote>
com.collabraspace.csuite.server.i9n.interfaces.CSuiteCollaborationRemote
   </remote>
</ejb-ref>
```

Figure 2: Additions to weblogic.xml

```
<ejb-reference-description>
    <ejb-ref-name>ejb/CSuiteCollaboration</ejb-ref-name>
    <jndi-name>ejb/CSuiteCollaboration</jndi-name>
</ejb-reference-description>
```

Connecting

The Integration API is accessed via Stateless Session Enterprise Java Beans (EJBs). The API can be access both locally and remotely. Local clients run inside the application server while remote clients run stand-alone outside of the application server. The only difference between the two methods is how the code finds and connects to the server using JNDI. When running inside the web tier of an application server, no extra information is required to lookup one of the Stateless Session EJBs:

```
CSuiteAdminRemote csAdmin =
        CSuiteFactory.getCSuiteAdminRemoteInstance();
```

Connecting from a remote client requires more information such as the JNDI initial context factory, provider URL, username and password. The following is an example of connecting remotely using WebLogic.

```
Hashtable h = new Hashtable();
h.put(javax.naming.Context.INITIAL_CONTEXT_FACTORY,
      "weblogic.jndi.WLInitialContextFactory");
h.put(javax.naming.Context.PROVIDER_URL, “t3://localhost:7001”);
h.put(javax.naming.Context.SECURITY_PRINCIPAL, “testuser”);
```

```
h.put(javax.naming.Context.SECURITY_CREDENTIALS, “password”);

CSuiteAdminRemote csAdmin = CSuiteFactory.getCSuiteAdminRemoteInstance(h);
```

Using the API

The API consists of classes defined in two packages:

- com.collabraspace.csuite.server.i9n.util
- com.collabraspace.csuite.server.i9n.ejb

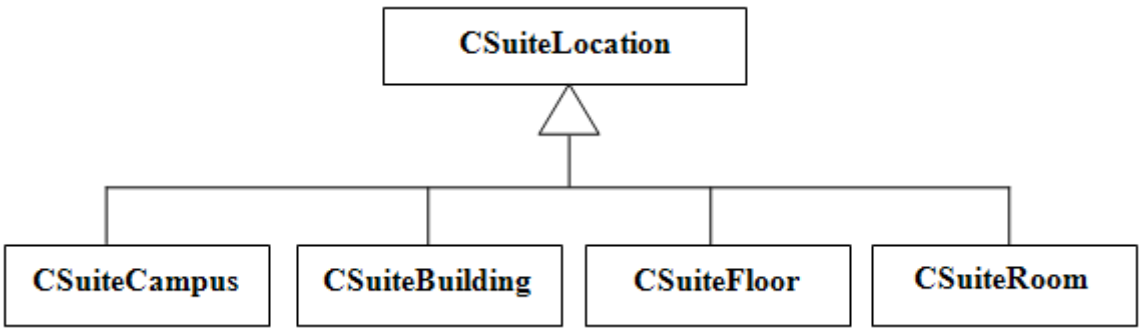
Utility Classes

The *com.collabraspace.csuite.server.i9n.util* package consists of a selection of utility classes and application exceptions, which facilitate the use of the EJBs that define the main API functionality.

The *com.collabraspace.csuite.server.i9n.util* package contains a set of utility classes which are used as arguments and return types of the main EJB methods that form the backbone of the API. These utility classes can be separated into three main types; those which create a CollabraSuite location; those which create a CollabraSuite folder or document; those which create a CollabraSuite group or user. Each utility class accepts a string as a descriptor to construct the related object. There are static methods in each class that can be used to construct these descriptor Strings from the basic building blocks. The user is free to construct the Strings manually, as well. Supplying a descriptor String with an incorrect format (i.e., accidentally using a CSuiteUser descriptor String in a CSuiteRoom constructor) will result in a MalformedDescriptorException (see below).

The classes responsible for creating a CollabraSuite location, as well as, extending the base class CSuiteLocation are represented below:

Figure 3: Location Class Heirarchy



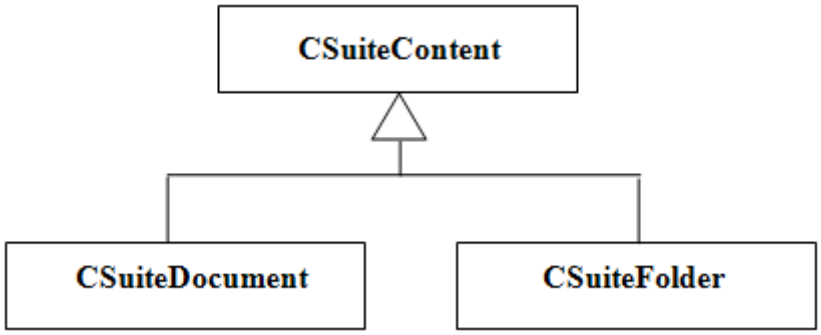
The constructors for the CSuiteLocation classes accept a String argument that describes the fully qualified location within a CSuiteCampus. A descriptor of “SampleCampus/SampleBuilding/SampleFloor/SampleRoom”, for example, defines a room, “SampleRoom”, located within a floor, “SampleFloor”, contained within a building,

"SampleBuilding", all within the campus "SampleCampus". Each level of this descriptor String can be represented by its own CSuiteLocation object, which is a subset of the "SampleRoom" example, above.

In addition to providing descriptor Strings, alternate constructors, defined above, allow one to "extend" an existing CSuiteLocation object. Thus, given a CSuiteCampus object, a CSuiteBuilding within that campus can be created by using the constructor of the form CSuiteBuilding (CSuiteCampus, String), where the String argument is the name of the building. A similar process can be used to create a CSuiteFloor and CSuiteRoom.

The classes responsible for creating a CSuite folder or document are depicted below, and all extend the base class CSuiteContent:

Figure 4: Content Class Heirarchy



The constructors for the CSuiteContent classes take a String argument that describes the file structure within CollabraSuite of the desired content. For example, a document named "MyDocument" within a folder named "Folder1" would be constructed with a descriptor String of "/Folder1/MyDocument". As with CSuiteLocation, nested folders must be created one at a time (although this is not true for deleting content).

It should also be noted that the CSuiteContent classes are used for both File Cabinet and Briefcase operations. The distinction is made by the specific EJB method that is called (i.e., *createBriefcaseDocument ()* vs. *createFileCabinetDocument ()*). The descriptor String "/" always denotes the root of the location (either the File Cabinet or the Briefcase).

The classes responsible for creating a CSuite group or user include:

- CSuiteGroup
- CSuiteUser

The constructors for these objects consist of a campus name and either a user or group name, as appropriate. For example, to create a CSuiteUser object for a user in campus "SampleCampus" whose login is "testUser1", the descriptor String would be: "SampleCampus: testUser1".

Another useful utility class is the CSuitePriority class, which defines a priority within CollabraSuite that can be used as an argument to a page. Unlike the other utilities, this class does not take a descriptor String in its constructor. Rather, it takes the name of the priority (i.e., "High") and a numeric sort order used to compare one priority against another. An optional icon can be supplied, as well.

- CSuitePriority – represents a priority level to a page, and can also be used to add a new priority to CSuiteCampus.

Finally, the API also defines a utility class that performs the JNDI lookups associated with obtaining handles to the two EJBs:

- CSuiteFactory – a utility class that creates EJB interfaces for use by clients. It has methods to get remote instances of the beans that can be used by a client in a web tier.

Exceptions

The *com.collabraspace.csuite.server.i9n.util* package contains two varieties of exceptions that are described in the API. The two varieties are the standard Java/EJB Exceptions and CollabraSuite API specific Integration Application Exceptions. These Integration Application Exceptions are defined in the document as:

- InvalidResourceException – denotes an invalid resource or location in CSuite that is being passed as a descriptor. An example is an incorrect path to a campus location.
- MalformedDescriptor Exception – denotes a syntactical error in the string being passed as a descriptor or supplying a descriptor String with an incorrect format (i.e., accidentally using a CSuiteUser descriptor String in a CSuiteRoom constructor)

It must be noted that there are several other CollabraSuite exception types that may be thrown by the API method. One example of this exception type is the *com.collabraspace.cserver.interfaces.ServiceException*. These exceptions are generated from within the CollabraSuite server code that sits behind the API methods themselves. These are not Integration-specific CollabraSuite exceptions and are not detailed in this API.

Package com.collabraspace.csuite.server.i9n.ejb

The *com.collabraspace.csuit.server.i9n.ejb* package contains two EJBs, one handling basic administrative functions and the other handling collaborative functions. These are described in more detail below.

Administrative Functions

Functions necessary for the administration of a CollabraSuite campus are provided in this API. They can be grouped into three categories: information retrieval/modification, location creation/deletion, and permission modification.

Information Retrieval and Modification

The information retrieval functions provide users with information relating to the structure of the CSuite campus, such as the definition of the buildings, floors, and rooms contained within the campus. They also provide information on the users in the campus; such as, who are the active users in a campus? Where are they located? Which are currently on-line? What are their skills? These methods typically return *java.util.Collections* containing CSuite utility types described in the above Utility Classes section. For example, ***getUsers*** returns a Collection of CSuiteUser objects, a utility class.

The information modification functions allow for the creation and deletion of skills which are assigned to users and campuses. The list of available skills is customizable and allow for greater knowledge sharing and problem solving because users can seek out other users with a required skill set in order to tackle an issue or problem. The functions, ***createSkill*** and ***deleteSkill***, accept the CSuiteCampus utility and a String, which is the skill to be created or deleted, and complete the action within the campus specified. The information modification functions also allow for the creation of priorities within a campus. These are used to prioritize pages and secure chat sessions. Default priorities are “Low”, “Medium” and “High”, but other priorities can be added to a campus. Using the function, ***createPriority,*** a CSuiteCampus class and the CSuitePriority class are used to assign a new, non-default priority to a CollabraSuite campus.

Specifically, these information retrieval/modification functions are:

- getAssociates
- getBuildings
- getCampuses
- getFloors
- getGroups
- getOnlineUsers
- getPriorities
- getRooms
- getSkills
- getSkillsForUser
- getUserLocations
- getUsers
- createPriority
- createSkill
- deleteSkill

Location Creation/Deletion

The location functions deal solely with the creation and deletion of locations within a campus. A location can be a room, a floor, a building, and even a campus. Also available is the verification

of the existence of a location within a campus or of the campus, itself. This verification ensures that duplicate locations cannot be created or that a location can be identified before it is deleted or modified. The location functions are:

- createLocation
- deleteLocation
- doesLocationExist

It is important to understand that when creating nested locations, it is necessary to create the locations in proper order. That is, one cannot create a CSuiteRoom object without first creating the CSuiteFloor object, or the CSuiteFloor before the CSuiteBuilding, etc. For example, in order to create the "SampleRoom" (described above), it would be necessary to create first the "SampleCampus", and then the "SampleBuilding" location, followed by the "SampleFloor", and lastly the "SampleRoom" location. Any attempt to create this location in a single call, that is, creating the campus, building, floor and room all at the same time, will result in an *InvalidResourceException*, defined in the Exceptions section, above. In the prior example, it is not necessary to create descriptor Strings for each level. As indicated above in the Utility Classes section, and will be shown in the Samples section, which follows, CSuiteLocation objects can be used in the constructors for other CSuiteLocations, easing the developer's task in creating nested locations within a campus.

Permissions

Functions to modify permissions are also available in this API. These permissions take on a variety of forms. They can be access or administration privileges given to locations for specified groups and users. They can even be associates lists which are assigned to specific users. Regardless of form, these all, in some way, regulate or restrict access to locations and users within in the campus:

- setAssociates
- setLocationAccess
- setLocationAdministrators
- setSkillsForUser

In many of these cases, the arguments for the methods include arrays of CSuiteUser objects and arrays of CSuiteGroup objects. The general rule of thumb is that any user or group specified will be used in the given operation. If the users array is null, only the specified groups will be used. Conversely, if the groups array is null, only the specified users will be used. When both the users and groups arrays are null, the operation will be applied to all users in the campus (i.e., the "Everyone" group).

Also, in some instances, such as *setLocationAccess()*, a grantMode is required. This defines whether we intend to grant or deny access to the supplied users/groups. The constants used for the grantMode argument are defined in the ACLConstants class, contained within the *csuite-ejb-client.jar* file.

Collaboration Functions

Functions required to establish and maintain a collaborative session are also included in this API. These functions relate to the creation, management, and deletion of documents, the sending of pages between users and/or groups and the initiation of sidebar sessions. These functions are provided via utility classes found in the *com.collabraspace.csuite.server.i9n.ejb* package. In a collaborative session, the creation of documents and folders, and the ability to manage and share them in that session, is essential. The following allow for the creation, deletion and examining of the contents of files and folders during collaboration:

- createBriefcaseDocument
- createBriefcaseFolder
- createFileCabinetDocument
- createFileCabinetFolder
- deleteBriefcaseItem
- deleteFileCabinetItem
- getBriefcaseDocumentContents
- getBriefcaseItems
- getFileCabinetDocumentContents
- getFileCabinetItems
- setFileCabinetItemPermissions
- setFileCabinetItemSubscribers

The *setFileCabinetItemPermissions()* method defines both a grantMode, explained above in the Administrative Functions section, as well as an accessType. This access type defines whether permissions for an item are being set to read, write or read-write. The constants used for these arguments are also defined in ACLConstants.

Sending pages and participating in sidebar sessions are also major aspects of collaboration and the utilities, *sendPage()* and *createSidebar()*, allow for this functionality in a CSuite campus.

Logging

The Integration API uses Log4j as its logging implementation. For details on configuring Log4j, see http://logging.apache.org/log4j/docs/manual.html. When running inside the WebLogic application server (or when weblogic.jar is on the CLASSPATH), log messages are integrated with the WebLogic log. In both cases, the “cs.log.debug” Java system property can be used as a convenience to enable debugging on a package or class basis. The following example starts a client with debugging turned on:

```
% java –classpath csuite-i9n.jar:common-server.jar:csuite-ejb-client.java:
log4j.jar:${WL_HOME}/server/lib/weblogic.jar:.
-Dcs.log.debug=com.collabraspace CSuiteIntegrationClient
```

Samples

Sample code is provided here to demonstrate typical uses of the Integration API.

Creating a document

```
CSuiteCollaborationRemote collaboration =
        CSuiteFactory.getCSuiteCollaborationRemoteInstance();

// Owner of the document
String campusName = "SampleCampus";
String userName = "testUser";
String userDesc = CSuiteUser.buildUserDescriptor(campusName, userName);
CSuiteUser csUser = new CSuiteUser(userDesc);

// Document’s location inside of CollabraSuite
CSuiteDocument document = new CSuiteDocument("/");

// Path to the existing file to be imported into CollabraSuite
File file = new File("exampleFile.txt");

// Create the file in the user’s briefcase
collaboration.createBriefcaseDocument(csUser, document, "File description",
                                      file);
```

Sending a Page to a User

```
CSuiteCollaborationRemote collaboration =
        CSuiteFactory.getCSuiteCollaborationRemoteInstance();

String campusName = "SampleCampus";
String userName = "testUser";
String userDesc = CSuiteUser.buildUserDescriptor(campusName, userName);
CSuiteUser csUser = new CSuiteUser(userDesc);

CSuiteUser[] toUsers = new CSuiteUser[] { csUser };

String campusDesc = CSuiteCampus.buildCampusDescriptor(campusName);
CSuiteCampus csCampus = new CSuiteCampus(campusDesc);

// Send the page
collaboration.sendPage(csCampus, null, toUsers, "subject", "Page text",
                       PageConstants.NO_RESPONSE_REQUIRED_MODE);
```

Retrieving Online Users

```
CSuiteAdminRemote admin = CSuiteFactory.getCSuiteAdminRemoteInstance();

String campusName = "SampleCampus";
String campusDesc = CSuiteCampus.buildCampusDescriptor(campusName);
CSuiteCampus csCampus = new CSuiteCampus(campusDesc);
```

```
// Get the online users and print them out
Collection onlineUsers = admin.getOnlineUsers(csCampus);
for (Iterator i = onlineUsers.iterator(); i.hasNext(); ) {
    CSuiteUser user = (CSuiteUser) i.next();
        System.out.println(user);
}
```

Retrieving Rooms

```
CSuiteAdminRemote admin = CSuiteFactory.getCSuiteAdminRemoteInstance();

String campusName = "SampleCampus";
String campusDesc = CSuiteCampus.buildCampusDescriptor(campusName);
CSuiteCampus csCampus = new CSuiteCampus(campusDesc);

// Iterate over all Buildings, floors and rooms
Collection buildings = admin.getBuildings(csCampus);
for (Iterator i = buildings.iterator(); i.hasNext(); ) {
    Collection floors = admin.getFloors((CSuiteBuilding) i.next());
    for (Iterator j = floors.iterator(); j.hasNext(); ) {
        Collection rooms = admin.getRooms((CSuiteFloor) j.next());
        for (Iterator k = rooms.iterator(); k.hasNext(); ){
            CSuiteRoom room = (CSuiteRoom) k.next();
            System.out.println(room);
        }
    }
}
```

Retrieving/Printing a User's Skills

```
CSuiteAdminRemote admin = CSuiteFactory.getCSuiteAdminRemoteInstance();

String campusName = "SampleCampus";
String campusDesc = CSuiteCampus.buildCampusDescriptor(campusName);
CSuiteCampus csCampus = new CSuiteCampus(campusDesc);

String userName = "testUser";
String userDesc = CSuiteUser.buildUserDescriptor(campusName, userName);
CSuiteUser csUser = new CSuiteUser(userDesc);

// Get the user's skills and print them out
Collection userSkills = admin.getSkillsForUser(csUser);
for (Iterator i = userSkills.iterator(); i.hasNext(); ) {
    String skill = (String) i.next();
    System.out.println(skill);
}
```