



CollabraSuite®

Developer Guide

Version 7.0

Copyright

Copyright © 2000-2008 CollabraSpace, Inc. All Rights Reserved.

Restricted Rights Legend

This software is protected by copyright, and may be protected by patent laws. No copying or other use of this software is permitted unless you have entered into a license agreement with CollabraSpace authorizing such use. This document is protected by copyright and may not be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form, in whole or in part, without prior consent, in writing, from CollabraSpace, Inc.

Information in this document is subject to change without notice and does not represent a commitment on the part of CollabraSpace. THE DOCUMENTATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. FURTHER, COLLABRASPACE DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE DOCUMENT IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

Contents

Overview

Single Sign On	1-1
Two way SSL Authentication	1-2
Single Application Server Deployment	1-2
Multiple Application Server Deployment	1-2

Web Applications

Using Hyperlinks	2-1
Using IFrames	2-2
Using the JavaScript API	2-3

WebLogic Portal

Create the Portal	3-1
Extend the Portal	3-1
WebLogic 10 Portal	3-1
WSRP	3-3

WebCenter Interaction

WebCenter Interaction Portal	4-1
------------------------------------	-----

MediaWiki Integration

Prerequisites	5-1
Installation	5-1
Troubleshooting	5-2

Eclipse Plugin

Prerequisites	6-1
Installation	6-1
Running the Plugin	6-2

Integration API

Compiling	7-1
Running	7-2
Connecting	7-4
API Overview	7-5
Utility Classes	7-5
Administrative Functions	7-8
Information Retrieval and Modification	7-8
Location Administration	7-10
Permissions	7-11
Collaboration Functions	7-11
Exceptions	7-13
Logging	7-13
Transactions	7-13
Sample Code	7-14
Creating a document	7-15
Sending a Page to a User	7-16
Retrieving Online Users	7-16
Retrieving Rooms	7-17
Retrieving/Printing a User's Skills	7-18

Appendices

Changing the Authentication Method	8-1
--	-----

Changing the Session Tracking Cookie Name	8-2
CollabraSuite HTML Page	8-2
User Creation Parameters	8-2
Login Parameters	8-3
Components	8-4
Component Parameters	8-5

Overview

An effective collaborative environment not only brings people together, it provides seamless access to important data by tightly integrating with other business critical systems. CollabraSuite components can be integrated into existing applications using a variety of techniques. CollabraSuite can be integrated into web applications using hyperlinks, iframes or the CollabraSuite JSP tag library. CollabraSuite ships with JSR-168 compliant portlets for integrating into portal applications such as WebLogic Portal. CollabraSuite portlets support Web Services for Remote Portlets (WSRP) in order to support federated portal environments. CollabraSuite also ships with remote portlets for use with WebCenter Interaction. CollabraSuite can be integrated into Wiki environments such as Media Wiki with the CollabraSuite Media Wiki Extension. Applications deployed using the Eclipse Rich Client Platform (RCP) can be enhanced with CollabraSuite functionality with the CollabraSuite Eclipse Plugin. Finally, CollabraSuite can be integrated with other enterprise services via the CollabraSuite Integration API.

Single Sign On

Regardless of the strategy used to integrate CollabraSuite with an existing web application, it is usually desirable to achieve single sign on (SSO) between CollabraSuite and the web application so end users are not required to login twice. There are several architecture decisions that affect how single sign on should be implemented. For example, whether both applications deployed into the same or different application servers; what authentication mechanism is being used; or if perimeter authentication has been implemented by a third party SSO provider.

Two way SSL Authentication

When using SSL with two way authentication, the client browser is asked to present the user's certificate in order to authenticate the user. In this case, the browser can automatically present the certificate to both applications, relieving the user of having login separately to each application. In this case, CollabraSuite must be modified to use the `CLIENT-CERT` authentication method as described in the **Changing the Authentication Method** appendix.

Single Application Server Deployment

When CollabraSuite and the web application are deployed into the same application server or into a homogeneous cluster of application servers using BASIC or FORM based authentication, both applications must use the same session tracking cookie name. CollabraSuite uses the default name of JSESSIONID, and if the web application uses this same default value, nothing else has to be configured. Once the user has been authenticated with one of the web applications, they will automatically be considered authenticated with the other application. Refer to the **Changing the Session Tracking Cookie Name** appendix for details on changing the cookie name used by CollabraSuite.

Multiple Application Server Deployment

When CollabraSuite and the web application are deployed into different application servers or into a heterogeneous cluster of application servers, then perimeter authentication must be provided by a third party SSO provider such as CA SiteMinder or RSA AccessManager. To support this configuration, the session tracking cookie name must be changed so that CollabraSuite and the web application use *different* cookie names. Because there are two application servers involved, they will each use their own HTTP cookie for session tracking. If the cookies have the same name, they will interfere with each other. Refer to the **Changing the Session Tracking Cookie Name** appendix for details on changing the cookie name used by CollabraSuite. In order to support perimeter authentication, CollabraSuite must also be modified to use the `CLIENT-CERT` authentication method as described in the **Changing the Authentication Method** appendix.

Web Applications

CollabraSuite can be integrated into existing web applications in a variety of ways. An application can simply link to one of the supplied HTML pages using hyperlinks. Another strategy is to use iframes that point to either a custom page or to one of the pages delivered with the product. Yet another strategy is to use the CollabraSuite JSP tag library to embed collaboration components directly in a web page.

Note: Regardless of the technique used to integrate CollabraSuite, an end user may only have one page containing CollabraSuite components open at a time. This does not include any popup pages opened through CollabraSuite itself.

Using Hyperlinks

The easiest form of integration is simply linking to the predefined HTML page. For example:

```
<a href="http://host:port/csuite/ui/client/component.jsp?
campus=SampleCampus&component=Workplace&toolbar=true"
target="CSWorkplace">
CollabraSuite Workplace</a>
```

The available components are listed in the **Components** appendix. Any additional parameters listed in the **Login Parameters** and **User Creation Parameters** appendices may also be specified.

Advantages

- Ease of implementation.

- Only minor changes required to an existing application.

Limitations

- This is a very loose form of integration.
- Users are not in the collaborative environment unless they explicitly click on the hyperlink to open the page.
- Collaborative components appear in a separate page instead of appearing directly on the page along with other relevant information.

Using IFrames

This is still a simple form of integration, and with the proper styling, the iframe is nearly undetectable to the user. To use an iframe, place it in the web page as desired and point the iframe's source attribute to the predefined CollabraSuite HTML page. For example:

```
<iframe src="http://host:port/csuite/ui/client/component.jsp?
campus=SampleCampus&component=Workplace&toolbar=true"
style="width:100%; height:400px; border:0px;" frameborder="0">
  <p>Your browser does not support iframes.</p>
</iframe>
```

The available components are listed in the **Components** appendix. Any additional parameters listed in the **Login Parameters** and **User Creation Parameters** appendices may also be specified.

Advantages

- Ease of implementation.
- Only minor changes required to an existing application.
- The CollabraSuite content appears in-line with other relevant information.

Limitations

- Not everyone wants to use iframes.
- CollabraSuite content such as dialog boxes and context menus are limited to the boundary of the iframe.

Using the JavaScript API

CollabraSuite provides a JavaScript API that allows developers to easily embed CollabraSuite components directly into their web pages. To use the API, developers must first include the following line in their web pages:

```
<script type="text/javascript"
  src="http://host:port/csuite/ui/boot.jsp"></script>
```

Developers can then use the CollabraSuite JavaScript API to login, logout and place UI components on the page.

Logging In

The login method logs the user into the CollabraSuite environment so they appear online to other users without rendering any CollabraSuite UI components. After the login method executes; users are available to receive collaborative events such as Sidebar sessions. This method is only necessary if no other UI components are to be displayed on the page.

```
csuite.login({
  campus: 'SampleCampus'      // The campus name (required)
});
```

For a complete list of parameters to the login method, refer to the **Component Login Parameters** appendix.

Logging Out

The logout method logs the user out of the CollabraSuite environment so they appear offline to other users. After the logout method executes; users no longer receive collaborative events.

```
csuite.logout();
```

Creating UI Components

Developers can place multiple CollabraSuite UI components on a page using the create method. HTML DIV elements and CSS can be used to arrange the components on the page together with existing content.

```
<script rel="CollabraSuite">
  csuite.create({
    xtype: csuite.types.Workplace
    , campus: 'SampleCampus'
    , height: 400
  });
</script>
```

Where `xtype` is one of the components listed in the **Components** appendix with a prefix of `csuite.types`. For a full list of the create method parameters refer to the **Login Parameters** and **Component Parameters** appendices.

Advantages

- The CollabraSuite content is embedded in-line with other relevant information
- Multiple CollabraSuite components can be used on the same page.

Limitations

- More extensive changes are required to an existing web application.

WebLogic Portal

CollabraSuite provides JSR-168 complaint portlets that can be used to integrate CollabraSuite components into existing WebLogic portal applications.

Create the Portal

For details on creating a WebLogic Portal application, consult Oracle's website on [Getting Started with Portal Development](http://edocs.bea.com/wlp/docs81/startdev/index.html).
(<http://edocs.bea.com/wlp/docs81/startdev/index.html>)

Extend the Portal

The following sections describe integrating CollabraSuite into a WebLogic Portal environment.

WebLogic 10 Portal

This section describes CollabraSuite integration with WebLogic Portal 10.

1. Start WebLogic Workshop.
2. Open an existing portal EAR project or create a new one via **File->New->Portal EAR Project...**
3. Select an existing portal WAR project or create a new one via **File->New->Portal Web Project...**

- a. Select the following project facets:
 - **CollabraSuite > CollabraSuite Portlets**
 - **CollabraSuite > CollabraSuite Taglibs**
- b. Select the **Use Shared J2EE Libraries** checkbox.
4. Create a new Portal by right clicking on the portal web project and selecting **New->Portal**
5. CollabraSuite portlets are available in the **Design Palette** view. Simply select a portal page and drag them onto the page in the desired location.
6. CollabraSuite provides a JSP tag library for creating custom CollabraSuite portlets. The CollabraSuite tags are available under the **JSP Design Palette** in WebLogic Workshop.
7. Edit the `WEB-INF/portlet.xml` file as necessary to specify default portlet preferences.
 - a. Go to the **Merged Projects View (Window->Show View->Merged Projects)**
 - b. Select the `WEB-INF/portlet.xml` file.
 - c. Right click, select **Copy To Project**, and then open the file.
 - d. Set the `campus` portlet preference for all of the portlets to the correct campus name. This can be accomplished by performing a global search and replace on the default 'SampleCampus' value.
 - e. Optionally, add a `createUser` portlet preference to automatically create a CollabraSuite user if they don't already exist.

Note: This requires **Auto Account Creation** to be enabled on the campus. See the **CollabraSuite Administration Guide** for details.
8. Edit the portlet preferences from the Portal Administration Console
 - a. Open the WebLogic Portal Administration Console in a browser. The URL will be `http://host:port/PortalEarProjectNameAdmin`
 - b. Go to **Portal Management**.
 - c. Go to **Portal Resources->Library->Portlets**.

- d. Select a CollabraSuite portlet, click on **Portlet Preferences** and edit the desired preferences.
- e. Click the **Update WebApp** button, select the portal web application and click **Save**.

WSRP

CollabraSuite supports federated WebLogic Portals through the use of Web Services for Remote Portlets (WSRP). In a federated portal scenario, the producer portal contains CollabraSuite components and serves the content to consumer portals as a web service. The producer portal is configured with CollabraSuite according to the instructions in the previous sections. The consumer portal is set up with the following steps:

1. Create a domain that supports WebLogic Portal.
2. Open WebLogic Workshop and create a Portal EAR project.
3. In WebLogic Workshop, create a Portal Web project and add this project to the Portal EAR project.
4. Edit `WEB-INF/weblogic.xml` to change the `JSESSIONID` cookie to something other than `JSESSIONID`. Alternatively, the `JSESSIONID` can be changed in the in CollabraSuite application and in the producer portal web project. Add the following to `weblogic.xml` in order to change the cookie name:

```
<wls:session-descriptor>
    <wls:cookie-name>WLPJSESSIONID</wls:cookie-name>
    <wls:cookie-path>/</wls:cookie-path>
</wls:session-descriptor>
```

5. Create the consumer portal in the Portal Web project.
6. Add a login portlet. CollabraSuite ships with a sample login portlet that can be used.
7. Use WebLogic Workshop to create remote portlets for the desired CollabraSuite portlets.
8. Add the remote portlets to the consumer portal.
9. Deploy the Portal EAR project to the consumer domain

Note: The system clocks for the producer and consumer machines must be synchronized. The SAML authentication between the producer and the consumer WebLogic instances will fail if the clocks are sufficiently out of sync. It is recommended that NTP or a similar mechanism be used to keep the system clocks accurate. Alternatively, the **Default Time To Live Offset** option can be set in the **SAMLCredentialMapper** in the consumer domain's security realm to compensate for time differences.

WebCenter Interaction

For details on creating an WebCenter Interaction Portal, consult [Oracle's WebCenter Interaction Documentation](http://download.oracle.com/docs/cd/E13158_01/alui/wci/docs103/index.html)

(http://download.oracle.com/docs/cd/E13158_01/alui/wci/docs103/index.html)

WebCenter Interaction Portal

This section describes CollabraSuite integration with a WebCenter Interaction (formerly AquaLogic User Interaction) portal.

In order to achieve single sign on with the WebCenter Interaction Portal, the WebLogic instance hosting CollabraSuite must be configured with additional security providers.

1. If necessary, copy the `WCI/csWCISecurityProviders.jar` file from the CollabraSuite install directory into the `WL_HOME/server/lib/mbeantypes` directory. This jar is placed in the correct directory when CollabraSuite is installed.

Note: When deploying to a domain that contains servers on multiple machines, the `csWCISecurityProviders.jar` file must be manually copied into the `WL_HOME/server/lib/mbeantypes` directory on each machine. Failure to do this will result in a validation error when applying changes to the WebLogic console.

2. In the WebLogic Console, navigate to the active security realm (**myrealm** by default).
3. From the **Providers** tab, click **New**. Enter **WCIdentityAssertionAuthenticator** for both the **Name** and the **Type** fields and click **Ok**.

4. From the **Providers** tab, click **New**. Enter **WCIdentityAsserter** for both the **Name** and the **Type** fields and click **Ok**.
5. From the **Providers** tab, select the **WCIdentityAssertionAuthenticator** and go to the **Provider Specific** tab. Fill in the **Allowed IP Addresses** field with a comma separated list of IP addresses that will be hosting the WebCenter Interaction Portal. This limits logins from only those machines. This field may be left blank to allow any machine to successfully connect.

Note: The Allowed IP Addresses field should not be left empty in a production environment.

6. From the **Providers** tab, select the **WCIdentityAsserter** and go to the **Provider Specific** tab. Fill in the **Allowed IP Addresses** field with a comma separated list of IP addresses that will be hosting the WebCenter Interaction Portal. This limits logins from only those machines. This field may be left blank to allow any machine to successfully connect.

Note: The Allowed IP Addresses field should not be left empty in a production environment.

7. From the **Providers** tab, select any other Authenticators that are configured in the security realm and set the **Control Flag** to **SUFFICIENT**.
8. Restart the WebLogic server.

When using WebLogic 10 and later, both the **CollabraSuite** and the **CollabraSuite WebCenter Interaction** templates can be applied at the same time, either during domain creation, or later by extending the domain.

1. Update the CollabraSuiteWCIPortlets deployment descriptors.
 - a. In order to automatically create CollabraSuite users corresponding to WebCenter Interaction users, the name of a CollabraSuite administrative user is contained in the `weblogic.xml` file of the `wciportlets` web application. The default user name is `csadmin`. If the `csadmin` user doesn't exist in the domain or it is not in the `CollabraSuiteAdministrators` group, the `weblogic.xml` must be modified. Change the following XML in
`<applications_dir>/CollabraSuiteWCIPortlets.ear/wciportlets.war/WEB-INF/weblogic.xml` to reflect the name of a CollabraSuite administrative user in your domain:

```

<security-role-assignment>
  <role-name>SSO_ROLE</role-name>
  <principal-name>csadmin</principal-name>
</security-role-assignment>

<run-as-role-assignment>
  <role-name>SSO_ROLE</role-name>
  <run-as-principal-name>csadmin</run-as-principal-name>
</run-as-role-assignment>

```

- b. By default, CollabraSuite will create new users with the same user name that is used to log into WebCenter Interaction, with a default homeroom of SampleCampus/SampleBuilding/SampleFloor/SampleRoom. If you would like to change any of these defaults, you need to modify the Servlet parameters in the CollabraSuiteWCIPortlets EAR. These are found in the `web.xml` file of the `wciportlets` web application.

Note: Remember to redeploy this EAR if you make any modifications to the deployment descriptors.

2. Import the CollabraSuite portlets

- a. If necessary, start up the server that is running the WebCenter Interaction application.
- b. Navigate to `http://host:port/portal/server.pt`
- c. Log in as the Administrator. The default is 'Administrator' with no password.
- d. Click on the **Administration** link.
- e. Under the **Select Utility...** drop down, choose **Migration - Import**.
- f. Click **Browse...** under File Path, and select the following file:
`WL_HOME/collabrasuite-700/WCI/QuickStart/
CollabraSuiteALUIComponents.ptc`
- g. Click **Load Package**.
- h. Click **Finish** in the upper-right-hand corner of the window, and then click **OK** to confirm the migration.

- i. You now have a WebCenter Interaction Experience Definition that contains all of the CollabraSuite portlets, as well as a default page to that contains CollabraSuite portlets.
3. Modify Remote Server Properties. The migration in the previous step created a remote server object, which should point to the CollabraSuiteWCIPortlets application that was deployed by the extension template. To change the default of `localhost` as the server host and `7001` as the server port:
 - a. Make sure that the server running the WebCenter Interaction application is running.
 - b. Log into the WebCenter Interaction application as the Administrator.
 - c. Click on the **Administration** link.
 - d. Click on the **CollabraSuite** link.
 - e. Click on the **Remote Server** link, then click on the **CS Remote Server** link.
 - f. Change the host and port in the **Base URL** text field.
 - g. Click **Finish** in the upper-right-hand corner of the window.
 - h. Log out of the WebCenter Interaction application.
 4. Modify CollabraSuite Campus settings in the CollabraSuite Administration tool.
 - a. Make sure that the server running CollabraSuite is running.
 - b. Log into the CollabraSuite administration using the CollabraSuite administrator account. The URL is `http://host:port/csuite/ui/admin`.
 - c. Make sure that the **Auto-Account Creation** checkbox is checked. If not, check this box and click **Save**.
 - d. Log out of the CollabraSuite administration.
 5. Configure WebCenter Interaction time-out.
 - a. Verify the following values are set in the `<PLUMTREE_INSTALL_DIR>/settings/portal/portalconfig.xml` file
 - `ImageServerConnectionURLTimeout: -1`

Modifying WebCenter Interaction Portlet Preferences

Administrators can configure WebCenter Interaction portlet preferences to customize the CollabraSuite configurations users see in WebCenter Interaction. To edit portlet preferences, the user must be a WebCenter Interaction community administrator and have an admin account on the CollabraSuite server.

1. Add the desired CollabraSuite portlets to a page.
2. Go to the WebCenter Interaction community hosting the CollabraSuite portlets, and click the **Edit This Community** link.
3. On the **Edit Community** page, click the **Portlet Preferences** link.
4. Click **OK** if prompted with a dialog indicating that the community must be saved before going to the next page.
5. All of the pages and all of the portlets used in the community will be displayed. Click on the pencil icon in the **Community** tab for a portlet.
6. On the **Edit Portlet Preferences** page, set the portlet preferences for that specific portlet.
7. Click the **Apply Preferences** button.
8. Repeat for each portlet to change preferences.

WebCenter Interaction Content Crawler Configuration

The CollabraSuite WebCenter Interaction Content Crawler enables users to utilize the WebCenter Interaction search capabilities to perform a full text search of documents in the CollabraSuite file cabinets. To install the content crawler:

1. Copy the `<applications_dir>/CSDocFetchCrawler.war` directory to the desired location on your web server.
2. Edit the `CSDocFetchCrawler.war\WEB-INF\web.xml` file and change the **cs.remote.crawler.campus.name** init parameter to be the name of the CollabraSuite campus that you wish to run the content crawler against.
3. If you have set up a different CollabraSuite administrator than the default 'csadmin' user, then edit the `CSDocFetchCrawler.war\WEB-INF\weblogic.xml` file and change the **principal-name** value for **security-role-assignment** element and the **run-as-principal-name** value for the **run-as-role-assignment** element to the login name of the new CollabraSuite administrator.

4. Edit the `CSDocFetchCrawler.war\WEB-INF\server-config.wsdd` file and replace the **IndexFilePath** and **IndexURLPrefix** parameters. **IndexFilePath** is the path on disk to the `CSDocFetchCrawler.war` file (with path separators escaped - i.e. `C:\\\\Path\\To\\CSDocFetchCrawler`) and **IndexURLPrefix** the URL equivalent path to the `CSDocFetchCrawler.war` web application (i.e. `http://host:port/CSDocFetchCrawler`)
5. Deploy the `CSDocFetchCrawler.war` application in WebLogic
6. Log into WebCenter Interaction as the administrator, and in the **Select Utility...** drop-down menu, select **Global Document Property Map**.
7. Click **Add Property**, select **Name** and **Title**, and click **OK**.
8. Click the **Name** and **Title** Property Name links, and under Document **Attributes**, type 'FileName'
9. Run the CollabraSuite Content Crawler job in the CollabraSuite Content Crawler directory.

Note: You may need to register the CollabraSuite Content Crawler job to an automation service. To make this happen, do the following:

- As a WebCenter Interaction administrator, click the **Select Utility...** drop-down and choose **Automation Service**.
- Click on the link for the WebCenter Interaction automation service that is running (if one is not running, then see your WebCenter Interaction documentation on how to create and run one)
- Click the **Add Folder** link, choose the **CollabraSuite Content Crawler** directory, and click finish.

Note: The updates from the content crawler may not appear in your search results or WebCenter Interaction directories until the following WebCenter Interaction jobs are run (they appear under the **Job** section of the **Intrinsic Operations** administration directory, and are typically configured by default to run every hour, but can also be run on demand):

- Document Refresh
- Search Update 1

Note: If CollabraSuite users are added to WebCenter Interaction after a document has been crawled, their permissions for that document will not be set on the document until the next crawl. How often the crawl is performed can be

configured by logging into WebCenter Interaction as an administrator and configuring the **CollabraSuite Content Crawler Job** in the **CollabraSuite Content Crawler** folder.

WebCenter Interaction

MediaWiki Integration

CollabraSuite provides a MediaWiki extension to embed CollabraSuite components within a wiki page by simply using the wiki extension tags defined in the CollabraSuite MediaWiki extension. A sample for adding a CollabraSuite chat component to a wiki page would be:

```
<CollabraSuite component="Chat" />
```

Prerequisites

The following prerequisites must be satisfied before installing CollabraSuite MediaWiki extension.

- A MediaWiki distribution is installed, configured and working properly. Detailed instructions can be found on the mediawiki website (<http://www.mediawiki.org/>).
- A CollabraSuite instance is deployed and running.
- The CollabraSuite instance is reachable via `http://host:port/csuite/ui`. Note that the root context of `/csuite/ui` may be different and can be configured within the `CollabraSuite.php` script.
- Although not required, knowledge of PHP is useful.

Installation

1. Copy the `CollabraSuite.php` script to `<mediawiki-dir>/extensions/CollabraSuite` directory

2. Edit the `CollabraSuite.php` script and modify the following where necessary:
 - a. The `serverUrl` variable in the `getServerUrl` function to point to the host and port where CollabraSuite is installed. Don't forget to modify the default value for the context (e.g. `csuite/ui/client`) if it is different.
 - b. Change the default value for the `campusName` variable in the `getCampusName` function to match the desired campus in the CollabraSuite environment.
3. Edit the `<mediawiki-dir>/extensions/LocalSettings.php` script and add the following line at the end of the file:

```
require_once("$IP/CollabraSuite/CollabraSuite.php");
```

4. Remember to remove or comment out the loading of any previous versions of the CollabraSuite mediawiki extension.
5. Restart the Apache or IIS server hosting the media wiki application.
6. Create a new or modify an existing wiki page and add the following tag to verify the extension is working properly:

```
<CollabraSuite component="Workplace"/>
```

7. CollabraSuite should load in the page.
8. Copy the text in the `/MediaWiki/CollabraSuite-wiki.txt` file found on the install CD into a wiki page to describe the capabilities and tag attributes to your wiki community.

Troubleshooting

1. Received an error when page loads reporting "campus <some campus name> does not exist."
 - The default campus defined in the CollabraSuite PHP script or the campus attribute used in the tag is incorrect.
2. The page does not load but the browser appears to have completed loading the page.
 - This is an indication that there is an error in the PHP script. View the apache or IIS error log and correct the error reported. This often happens whenever the attributes defined in the CollabraSuite PHP script contain a syntax error.

Eclipse Plugin

CollabraSuite provides an Eclipse plugin to embed CollabraSuite chat and online users components within an Eclipse view.

Prerequisites

The following prerequisites must be satisfied before installing CollabraSuite Eclipse Plugin.

- Eclipse 3.4 or later is installed.
- A CollabraSuite instance is deployed and running.
- The CollabraSuite instance is reachable via `http://host:port/csuite/ui`.

Installation

1. Start Eclipse.
2. Go to **Help->Software Updates**.
3. Click on the **Available Software** tab.
4. Click on the **Add Site...** button.
5. Enter the following location: `http://host:port/csuite/eclipse/update`.
6. A new entry will then appear in the **Available Software** tab.

7. Click on the plus (+) sign next to the newly entered site and on the CollabraSuite sub-item under the site name.
8. Check the box next to **CollabraSuite Plugins**, and click **Install**.
9. Click **Next**, accept the licensing agreement and finish the installation.
10. Restart Eclipse after the installation.

Running the Plugin

Once installed, the CollabraSuite Eclipse Plugin can be used by taking the following steps:

1. Start Eclipse.
2. Go to **Window->Show View->Other...**
3. Go to the **CollabraSuite** category, and choose the CollabraSuite component you want to open.
4. Click on the arrow on the top row of the CollabraSuite component and choose **Log In** in the drop-down menu.
5. Enter the CollabraSuite host, port, and campus name to connect to, as well as the CollabraSuite user name and password, then click **OK**.
6. To log out, close the view or choose the **Log Out** option which appears next to the **Log In** option in the drop-down menu.

Note: Opening up additional CollabraSuite views when logged into one will automatically log you into all CollabraSuite views. Logging out of one CollabraSuite view will automatically log you out of all CollabraSuite views.

Integration API

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.

Compiling

In order to begin compiling code using the Integration API, the following CollabraSuite JAR is required: `csuite-i9n-client.jar`. This JAR is located under the CollabraSuite installation in the `lib` directory. Additionally, the standard Java 2 Enterprise Edition (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-client.jar:${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-client.jar: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`. Note that all of the necessary modifications are automatically performed when installing CollabraSuite into an existing web application via WebLogic Workshop. See the **Installation Guide** for additional details.

Figure 1: Additions to web.xml

```
<ejb-ref>
  <ejb-ref-name>ejb/CSuiteAdmin</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <home>
com.collabrapace.csuite.server.i9n.interfaces.CSuiteAdminRemoteHome
  </home>
  <remote>
com.collabrapace.csuite.server.i9n.interfaces.CSuiteAdminRemote
  </remote>
</ejb-ref>
<ejb-ref>
  <ejb-ref-name>ejb/CSuiteCollaboration</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <home>
com.collabrapace.csuite.server.i9n.interfaces.CSuiteCollaborationRem
oteHome
  </home>
  <remote>
com.collabrapace.csuite.server.i9n.interfaces.CSuiteCollaborationRem
ote
  </remote>
</ejb-ref>
```

Figure 2: Additions to weblogic.xml

```

<ejb-reference-description>
  <ejb-ref-name>ejb/CSuiteAdmin</ejb-ref-name>
  <jndi-name>ejb/CSuiteAdmin</jndi-name>
</ejb-reference-description>
<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) provided by the CollabraSuite application. 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. Additionally, in WebLogic there is an extra JNDI parameter to pass in order to execute calls as a specific user instead of as the *anonymous* user. Setting `weblogic.jndi.enableDefaultUser` to `true` will allow the call to execute as the user specified in the `SECURITY_PRINCIPAL` parameter. 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, "username");

```

```

h.put(javax.naming.Context.SECURITY_CREDENTIALS, "password");
h.put("weblogic.jndi.enableDefaultUser", "true");
CSuiteAdminRemote csAdmin =
    CSuiteFactory.getCSuiteAdminRemoteInstance(h);

```

API Overview

The API consists of classes defined in two packages:

- `com.collabrapace.csuite.server.i9n.util`
- `com.collabrapace.csuite.server.i9n.ejb`

The `com.collabrapace.csuit.server.i9n.ejb` package contains two classes, one handling basic administrative functions and the other handling collaborative functions.

After installation, the CollabraSuite JavaDoc API documentation can be found at <http://host:port/csuite/docs>.

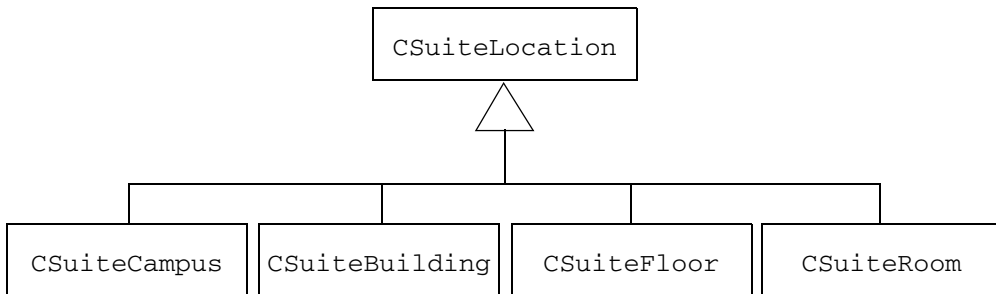
Utility Classes

The `com.collabrapace.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.collabrapace.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 represent a CollabraSuite location; those which represent a CollabraSuite folder or document; those which represent 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 representing a CollabraSuite location, as well as, extending the base class `CSuiteLocation` are represented below:

Figure 3: Location Class Hierarchy



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 build a location relative to 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`.

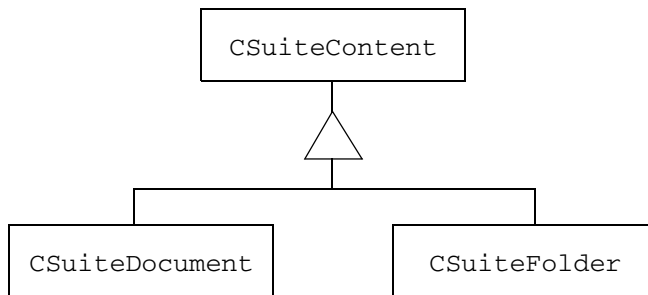
Finally, a third constructor form allows for individual strings, corresponding to each of the elements of the descriptor string. Thus, for the SampleRoom example above, the object can also be created using a constructor of the form `CSuiteRoom (“SampleCampus”, “SampleBuilding”, “SampleFloor”, “SampleRoom”)`.

The `CSuiteLocationInfo` object acts as a wrapper around a `CSuiteLocation` and contains additional information about a CollabraSuite location.

Where a `CSuiteLocation` object contains information only about a specific location (e.g. building), the `CSuiteLocationInfo` object contains information about the location's description and icon. In the case of a room it also contains its lockable status.

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

Figure 4: Content Class Hierarchy



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”. A forward slash (“/”) is always used as the file separator. 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 passing either a user (for Briefcase) or room (for File Cabinet) into the specific API methods. The descriptor string “/” always denotes the root of the location (either the File Cabinet or the Briefcase).

Once content is created, its metadata can be manipulated via `CSuiteDocumentInfo` and `CSuiteFolderInfo` objects. Content information can be read by using the `getFolderContentInfo()` methods, while it is set through the use of the `setContentInfo()` methods.

The classes responsible for representing a CollabraSuite group and 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”.

The `CSuiteACL` class defines an Access Control List comprised of a set of users and/or groups. These users and/or groups can be either granted or denied access to a location or resource.

The `CSuiteDocumentType` class represents a Document Type in CollabraSuite and consists of a mime-type, a file extension, a name, a description and an image.

Where the `CSuiteUser` class represents a CollabraSuite user, the `CSuiteUserInfo` class contains information about a user such as home room and contact information.

The `CSuitePriority` class is used to define 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 icon should also be supplied.

Finally, the API also defines the `CSuiteFactory` utility class to perform JNDI lookups associated with obtaining handles to the two session EJBs:

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, resource creation/deletion, and permission modification.

Information Retrieval and Modification

The information retrieval functions provide users with information relating to the structure of the CollabraSuite 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 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. 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 `CSuiteCampus` 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, the information retrieval methods are:

- `doesUserExist()`
- `getAssociates()`
- `getBuildings()`
- `getCampuses()`
- `getDocumentTypesByExtension()`
- `getFloors()`
- `getGroups()`
- `getLocationAccess()`
- `getOnlineUsers()`
- `getPriorities()`
- `getRooms()`
- `getSkills()`
- `getSkillsForUser()`
- `getUserInfo()`
- `getUserLocations()`
- `getUsers()`

The corresponding modification methods are:

- `createPriority()`
- `createSkill()`
- `deleteSkill()`
- `createGroup()`
- `deleteGroup()`

- `setGroupMembers()`
- `createUser()`
- `deleteUser()`
- `setUserInfo()`
- `modifyDocumentType()`

Location Administration

The location functions deal solely with the administration 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 are not created or that a location can be identified before it is deleted or modified. The location functions are:

- `createLocation()`
- `moveFloor()`
- `MoveRoom()`
- `deleteLocation()`
- `doesLocationExist()`
- `getLocationInfo()`
- `setLocationInfo()`

To create new locations, create a `CSuiteLocation` object representing the new location, then build a new `CSuiteLocationInfo` object using the `CSuiteLocation` object and add the additional information. Then use the `createLocation()` method to create the location.

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, `CSuiteLocation` objects can be used in the constructors for other `CSuiteLocations`, easing the developer's task in creating nested locations within a campus.

The `moveFloor()` and `moveRoom()` methods can be used to move floors and rooms to different locations within the same campus. They can also be used to rename a floor or room simply.

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 some 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).

In other instances, such as `setLocationAccess()`, a `CSuiteACL` object is used to specify the users, groups and the grant mode. The grant mode defines whether we intend to grant or deny access to the supplied users/groups. The constants used for the grant mode argument are defined in the `CSuiteACL` class.

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.collabrapace.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:

- `createDocument()`
- `createFolder()`
- `checkoutDocument()`
- `checkInDocument()`
- `deleteItem()`
- `getDocumentContents()`
- `getFolderContents()`
- `getFolderContentInfo()`
- `getItemInfo()`
- `setItemInfo()`

In order to modify a document the client must first check out the document from CollabraSuite. This can be accomplished by invoking either of the `checkoutDocument` methods (one method is for briefcase documents, the other for file cabinet documents). After the document is checked out, invoke the `getDocumentContents` method to retrieve the actual document to the local system. At this point the document may be modified either programmatically or via an external application (e.g. Word, Excel, etc.). Once modifications are complete, the document must be checked back into CollabraSuite. This can be accomplished by using one of the `checkInDocument` methods. In general, the call pattern to modify a document in CollabraSuite will be:

- a. `checkoutDocument()`
- b. `getDocumentContents()`
- c. `checkInDocument()`

The `setItemInfo()` method takes a `CSuiteContentInfo` that contains a `CSuiteACL`, explained above in the **Administrative Functions** section. In this case, the `CSuiteACL` also contains an `accessType` that defines whether permissions for an item are being set to read, write or read-write.

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

Exceptions

The `com.collabrapace.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 CollabraSuite that is being passed as a descriptor. An example is an incorrect path to a campus location.
- `MalformedDescriptorException` – 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.collabrapace.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.

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-client.jar:log4j.jar:${WL_HOME}/server/lib/weblogic.jar:.
-Dcs.log.debug=com.collabrapace CSuiteIntegrationClient
```

Transactions

It is often desirable to perform multiple Integration API method calls such that they are committed or rolled back as a group. This is accomplished by executing the multiple calls in the context of a single transaction.

If the caller does not have a current transaction, one will be started at the beginning of the Integration API method call and committed when the call successfully returns. The transaction will be rolled back if an exception is thrown. When the caller already has an active transaction, the Integration API methods will execute within the context of that transaction.

When invoking the Integration API from the Enterprise JavaBean tier of a J2EE application server, this is easily accomplished with container managed transactions. However, the transactions must be managed manually from the web tier or from a standalone client. The `CSuiteFactory` provides two convenience methods for manually managing transactions: `beginTransaction()` and `commitTransaction()`.

For a full discussion on the topic of transaction management please refer to the Enterprise JavaBeans Specification.

Sample Code

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

Creating a document

```
CSuiteCollaborationRemote collaboration =
    CSuiteFactory.getCSuiteCollaborationRemoteInstance();
CSuiteAdminRemote administration =
    CSuiteFactory.getCSuiteAdminRemoteInstance();
// Build a campus descriptor
String campusName = "SampleCampus";
String campusDesc = CSuiteCampus.buildCampusDescriptor(campusName);
CSuiteCampus csCampus = new CSuiteCampus(campusDesc);
// Owner of the document
String userName = "testUser";
String userDesc = CSuiteUser.buildUserDescriptor(campusName, userName);
CSuiteUser csUser = new CSuiteUser(userDesc);
// Get the file type for text documents
CSuiteDocumentType textDocType =
administration.getDocumentTypeByExtension(csCampus, "txt");
// Document's location inside of CollabraSuite
CSuiteDocumentInfo document =
    new CSuiteDocumentInfo(new CSuiteDocument("/", textDocType));
document.setDescription("File description");
// 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.createDocument(csUser, document, 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);
// Build a Set of recipients
Set toUsers = new HashSet();
toUsers.add(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);
}
```

Appendices

Changing the Authentication Method

The J2EE specification defines four methods for controlling how users supply authentication credentials. Those methods are BASIC, DIGEST, FORM and CLIENT-CERT. By default, CollabraSuite supports the BASIC and CLIENT-CERT authentication methods. The following steps should be taken in order to change the authentication method used by CollabraSuite:

1. Extract the `web.xml` file:

```
> jar -xvf CollabraSuite.ear csuite-ria.war
> jar -xvf csuite-ria.war WEB-INF/web.xml
```

2. Edit `web.xml` and change:

```
<login-config>
    <auth-method>BASIC,CLIENT-CERT</auth-method>
    <realm-name>default</realm-name>
</login-config>
```

3. Repackage the `CollabraSuite.ear` file:

```
> jar -uvf csuite-ext.war WEB-INF/web.xml
> jar -uvf CollabraSuite.ear csuite-ext.war
```

4. Redeploy the `CollabraSuite.ear` file.

Changing the Session Tracking Cookie Name

Because HTTP is a stateless protocol, an HTTP cookie is often used by application servers to maintain a user's session. CollabraSuite ships using the default cookie name of `JSESSIONID`. The following steps should be taken in order to change the session tracking cookie name used by CollabraSuite:

1. Extract the `weblogic.xml` file:

```
> jar -xvf CollabraSuite.ear csuite-ext.war
> jar -xvf csuite-ext.war WEB-INF/weblogic.xml
```

2. Edit `weblogic.xml` and add:

```
<wls:session-descriptor>
    <wls:cookie-name>CSJSESSIONID</wls:cookie-name>
    <wls:cookie-path>/</wls:cookie-path>
</wls:session-descriptor>
```

3. Repackage the `CollabraSuite.ear` file:

```
> jar -uvf csuite-ext.war WEB-INF/weblogic.xml
> jar -uvf CollabraSuite.ear csuite-ext.war
```

4. Redeploy the `CollabraSuite.ear` file.

CollabraSuite HTML Page

CollabraSuite includes a web page which can be used to provide specific CollabraSuite functionality. The page can display any predefined CollabraSuite component and also includes parameters used to configure the components. The pages can be accessed via `http://host:port/csuite/ui/client/component.jsp`.

Additional parameters may be specified on the URL to configure the components.

Below is a list of components available, along with the names, descriptions and default values for their configuration parameters. Required parameters are listed in **bold**.

User Creation Parameters

The following parameters can be passed to the `component.jsp` page to dynamically create or update a CollabraSuite user's information:

Note: This requires **Auto Account Creation** to be enabled on the campus. See the **CollabraSuite Administration Guide** for details.

Name	Description	Default Value
createUser	Dynamically create the user if they don't already exist	false
updateUser	Dynamically update the user's info when they login	false
loginId	The user's login ID	None
name	The user's full name	None
campus	The name of the user's campus	None
building	The name of the user's home building	None
floor	The name of the user's home floor	None
room	The name of the user's home room	None
emailAddress	The user's email address	None
workPhone	The user's work phone number	None
cellPhone	The user's cell phone number	None
homePhone	The user's home phone number	None
organization	The user's organization	None

Login Parameters

The following parameters may be specified on the `components.jsp` page as well as used by the JavaScript `csuite.login()` method. Bold fields are required.

Name	Description	Default Value
campus	The name of the user's campus	None
building	The name of the building to enter	None

Name	Description	Default Value
debug	Use uncompressed JavaScript and CSS code for debugging	false
floor	The name of the floor to enter	None
room	The name of the room to enter	None
session	The name of the session the components belong to	default

Components

The following components may be specified on the `components.jsp` page as well as used by the JavaScript `csuite.create()` method.

Name	Description
Admin	The Administration component
AdminDetail	Administration component without tree navigation
Associates	The Associates component
Briefcase	The Briefcase component
Chat	The Chat Monitor and Chat Input components
ChatInput	The ChatInput component
ChatMonitor	The ChatMonitor component
FileCabinet	The FileCabinet component
Navigator	The Navigator component
OccupantsChat	The Occupants and Chat components
OnlineUsers	The Online Users component.
OnlineUsersLite	The OnlineUsers component that displays only the user name.
PageComposer	The PageComposer component

Name	Description
PageViewer	The PageViewer component
Presenter	The Presenter component
Presentation	The RoomOccupants, Presenter and Chat components
Reports	The Reporting component
RoomContributors	The RoomContributors component
RoomOccupants	The RoomOccupants component
Suggestions	The Suggestions component
Toolbar	The Toolbar component
ToolbarLite	The pulldown toolbar component
Whiteboard	The Whiteboard component
Workplace	The Toolbar, RoomOccupants, RoomContributors, Chat, FileCabinet and Briefcase components

Component Parameters

The following parameters are common to all of the components:

Name	Description	Default Value
appletEnabled	Enable a Java applet for certain operations	true
campus	The name of the user's campus	None
building	The name of the building to enter	None
debug	Use uncompressed code for debugging	false
floor	The name of the floor to enter	None
height	The height of the component in pixels	None
room	The name of the room to enter	None

Name	Description	Default Value
session	The name of the session the components belong to	default
showAssociates	Display the Associates icon on the toolbar	true
showBriefcase	Display the Briefcase icon on the toolbar	true
showChat	Display the Chat icon on the toolbar	true
showConference	Display the Audio/Video Conference icon on the toolbar	true
showInviteUser	Display the InviteUser icon on the toolbar	true
showFileCabinet	Display the FileCabinet icon on the toolbar	true
showHelp	Display the Help icon on the toolbar	true
showNavigator	Display the Navigator icon on the toolbar	true
showOnlineUsers	Display the OnlineUsers icon on the toolbar	true
showPageViewer	Display the PageViewer icon on the toolbar	true
showPageUser	Display the PageUser icon on the toolbar	true
showPhoneUser	Display the Phone User icon on the toolbar	true
showPreferences	Display the Preferences icon on the toolbar	true
showPresenter	Display the Presenter icon on the toolbar	true
showPriority	Enable priority chat and page messages	false
showRoomLock	Display the RoomLock icon on the toolbar	true
showRoomOccupants	Display the RoomOccupants icon on the toolbar	true
showSidebarUser	Display the SidebarUser icon on the toolbar	true
showUserProfile	Display the UserProfile icon on the toolbar	true
showVideoTransmit	Display the Video Transmit icon on the toolbar	true
showWhiteboard	Display the Whiteboard icon on the toolbar	true

Name	Description	Default Value
showUserIcons	Display icons for users in various components	true
toolbar	Display a toolbar above the component	false
width	The width of the component in pixels	None

Appendices

Index

A

Additions

web.xml 3

weblogic.xml 4

additions 4

Administrative Functions 8

C

Code

compiling 1

running 2

Collaboration Functions 11

CollabraSuite 1, 2

Compiling

code 1

Connecting 4

Content Class Hierarchy 7

E

Exceptions 13

G

Getting Started 1

H

heading

level 1 1

Level 2 2

Hierarchy

content class 7

location class 6

I

Information Retrieval and Modification 8

Information Retrieval Methods 9

Integrate Portal 1

Introduction 1

introduction to WebLogic Portal 1

L

Level 1 heading 1

Level 2 heading 2

Location Class Hierarchy 6

Location Creation 10

Location Deletion 10

M

Modification Methods 9

O

Overview 1

P

Package com.collabrapace.csuite.server.i9n.ejb 14

administrative functions 8

information retrieval and modification 8

Permissions 11

Portal Extension 1

Prerequisites 1

R

regular paragraph 1

- Running
 - code 2
- S
- Samples 1
 - creating a document 15
 - retrieving online users 16
 - retrieving rooms 17
 - retrieving/printing a user's skills 18
 - sending a page to a user 16
- Single Sign On 1
- U
- Using API 1
- Utility Classes 2, 5
- W
- web.xml
 - additions 3
- WebLogic Portal
 - introduction 1
- Weblogic.xml 4