Agile

Engineering Collaboration Platform

ORACLE

PLM API Technical Guide

Engineering Collaboration 1.3 for Agile 9

Part No. E10832-01

Make sure you check for updates to this manual at the
Oracle Technology Network Website

Copyrights and Trademarks

October 10, 2007

REVISIONS

Revision	Date	Pages Affected	Description
A	10/08/2007	All	New release of PLM API Technical Guide

PREFACE

The Agile documentation set includes Adobe® Acrobat™ PDF files. The Oracle Technology Network (OTN) Web site (http://www.oracle.com/technology/documentation/index.html) contains the latest versions of the Oracle|Agile PLM PDF files. You can view or download these manuals from the Web site, or you can ask your Agile administrator if there is an Oracle|Agile Documentation folder available on your network from which you can access the Oracle|Agile documentation (PDF) files.

Note To read the PDF files, you must use the free Adobe Acrobat Reader™ version 7.0 or later. This program can be downloaded from the Adobe Web site (http://www.adobe.com).

The Oracle Technology Network (OTN) Web site (http://www.oracle.com/technology/documentation/index.html) can be accessed through **Help** > **Manuals** in both the Agile Web Client and the Agile Java Client. If applicable, earlier versions of Oracle|Agile PLM documentation can be found on the Agile Customer Support Web site (http://www.agile.com/support).

If you need additional assistance or information, please contact support@agile.com or phone (408) 284-3900 for assistance.

Note Before calling Agile Support about a problem with an Oracle|Agile PLM manual, please have ready the full part number, which is located on the title page.

Readme

Any last-minute information about Oracle|Agile PLM can be found in the Readme file on the Oracle Technology Network (OTN) Web site (http://www.oracle.com/technology/documentation/index.html).

Agile Training Aids

Go to the Agile Training Web page (http://training.agile.com) for more information on Agile Training offerings.

Chapter 1 Getting started

Chapter 2 Server Side PLM-API Functions

CHAPTER 1
Getting started

This chapter will instruct you how to install the PLM API both manually and with the installer provided and how to set up your environment for using the PLM API in your own projects. It contains the following topics:

- *Introdocution*
- *Prerequesites*
- *Installation*
- *Setting up a Development Project*
- *Understanding the API*
- *Running the Samples*

Introduction

In EC 1.3 the PLM API 2.0 is publicly available so that 3rd party solution/integration providers and customers can implement against the PLM API (client libraries) for direct connection to the PLM API server.

Prerequesites

PLM API is designed to run on any Java Runtime Environment (JRE), Version 1.4.2 or 1.5.0. When running on one of the supported application servers, only Java 1.4.2 is supported.

The PLM API is designed to run on any JRE version 1.4.2.

You may download a JRE from the following websites for the respective operating systems:

Table 1-1: Websites for a JRE Download

Windows	http://java.sun.com/j2se/
Solaris	http://java.sun.com/j2se/
Linux/Intel	http://java.sun.com/j2se/

PLM API Components

The PLM API 2.0 will be shipped with the following components:

- plm-api.jar
- plm-api-server.jar
- plm-api-ws.jar
- plm-api-cif.jar

These components have the following functions.

plm-api.jar

This component will be used by the client side and will contain all the classes needed for client to talk to the server.

plm-api-server.jar

This jar component will be responsible for the asynchronous mode of communication like queue handling, polling, subsequent sdk calls. Additionally this component will be using the Agile proprietary ECI libraries for communication.

plm-api-ws.jar

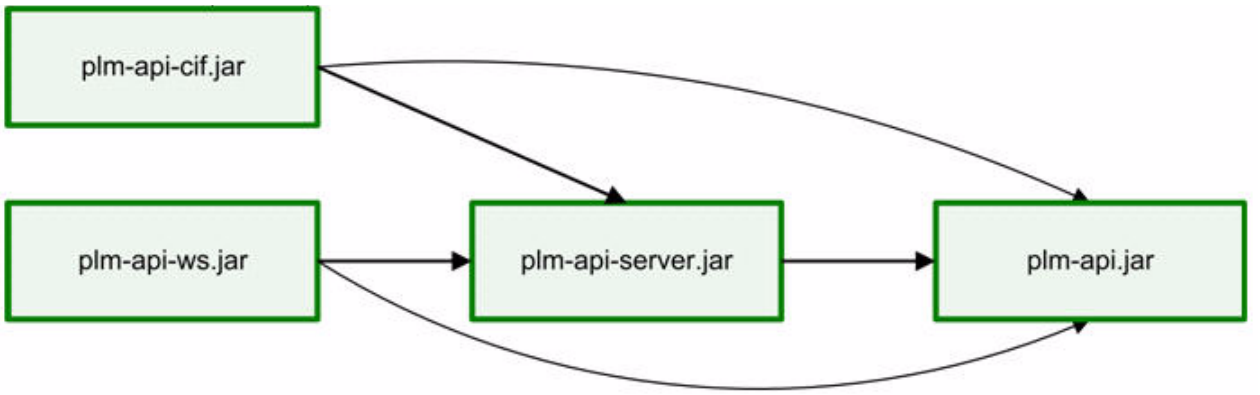
This jar file contains the components require to implement the WebService extension, including the WSX description file.

plm-api-cif.jar

This contains the pre-packaged extensions use case callables and handlers.

The following figure depicts the entire dependency scenario for these components. The file on the starting end depends on the file pointed to.

Figure 1-1: PLM-API component dependencies



Installation

Server Side Installation

This section will describe how to manually install the PLM API on the Agile A9 server, assuming that the Agile A9 system has been installed with the Oracle Application Server.

IBM Websphere

Note The Installer was not tested in a clustered environment!

Prerequesites

- Agile 9.2.2.1 is installed on Websphere

Installation of the Filefolders

First, the necessary folders with their files have to be created with the Agile9 Webclient:

- Unzip the filefolders.zip file on a client where you can start the Agile9 Webclient.
- Start the Agile9 Webclient with a user who has the privileges to create filefolder and attach files.
- Create the following filefolders and attach the files which can be found in the filefolders.zip file.

- PLMAPI_ASYNC: plm-api-server.xml
- PLMAPI_MAPPING: mastermapping.txt

  Note Any project specific mapping file listed in the mastermapping.txt should be stored in the configured file folder. See Chapter 1 of the PLM API Extensions Guide for further details

- PLMAPI_CONFIG: plm-api-config.properties
- PLMAPI_CONNECTOR: plm-api-sdk.xml
- PLMAPI_SCHEMA: plm-api-xml.xsd

Preparation/Configuration of the PLM-API Installation

- ❑ Log in as the Agile installation user.
- ❑ Copy the instplmapi.zip file to the <agile_home> folder.
- ❑ Unzip the instplmapi.zip file.
- ❑ Change to instplmapi direcory.
- ❑ Adapt the following parameters in install.properties:
  - plm.agile.destination.dir -> <agile_home>.
  - plm.agile.websphere.security -> if you deploy to a Java 2 security enabled Websphere, apply “y”, otherwise “n”.
  - plm.agile.websphere.restart -> if you want the installer to restart Websphere, apply “y”, otherwise “n” (restarting manually or by the installer with this switch is necessary if you have plm.agile.websphere.security -> “n”.

The next three steps are only necessary if you deploy to a Java 2 security enabled Websphere:

- ❑ Copy your current was.policy file of your currently deployed AgilePLM.ear application to ./instplmapi/ini/was_plmapi.policy
- ❑ Open the file ./instplmapi/ini/was_plmapi.policy
- ❑ Add the following lines to your "file: ${application}" section and save the file:

```
permission java.io.FilePermission "${was.install.root}${/}java${/}bin${/}..${/
}jre${/}lib${/}wsdl.properties", "read";
   permission java.util.PropertyPermission
"com.agile.share.prof.JavaProfiler.active", "read";
   // needed PLM-API Permissions
   permission java.io.FilePermission "${was.install.root}${/}ecp.log", "read,
write,delete";
   permission java.util.PropertyPermission "user.language", "write";
   permission java.util.PropertyPermission
javax.xml.parsers.DocumentBuilderFactory", "write";
   permission java.io.FilePermission "/plm-api-server.xml", "read";
```

PLM-API Installation

- ❑ Open a bash shell with the Agile installation user.
- ❑ Change to the home directory of the Agile installation user.
- ❑ Change to "./instplmapi".
- ❑ Set your java home environment variable to java installation of your WebSphere installation (e.g. export JAVA_HOME=/app/WebSphere/AppServer/java).
- ❑ Execute "./deploy.sh".

Now the PLM-API is deployed. If you deploy to Websphere with Java2 security NOT enabled, you have to restart Websphere manually as follows or by the installer as described above:

- To restart:
  - -> cd <agile_home>/agileDomain/bin/
  - -> ./StopAgile.sh
  - -> ./StartAgile.sh

Installation of DB queue

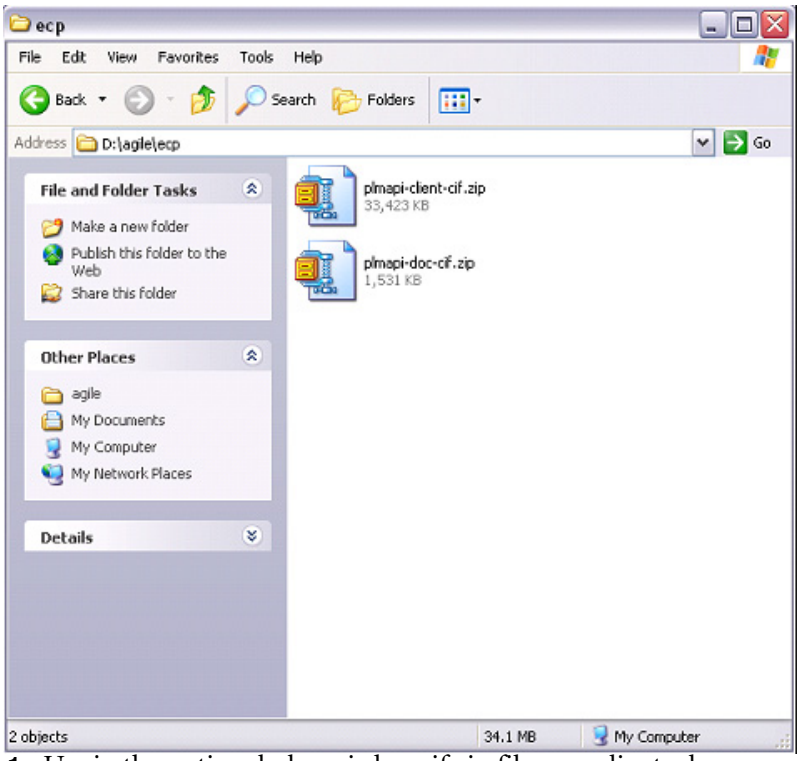
1. Login to the Database the Agile server is connecting to.
2. Run asyncservice.sql in the database. A new table will be created.
3. Make sure the Agile database connection points to the database where the sql is executed.

Client Side Installation

1. Unzip the plmapi-client-cif.zip file on a client where you want to develop against PLM API.

If you go to the install location, you will see that the following gzipped files have been created for you. In the remaining part of the tutorial you will be instructed how to setup the java libraries.

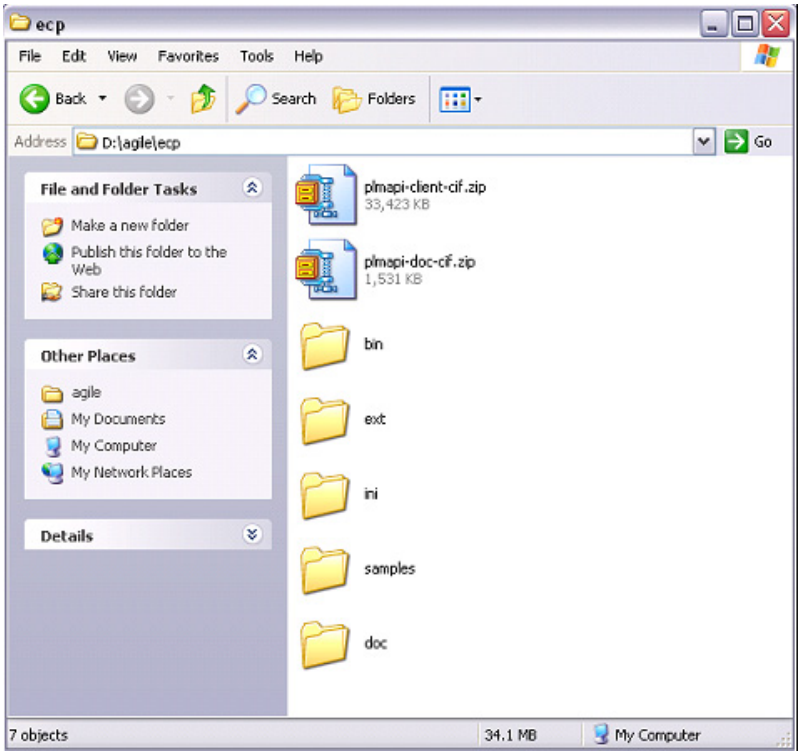
Figure 1-2: Installtion Location



1. Unzip the optional plmapi-doc-cif.zip file on a client where you want to develop against PLM API.

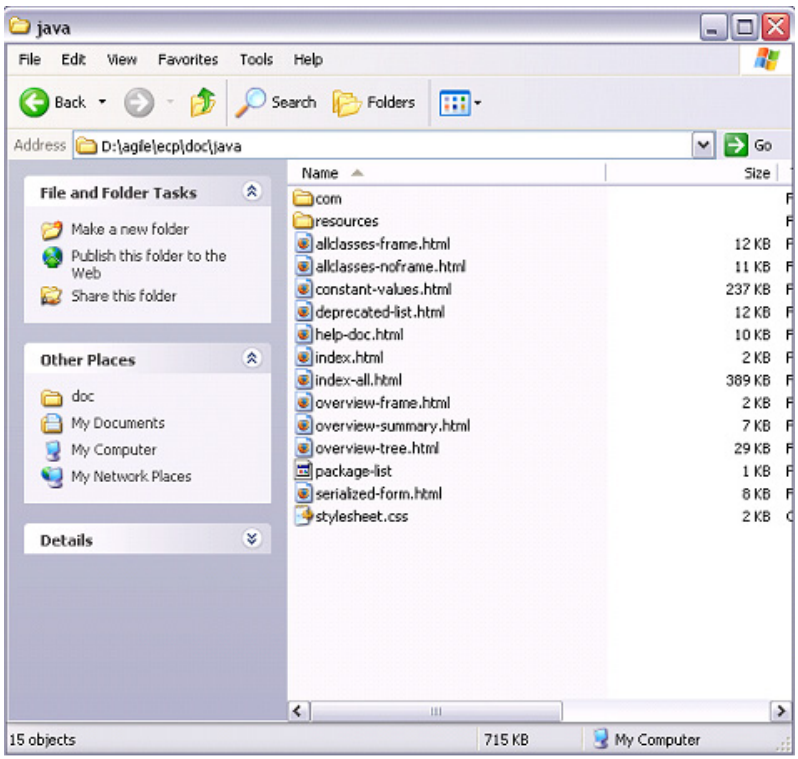
Now you can see the following directory structure that has been created (see the directory structure as depicted in the graphic below). Under the “bin” directory, the library files for the PLM API are located. The “ext” folder contains all the external third party libraries that you will need. The “ini” directory contains the configuration files that need to be in your projects class-path for the PLM API to access them. The “doc” folder contains the Javadoc files. Finally, the “sample” directory contains some sample programs for the PLM API.

Figure 1-3: Directory Structure



You can access the PLM API documentation at any time by going to the “INSTALL_LOCATION/doc/java” folder and opening the “index.html” file. Any frame-enabled browser (Firefox for instance) will typically open the documentation properly. You can then see an index of all the packages and classes of the PLM API on your left hand side frames and a description of whatever link you click there, on your right hand side frame.

Figure 1-4: Index Packages

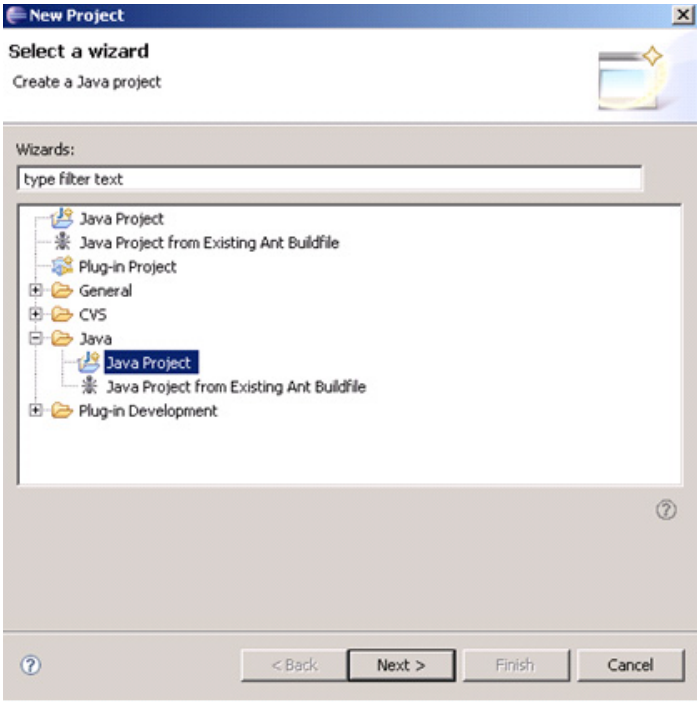


Setting up a Development Project

This chapter serves as an example of how to set up an own development project illustrated on the example of Eclipse. It should be a similar project setup for other integrated development denvironments (IDE).

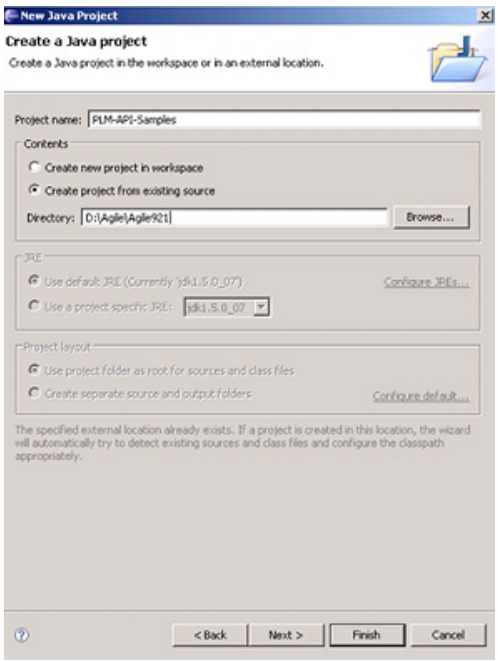
1 We will now try to run these sample programs. Open eclipse and create a new project via the File -> New -> Project menu. Select “Java Project” under the “Java” category and press “Next”.

Figure 1-5: Java Project



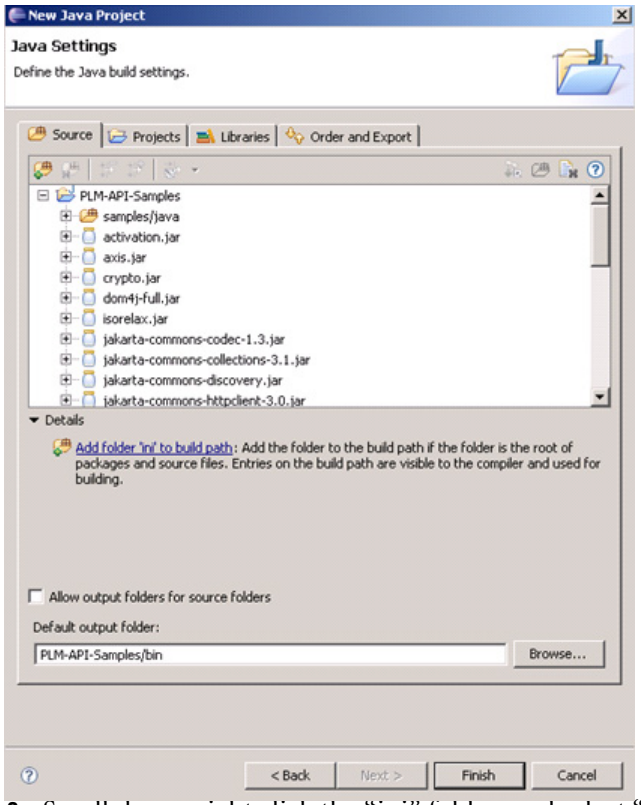
2 Fill in the “Project name” and under “Contents” select the checkbox labeled “Create project from existing source”. Browse to the directory where you extracted the PLM API. Press “Next”.

Figure 1-6: New Java Project

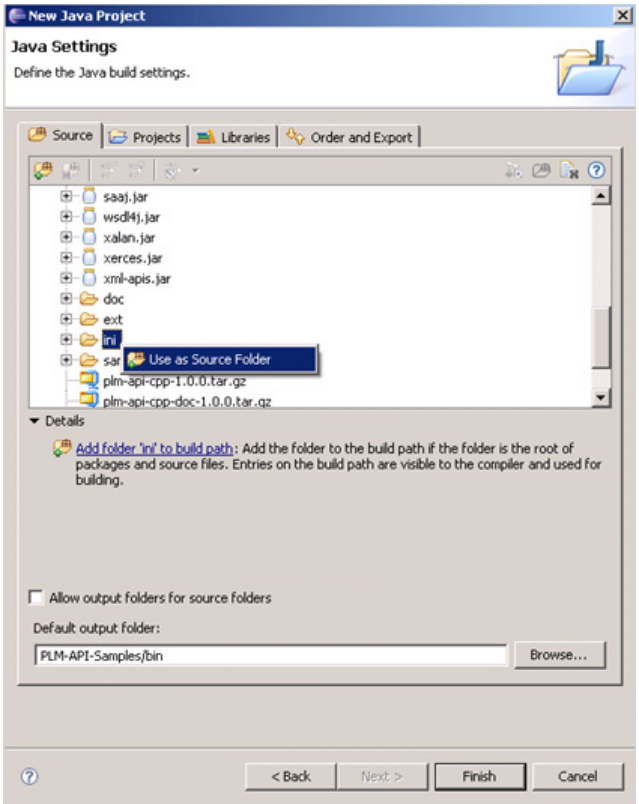


Now you can see that the Project Wizard discovers the sample programs as well as all the external libraries as well as the PLM API libraries required to run the samples.

Figure 1-7: Project Wizard



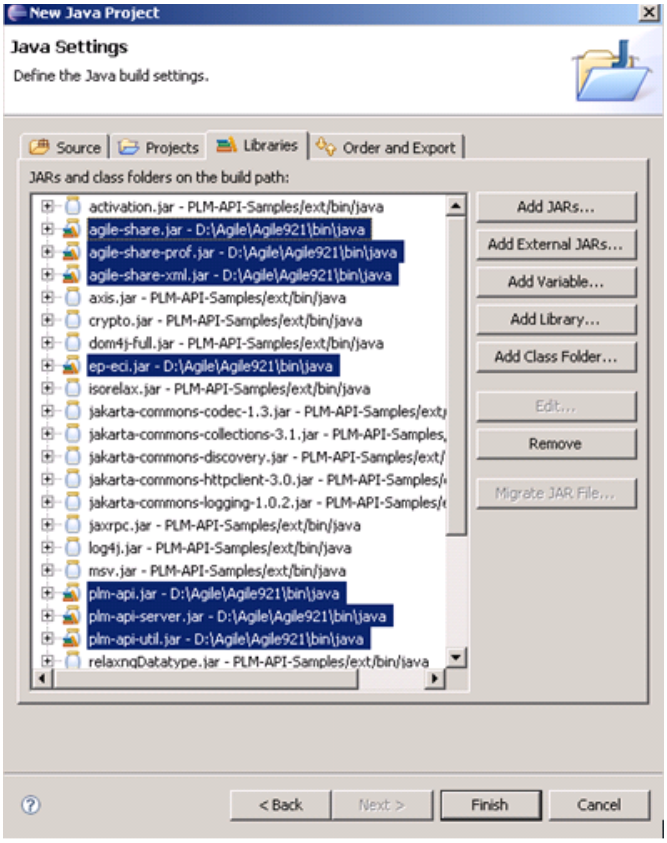
3 Scroll down, right click the “ini” folder, and select “Use as Source Folder”.



4 Click the “Libraries” tab and press “Add external jars”. Then navigate to the “bin/java” directory of PLM API extraction and add all the jar files to the project.

Note Please note that the folder ext\bin\java contains jar files that are labeled corresponding to the version of the A9 server that you are working with. These jar files include AgileAPI, agileclasses, ejb (only for A9.2 SP1) and pxapi. If you are working with version 9.2.2 or earlier of the PLM server, you need to add the files ending in A9.2 SP1 only, whereas if you are working with PLM server version 9.2.2.1 you need to add the files ending in A9.2 SP2 only.

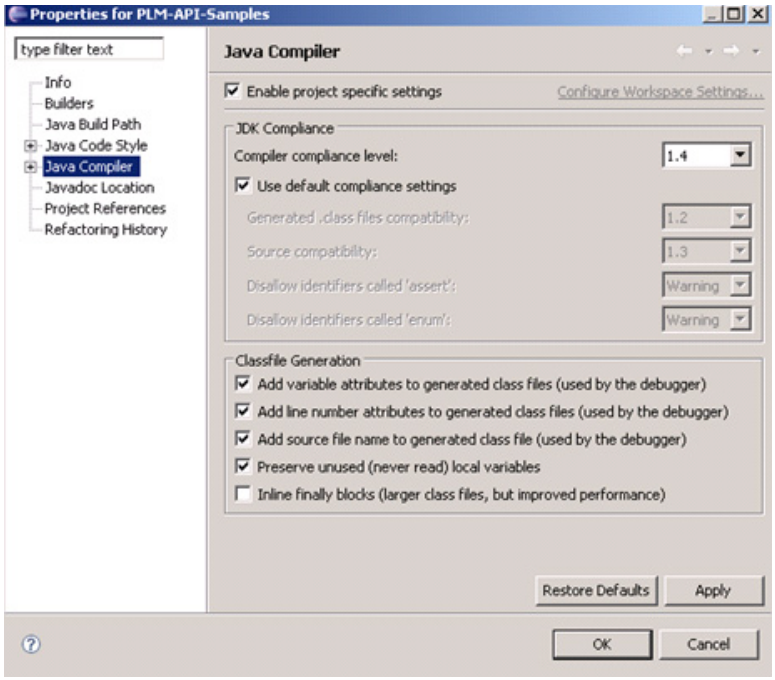
Figure 1-8: PLM-API Extractions



5 Click “Finish”.

You can now run the sample programs. If you get warnings after the resulting build process than it is probably because you are using JDK 1.5. The PLM API was written with JDK 1.4, but it should still work fine with JDK 1.5. If you want to remove these warnings, you can set the compliance settings for your compiler in your project to 1.4. This can be done by right clicking on your project in the “Package Explorer” view and selecting “Properties” there. Now change the setting to the ones shown below and the warnings will disappear.

Figure 1-9: Properties for PLM-API Samples



Understanding the API

The PLM API is a PLM agnostic API to access Agile's PLM systems. It provides a well defined and single API to Agile 9 (and Agile e6 in the future). This should help in minimizing the efforts to develop a client side code that should run on either of the Agile PLM systems (or both). Therefore, a general data model is provided that could be mapped to both PLM systems.

It is currently used in the Engineering Collaboration Platform (ECP) to connect both from a client application to the EC Client and from the EC Client to the PLM API's server side extensions. The supported protocols are Socket and Web Service (SOAP via HTTP(S)).

The server side extensions consist of a PLM specific connector that allows to plug-in any kind of function, which does PLM specific API calls. In case of Agile 9, the Agile SDK calls are made on the server side and are not doing remote RMI calls anymore, which was a huge performance enhancement.

The client side PLM API provides access to file servers and uses local file vaults where available (e.g. with iFS from Agile 9). This access is exposed as a function to the client and is also completely transparent for the different PLM systems.

In addition to a synchronous mode (blocking the client), an asynchronous mode is available to also allow long running operations without blocking the client.

Running the Samples

The samples are divided into three categories:

- General Samples (contains samples that demonstrate the general usage of PLM API)
- Synchronous Samples (contains samples that demonstrate the synchronous usage)
- Asynchronous Samples (contains samples that demonstrate the asynchronous usage)

All the samples are available in source code. It is advised that you open the samples while reading through this section.

General Samples

The general samples are located in the package com.agile.plmapi.samples.api.

BaseSample.java

This is the base class for the all the samples with a single utility method (getLoginParams) that shows you how to set up the parameters that help you in logging in to the system.

The PLM API can communicate in two ways: one through sockets with the EC-Client and the other using web-services with the Web Server proxy. You use the useClient flag to decide your communication mode.

The next convenience is that using the showLoginDialog flag. You can prompt the user for a username and password or supply a default one. These values are set by the KEY_USER and KEY_PASSWORD keys in the map. You can set your own values here or simply have the login dialog box pop-up. You can refer to the Javadoc for this list of keys and their description. After this map is constructed, it can be used to open a session. This is done by calling the createSession method on PlmFactory class. To see a list of other objects that can be created using the PlmFactory, refer to the documentation. The PlmFactory is responsible for creating all the objects that you will need in the PLM API.

The baseURL, the method argument here refers to the name of the machine where the Agile server is running.

CommonSample.java

The CommonSample.java is a straightforward example that uses the common PLM API. Below are the three simple tasks that this sample program performs. The statements in the brackets are the functions that perform the tasks:

- Creates a document (createDocument (...))
- Gets a document (getDocument (...))
- Checks-in a file with a document (checkInFile (...))

Before any of these methods are called, we need a PlmSession object to open a session with the Agile 9 server.

You can see how straightforward it is to create the session. All you need is the hostname of the machine running the Agile server, create a PlmSession object and call the open method on the session object. We already know how to create the login parameters in BaseSample.java, these are just passed in the session.open() method. If the session creation fails, you will get an exception (error message) describing the problem.

You could also check at some later time whether a session is still valid or not.

The function simply calls the PlmFactory to create a PlmObject of type “document”. This can easily be a “change”, “part”, “file” or so on. These values are being documented by us now and will be soon provided. However, for the time being you can refer to these values in the configuration file “plm-api.xml” under the “ini” directory. Additionally under each object name element, the names of attributes that this object type can take are listed.

The next important object is the PlmRequest object, which is used to wrap up our request to create this document. The PlmFactory is used to create a request object. You extract a PlmData object from this request and to this object add any number of PlmObjects that might be needed by your request to be executed properly. In this case, we need to create a document, so we add the document object to the PlmData object.

After these steps all you need to do is call the execute method on the PlmSession object with the proper callable. A list of these callable functions is available in the PLM API Extensions Guide. If the function is not supported, a PlmException will be thrown. If it executes successfully than we get back a response object. If anything went wrong during the document creation call, this can be checked by retrieving the status object from the response object and checking for errors. Otherwise, a PlmData object can also be retrieved from the PlmResponse object and used to verify the creation. Like stated earlier, work on a list of possible requests and the appropriate responses generated by the same is already under work and will soon be provided.

The other two tasks are retrieving a document and checking in a file. You can peek at them and see immediately what is happening. Therefore, this exercise is left to the reader to perform on his own and play with the API.

Synchronous Samples

The synchronous samples are located in the package com.agile.plmapi.samples.sync.

BaseSyncSample.java

This is the base class for the all the synchronous samples with some utility methods and some abstract methods that needs to be overwritten by the concrete sample classes.

SyncModeSample.java

This sample shows how to call an operation in synchronous mode without polling. For the polling sample see next section.

It simply calls the execute() method on a PlmSession.

SyncPollingModeSample.java

This sample shows how to call an operation in synchronous mode with polling. That will create a thread inside the method that waits for the response and thus is still a synchronous call from a user's perspective.

This mode is extremely useful when operating through firewalls or proxies that limit the connection time (connection cutoff after a certain amount of time).

It is activated by calling the setExecuteModePolling(true) method before the execute() method.

Asynchronous Samples

The asynchronous samples are located in the package com.agile.plmapi.samples.async.

For a detailed explanation of the different asynchronous modes, please see next chapter.

Basically, all asynchronous operations work with the different return type PlmFutureResponse of the method executeAsynch() that serves as a placeholder for the actual PlmResponse. Internally, a thread is started that waits for the PlmResponse. It is covered in the samples description how these PlmResponses could be accessed for the different asynchronous modes.

BaseAsyncSample.java

This is the base class for the all the asynchronous samples with some utility methods and some abstract methods that needs to be overwritten by the concrete sample classes.

AsyncPollingModeSample.java

This sample shows how to create an own thread in the client side code to receive the PlmReponse. There is no extra method call needed to use this behavior (except that you used the listener mode described below; then you need to call setExecuteAsynchModeListener(null) to enable the default behavior again).

The basic idea is to call the method getResult() on the PlmFutureResponse that will block until the response had been received from the server.

AsyncCallbackModeSample.java

This sample show how to register a callback listener class with the PlmFutureResponse returned from executeAsynch(). This listener will then be called with the PlmResponse object once the response had been received from the server.

It requires to implement an own class that inherits from PlmResponseListener and then register an instance of this class with the PlmFutureResponse by calling the setListener() method.

AsyncListenerModeSample.java

This sample shows how to use a servlet as the target of the response. This could be used if there is a servlet container available on the client side.

This mode is activated by calling the setExecuteAsynchModeListener() method on the PlmSession before calling the executeAsnych() method. The argument must be a string containing the URL to the target servlet.

There is also an AsyncListenerServlet base class provided that takes care of some housekeeping (as proper unmarshalling of the XML) and calls the abstract method onResponse() for further processing.

There is also a sample called AsyncListenerServletSample.java, that illustrated the usage of this base class.

CHAPTER 2
Server Side PLM-API Functions

This chapter describes the functions that could be called on the server side via PLM API. PLM API provides an extension framework with callables and handlers. This chapter contains the following topics:

- *Operations*

Operations

add-to-recently-visited

This action adds the objects included in the request to the current user's recently visited folder.

Action Constant

com.agile.plmapi.api.PlmActionConstants.ADD_TO_RECENTLY_VISITED

Action Entities

Part, change, document, filefolder.

Input

The request data must contain at least one entity. Each of these PlmObjects represents the object to be added to the user's recently visited folder.

Parameters

This function does not use any parameters.

Options

No options of the objects in the request are used when calling this function.

Result

The result of this operation will be summarized in the PlmStatus object associated with the PlmResponse. The PlmData will be empty.

Samples

The following sample adds an existing part to the recently visited folder of the currently logged-in user:

```
public void addtorecentlyvisited(PlmSession session) {
        try {
```

```
            PlmObject plmPart = PlmFactory.createObject("part");
            // Use the number value of a part that already exists on the PLM
            // server
            plmPart.setAttributeValue("number", "P00169");

                PlmObject plmDocument = PlmFactory.createObject("document");
               // Use the number value of a document that already exists on the PLM
                // server
                plmDocument.setAttributeValue("number", "D00041");

            PlmObject plmChange = PlmFactory.createObject("change");
 // Use the number value of a part that already exists on the PLM
            // server
            plmChange.setAttributeValue("number", "C00169");

            PlmObject plmFileFolder = PlmFactory.createObject("filefolder");
           // Use the number value of a file folder that already exists on the
            //PLM server
            plmFileFolder.setAttributeValue("number", "FOLDER00169");


PlmRequest request = PlmFactory.createRequest();
            PlmData data = request.getData();
            data.addObject(plmPart);
            data.addObject(plmDocument);
            data.addObject(plmChange);
            data.addObject(plmFileFolder);
 PlmResponse response =
session.execute(PlmActionConstants.ADD_TO_RECENTLY_VISITED_FOLDER, request);
            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError()) {
              System.out.println("add-to-recently-visited failed. Error code = " +
                                 status.getCode() + " " + status.getMessage());
            }
        } catch (PlmException e) {
            e.printStackTrace();
        }
    }
```

approve

This function approves all the top level change or filefolder class objects present in the request.

Action Constant

change:

com.agile.plmapi.connector.sdk.service.ChangeService.approveReject(PlmContext context, PlmRequest request, String action)

filefolder:

com.agile.plmapi.connector.sdk.service.FilefolderService. approveReject(PlmContext context, PlmRequest request, String action)

Action Entities

change, filefolder

Input

The request data may contain any number of toplevel PlmObjects of entity type 'change' or 'filefolder' which need to be approved.

Parameters

PlmImplConstants.PARAMETER_USERS_IDS. This parameter is required and allows to specify the users to send notification when approving the change object. This parameter must contain a list of the users' ids, separated by the delimiter |.| (a pipe sign followed by a dot and by another pipe sign).

PlmImplConstants.PARAMETER_APPROVE_REJECT_PASSWORD. This parameter is required and allows to send an encrypted version of the password of the currently logged-in user, required to approve a change. The password needs to be encrypted using a BlowFish encrypting mechanism.

PlmImplConstants.PARAMETER_COMMENT. This parameter is optional and allows to specify a comment to include when approving the change.

Options

No options of the object in the request are used when calling this function.

Result

The result of this operation will be summarized in the PlmStatus object associated with the PlmResponse. The PlmData in the response will be empty.

Samples

The following sample approves an existing change:

```
      public void change_approve(PlmSession session) {
          try {
              PlmObject plmChange = PlmFactory.createObject("change");
              plmChange.setAttributeValue("number", "C01244");
              PlmRequest plmRequest = PlmFactory.createRequest();
              //A list of valid user-ids can be retrieved with
   //userdata.getactiveusers callable
              plmRequest.getHeader().setParameter
  PlmImplConstants.PARAMETER_USERS_IDS, "6169343|.|704");
            plmRequest.getHeader().setParameter PlmImplConstants.PARAMETER_COMMENT,
  "Change seems ok.");
              plmRequest.getHeader().setParameter
  PlmImplConstants.PARAMETER_APPROVE_REJECT_PASSWORD,
  com.agile.plmapi.util.BlowFishEncrypter.encrypt("agile"));
              PlmData plmData = plmRequest.getData();
              plmData.addObject(plmChange);
              PlmResponse response = session.execute(PlmActionConstants.APPROVE,
  plmRequest);
              PlmStatus status = response.getStatus();
              if (status != null && response.getStatus().isError()) {
                  System.out.println("change.approve failed. Error code " +
  status.getCode() + " " + status.getMessage());
              }
          } catch (PlmException e) {
              e.printStackTrace();
          }
      }
```

The following sample approves the file folder given in the request.

```
public void filefolder_approve(PlmSession session) {
        try {
            PlmObject plmFileFolder = PlmFactory.createObject("filefolder");
            plmFileFolder.setAttributeValue("number", "FOLDER03021");
            PlmRequest plmRequest = PlmFactory.createRequest();
            //A list of valid user-ids can be retrieved with
     //userdata.getactiveusers callable
            plmRequest.getHeader().setParameter
PlmImplConstants.PARAMETER_USERS_IDS, "6169343|.|704");
          plmRequest.getHeader().setParameter PlmImplConstants.PARAMETER_COMMENT,
"Change seems ok.");
            plmRequest.getHeader().setParameter
PlmImplConstants.PARAMETER_APPROVE_REJECT_PASSWORD, com.agile.
plmapi.util.BlowFishEncrypter.encrypt("agile"));
            PlmData plmData = plmRequest.getData();
            plmData.addObject(plmFileFolder);
            PlmResponse response = session.execute(PlmActionConstants.APPROVE
plmRequest);
            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError()) {
                System.out.println("filefolder.approve failed. Error code " +
status.getCode() + " " + status.getMessage());
            }
        } catch (PlmException e) {
            e.printStackTrace();
        }
    }
```

change-status

This function changes the status of the change given in the request to the new status.

Action Constant

com.agile.plmapi.connector.sdk.change.ChangeChangeStatusHandler.changeStatus (PlmContext context, PlmRequest request)

Action Entities

change

Input

The request data must contain only one PlmObject of entity type ‘change’ representing the change whose status will be changed.

Parameters

PlmImplConstants.PARAMETER_DISABLED_WARNINGS: This parameter is used to specify which warnings should be disabled before attempting to change the status of the change object. This value should be a separated list of the warning codes, using the following character sequence as separator: |.| (a pipe sign followed by a dot and followed by another pipe sign). The warning codes must be relevant to the PLM connector implementation.

Note Although this parameter can be set before the first time the request is sent to the server, a typical use of it is related to reattempting the function call when the result of its execution is of type warning. That is, when the response’ status after calling this function is of type ‘warning’ (which can be determined by calling PlmStatus.isWarning()), the status’ code contains the code of the warning, so this value can be added to the disabled-warnings parameter to call the function again.

Options

PlmImplConstants.INTERNAL_OPTION_NEXT_STATUS_ID : This option is used to specify the id of the status to which the change object will be moved.

Result

The result of this operation will be summarized in the PlmStatus object associated with the PlmResponse. If the response's status is of type Warning, the warning code will be present in the response's status' code. The PlmData will be empty.

Samples

The following sample changes the status of an existing change to the status with id: 14302 (this value can be obtained by loading the change's metadata).

```
public void change_changestatus(PlmSession session) {
        try {
            PlmObject plmChange = PlmFactory.createObject("change");
            plmChange.setAttributeValue("number", "C01223");
 // Change the status of this change.
plmChange.setOptionValue(PlmImplConstants.INTERNAL_OPTION_NEXT_STATUS_ID,
"14302");
            PlmRequest plmRequest = PlmFactory.createRequest();
            //Uncomment the follwoing line to give a list of warnings each
            //seperated by these 3 characters "|.|"
            //plmRequest.getHeader().setParameter("disabled-warnings", "
integerValue1 |.| integerValue2 ...." );
            PlmData plmData = plmRequest.getData();
            plmData.addObject(plmChange);
         PlmResponse response = session.execute(PlmActionConstants.CHANGE_STATUS,
plmRequest);
            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError()) {
                System.out.println("change-status failed. Error code " +
status.getCode() + " " + status.getMessage());
            }
        } catch (PlmException e) {
            e.printStackTrace();
        }
    }
```

checkin

This action will change to PlmImplConstants.CHECKIN like all others but at the moment remains unchanged. This function iterates through the top-level objects of type 'document' in the request and creates or updates their cad file association (i.e. creates a file attachment) corresponding to the document-to-file relation.

Action Constant

com.agile.plmapi.connector.sdk.service.DocumentFileRelationService. associateWithFile(PlmContext context, PlmRequest request)

Action Entities

document

Input

The request data must contain at least one top-level object of entity type 'document', which should have one child object of type 'file', with a relation of type 'document-file-relation' binding them. The following attributes of the relation are required: "file-name", "file-path", "file-size", "file-description" and "cad-last-view-date". These values need to be set according to the properties of the file, see the sample below to determine how to set values for these attributes.

Parameters

This function does not use any parameters.

Options

The following three options of the PlmObject of type 'document' are optional; they determine how the file association should happen.

PlmImplConstants.OPTION_SELECTION: This option is used by the CAD integration to specify a value which will indicate that this type of file assignment is done through the CAD integration platform and is typically dependant on the CAD tool or CAD Connector.

lmImplConstants.OPTION_CHECK_IN: This option is used to specify whether the CAD file assignment should be checked-in if a previous CAD file assignment exists for the document and it is in checked-out status. Please see a complete description of the implications of this option below.

PlmImplConstants.OPTION_CHECK_OUT: This option is used to specify whether the CAD file assignment should be checked out before the function returns. Please see a complete description of the implications of this option below.

Result

The result of this operation will be summarized in the PlmStatus object associated with the PlmResponse. The PlmData will contain the same objects contained in the request, with the top-level 'document' object's attributes updated to reflect the values stored in the server. The 'document' object(s)' relations will not be updated with the current status in the server.

Regarding the options to check-in or check-out the CAD file assignment, please note the following:

1. The check-in option has an effect only if there is a previous version of the file already stored in the server. If this is the case, then the server behaves like this:
   a. If the previous version was not checked out, then the server checks-out the previous version, thus creating a new version of the previously stored file, overwrites this new version with the file sent in the request and then leaves the new association as the previous one was initially, that is, not checked-out (meaning that the server checks-in this new version, regardless of the setting in the check-in option).
   b. If the previous version was checked out, then the server replaces the checked-out version of the file with the new one and checks-in the new version only if the check-in flag is set to true.
2. Finally, before finishing the function, the server reads the value of the check-out option and if it's "true" then it checks-out the new version if it's not already checked-out.

Based on the explanation above, the following cases would be possible:

3. There is no previous CAD file association; neither flag is set: The result is a file association with version 1, not checked-out.
4. There is no previous CAD file association, only the check-in flag is set: The result is a file association with version 1, not checked-out.

5 There is no previous CAD file association, both flags are set: The result is a file association with version 2, checked-out (as soon as the version 1 is check-out, a new version of the association is created).

6 There is a previous CAD file association in version 1, not checked-out, neither flag is set: The result is version 2, not checked-out.

7 There is a previous CAD file association in version 1, not checked-out, only check-in flag is set: The result is version 2, not checked-out.

8 There is a previous CAD file association in version 1, not checked-out, both flags are set: The result is version 3, checked-out.

9 There is a previous CAD file association in version 2, checked-out, neither flags is set: The result is version 2, checked-out.

10 There is a previous CAD file association in version 2, checked-out, only check-in flag is set: The result is version 2, not checked-out.

11 There is a previous CAD file association in version 2, checked-out, both flags are set: The result is version 3, checked-out.

Samples

The following sample checks in a file attached to a document.

```
public void document_checkin(PlmSession session)
    {
        try {
            PlmObject plmDocument;
            plmDocument = PlmFactory.createObject("document");
            // Use the doc number of a document that already exists on the PLM
            // server
            plmDocument.setAttributeValue("number", "D00002");
            // plmDocument.setOptionValue("check-in", "true");
            // plmDocument.setOptionValue("check-out", "false");
            PlmObject plmFile;
            plmFile = PlmFactory.createObject("file");
            PlmRelation plmRelation = plmDocument.createRelation("document-file-
relation", plmFile);
            String filename = "1.txt";
            String filepath = "D:\\temp";
            File file = new File(filepath, filename);
            plmRelation.setAttributeValue("file-name", filename);
            plmRelation.setAttributeValue("file-path", filepath);
            plmRelation.setAttributeValue("file-size",
String.valueOf(file.length()));
            plmRelation.setAttributeValue("file-description", "Test");
            plmRelation.setAttributeValue("cad-last-view-date",
String.valueOf(file.lastModified()));
            PlmRequest plmRequest = PlmFactory.createRequest();
            PlmData plmData = plmRequest.getData();
            plmData.addObject(plmDocument);
          PlmResponse response = session.execute(PlmDocumentType.ACTION_CHECK_IN,
plmRequest);
            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError()) {
                System.out.println("document.checkin failed. Error code = "
                + status.getCode() + " " + status.getMessage());
            }
            plmDocument = (PlmObject)
response.getData().getObjects().iterator().next();
        } catch (PlmException e) {
```

```
            e.printStackTrace();
        }
    }
```

checkout

This function is a particular case of the more general function “UPDATE-CHECKOUT-STATUS” shown above for document class type objects, where the check-out-mode parameter is omitted and the action is always CHECKOUT. Therefore, this function checks-out the file(s) associated with the given document objects, which are represented by the relation of type document-to-file. The files are also downloaded.

Action Constant

com.agile.plmapi.connector.sdk.document.DocumentLocateAndCheckoutHandler.locateAndCheckOut(PlmContext context, PlmRequest request)

Action Entities

document

Input

The request data must contain at least one top-level object of entity type ‘document’, which should have at least one child object of type ‘file’, with a document-to-file relation binding them.

Parameters

This function does not use any parameters.

Options

No options of the objects in the request are used when calling this function.

Result

The result of this operation will be summarized in the PlmStatus object associated with the PlmResponse. The PlmData will contain the same objects and relations specified in the request; they will not be updated to show the updated status of the associated files.

Samples

The follwong sample checks out a file from the document specified.

```
public PlmObject document_checkout(PlmSession session) {
        try {
            PlmObject plmDocument;
            plmDocument = PlmFactory.createObject("document");
            // Use the doc number of a document that already exists on the PLM
            // server
            plmDocument.setAttributeValue("number", "D05029");
            //Specify a file entity that is attached to the document in question
            PlmObject plmFile;
            plmFile = PlmFactory.createObject("file");
            plmFile.setAttributeValue("file-name", "testclient");
            plmFile.setAttributeValue("folder-number", "FOLDER03021");


            PlmRelation plmRelation = plmDocument.createRelation("document-file-
relation", plmFile);
```

```
            plmRelation.setAttributeValue("folder-number",
plmFile.getAttributeValue("folder-number"));


            PlmRequest plmRequest = PlmFactory.createRequest();
            PlmData plmData = plmRequest.getData();
            plmData.addObject(plmDocument);
            PlmResponse response = session.execute(, plmRequest);
            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError()) {
                System.out.println("document.checkout failed. Error code = " +
status.getCode() + " " + status.getMessage());
            }
            plmDocument = (PlmObject)
response.getData().getObjects().iterator().next();
            return plmDocument;
        } catch (PlmException e) {
            e.printStackTrace();
        }
        return null;
    }
```

comment

This function comments the change or filefolder object present in the request. The comment functionality is server specific.

Action Constant

change:

com.agile.plmapi.connector.sdk.change. ChangeCommentHandler. comment (PlmContext context, PlmRequest request)

filefolder:

com.agile.plmapi.connector.sdk.filefolder. FilefolderCommentHandler. comment (PlmContext context, PlmRequest request)

Action Entities

change, filefolder

Input

The request data may contain any number of top level PlmObjects of entity type ‘change’ or filefolder to be commented.

Parameters

PlmImplConstants.PARAMETER_USERS_IDS: This parameter is required and allows to specify the users to send notification when commenting the FileFolder object. This parameter must contain a list of the users’ ids, separated by the delimiter |.| (a pipe sign followed by a dot and by another pipe sign).

PlmImplConstants.PARAMETER_COMMENT: This parameter allows to specify the text to use as comment.

PlmImplConstants.PARAMETER_NOTIFY_ORIGINATOR: This parameter is optional; if set to ‘true’ then the originator of the FileFolder will be notified of the comment action.

PlmImplConstants.PARAMETER_NOTIFY_CHANGE_COORDINATOR: This parameter is optional; if set to 'true' then the coordinator will be notified of the comment action.

PlmImplConstants.PARAMETER_NOTIFY_CHANGE_CONTROL_BOARD: This parameter is optional; if set to 'true' then the users included in the control board will be notified of the comment action.

Options

No options of the object in the request are used when calling this function.

Result

The result of this operation will be summarized in the PlmStatus object associated with the PlmResponse. The PlmData in the response will be empty.

Samples

The following sample adds a comment to the specified change.

```
public void change_comment(PlmSession session) {
        try {
            PlmObject plmChange = PlmFactory.createObject("change");
            plmChange.setAttributeValue("number", "C01223");
            PlmRequest plmRequest = PlmFactory.createRequest();
            //A list of valid user-ids can be retrieved with
     //userdata.getactiveusers callable

plmRequest.getHeader().setParameter(PlmImplConstants.PARAMETER_USERS_IDS,
"6169343|.|704");
          plmRequest.getHeader().setParameter(PlmImplConstants.PARAMETER_COMMENT,
"Change seems ok.");
            plmRequest.getHeader().setParameter(
PlmImplConstants.PARAMETER_NOTIFY_ORIGINATOR, "true");
            plmRequest.getHeader().setParameter (
PlmImplConstants.PARAMETER_NOTIFY_CHANGE_COORDINATOR, "true");
    plmRequest.getHeader().setParameter (
PlmImplConstants.PARAMETER_NOTIFY_CHANGE_CONTROL_BOARD, "true");
PlmData plmData = plmRequest.getData();
            plmData.addObject(plmChange);
            PlmResponse response = session.execute(PlmActionConstants.COMMENT,
plmRequest);
            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError()) {
                System.out.println("comment failed. Error code " +
status.getCode() + " " + status.getMessage());
            }
        } catch (PlmException e) {
            e.printStackTrace();
        }
    }
```

The following sample adds a comment to the specified file folder.

```
public void filefolder_comment(PlmSession session) {
        try {
            PlmObject plmFileFolder = PlmFactory.createObject("filefolder");
            plmFileFolder.setAttributeValue("number", "FOLDER03021");
            PlmRequest plmRequest = PlmFactory.createRequest();
            //A list of valid user-ids can be retrieved with
     //userdata.getactiveusers callable
```

```
plmRequest.getHeader().setParameter(PlmImplConstants.PARAMETER_USERS_IDS,
"6169343|.|704");
          plmRequest.getHeader().setParameter(PlmImplConstants.PARAMETER_COMMENT,
"Change seems ok.");
             plmRequest.getHeader().setParameter(
PlmImplConstants.PARAMETER_NOTIFY_ORIGINATOR, "true");
             plmRequest.getHeader().setParameter (
PlmImplConstants.PARAMETER_NOTIFY_CHANGE_COORDINATOR, "true");
    plmRequest.getHeader().setParameter (
PlmImplConstants.PARAMETER_NOTIFY_CHANGE_CONTROL_BOARD, "true");
             PlmData plmData = plmRequest.getData();
             plmData.addObject(plmFileFolder);
             PlmResponse response = session.execute(PlmActionConstants.COMMENT,
plmRequest);
             PlmStatus status = response.getStatus();
             if (status != null && response.getStatus().isError()) {
                System.out.println("filefolder.changecomment failed. Error code " +
 status.getCode() + " " + status.getMessage());
             }
         } catch (PlmException e) {
             e.printStackTrace();
         }
     }
```

create

This function creates one of more objects in the PLM server, according to the objects present in the request.

Action Constant

part, document :

com.agile.plmapi.connector.sdk.service.ItemService.createItem(PlmContext context, String operation, PlmRequest request)

change :

com.agile.plmapi.connector.sdk.handler.ChangeCreateHandler. createECO(PlmContext context, PlmRequest request)

Action Entities

part, change, document

Input

The request data must contain at least one PlmObject of class type ‘part’, ‘document’ or ‘change’. Each of these PlmObjects represents the entity that should be created in the PLM server.

Parameters

This function uses no parameters.

Options

These options only work for items (part and documents) at the moment and are ignored by change as only ECO is supported at the moment. The first autonumber source associated with the ECO subclass is used as the number source.

PlmImplConstants.OPTION_NUMBERSOURCE_ID: Provides a mechanism to specify what number source to use to retrieve an auto-number for the plm object to be created.

PlmImplConstants.OPTION_SUBCLASS: This option can be set for PlmObjects to specify the id or the name of the subclass to be used when creating a new object in the server.

Result

The result of this operation will be summarized in the PlmStatus object associated with the PlmResponse. Additionally, the PlmData will contain the representation of each object that was created, updated to reflect the values of the newly created object's attributes as currently set in the server.

Samples

The following sample can be used to create a part object in the server.

```
public PlmObject create(PlmSession session) {
        try {
            PlmObject part = PlmFactory.createObject("part");
            part.setAttributeValue("number", "#AUTO_NUMBER");
      //Uncomment below to specify which sub class
   //part.setOptionValue("PlmImplConstants.OPTION_SUBCLASS", "35707");
            PlmRequest request = PlmFactory.createRequest();
            request.getData().addObject(part);
 //Uncomment below to specify which auto number source
//request.getHeader().setParameter(PlmImplConstants.OPTION_NUMBERSOURCE_ID,
"12416");
 PlmResponse response = session.execute(PlmActionConstants.CREATE,
 request);
            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError()) {
                System.out.println("part.create failed. Error code = " +
status.getCode() + " " + status.getMessage());
            }
            PlmObject resultPart =
              (PlmObject)response.getData().getObjects().iterator().next();
            String number = resultPart.getAttributeValue("number");
           System.out.println("Part successfully created with number: " + number);
            return resultPart;
        } catch (PlmException e) {
            e.printStackTrace();
        }
        return null;
```

create-snapshot

This function creates a snapshot for the objects present in the request. A snapshot is a way to store the current revision of all components in a structure, so that at any point in time in the future, this structure can be retrieved, regardless of the latest revisions of the components at that point in time.

Action Constant

com.agile.plmapi.connector.sdk.callable.CreateSnapshotCallable.call(…)

Action Entities

document

Input

The request data must contain at least two top-level objects of entity type ‘document’, which represent the assembly and (at least one) component at their current revision, with a document-to-document relation binding them. Each of these objects must contain a document-to-file relation binding them to the CAD model file, which is represented by a PlmObject of entity type ‘file’.

Parameters

PlmImplConstants.PARAMETER_TOP_OBJECT_ID: This parameter is used to specify the number of the assembly at the top level of the structure, whose baseline will be created.

Options

PlmImplConstants.OPTION_SELECTED_TO_SAVE: A snapshot will be created for each object which has this option set to ‘true’. This allows to create a snapshot for subassemblies as well as for the top-level assembly in the structure.

Result

The result of this operation will be summarized in the PlmStatus object associated with the PlmResponse. The PlmData will contain the same objects and relations as the request, no attributes will have been updated with the values in the server.

Samples

The following sample shows how to create an assembly of documents, checkin files against them and then create a snapshot.

```
public PlmObject document_createbaseline(PlmSession session) {
        try {
            PlmObject plmDocumentParentAss1;
            PlmObject plmDocumentChild1Ass1;
            PlmObject plmDocumentChild2Ass1;

            // Create components of assembly
            plmDocumentParentAss1 = document_create(session);
            plmDocumentParentAss1.setOptionValue("selected-to-save", "true");
            plmDocumentChild1Ass1 = document_create(session);
            plmDocumentChild2Ass1 = document_create(session);

            PlmRelation documentRelation1;
            PlmRelation documentRelation2;

            try {
                // Create assembly
               documentRelation1 = plmDocumentParentAss1.createRelation("document-
document-relation", plmDocumentChild1Ass1);
                documentRelation1.setAttributeValue("child-number",
plmDocumentChild1Ass1.getAttributeValue("number"));
                documentRelation1.setAttributeValue("quantity", "1");

               documentRelation2 = plmDocumentParentAss1.createRelation("document-
document-relation", plmDocumentChild2Ass1);
                documentRelation2.setAttributeValue("child-number",
plmDocumentChild2Ass1.getAttributeValue("number"));
                documentRelation2.setAttributeValue("quantity", "1");
            } catch (PlmException e) {
               System.out.println("Failed to create document document relation.");
                e.printStackTrace();
```

```
            }

            PlmRequest request = PlmFactory.createRequest();
            request.getHeader().setParameter("delete-rows ", "false");
            PlmData data = request.getData();
            data.addObject(plmDocumentParentAss1);
            PlmResponse response = session.execute("document.createrelations",
request);
            PlmStatus status = response.getStatus();
            if (status.isError()) {
                System.out.println("Call to document.createrelations failed");
                return null;
            }

            // Check in files for each document, so we can create a baseline for
            // them
            checkinFile(plmDocumentParentAss1, session, "D:\\temp" ,"1.txt");
            checkinFile(plmDocumentChild1Ass1, session, "D:\\temp" ,"2.txt");
            checkinFile(plmDocumentChild2Ass1, session, "D:\\temp" ,"3.txt");

            request = PlmFactory.createRequest();

request.getHeader().setParameter(PlmImplConstants.PARAMETER_TOP_OBJECT_ID,
plmDocumentParentAss1.getAttributeValue("number"));
            request.getData().addObject(plmDocumentChild1Ass1);
            request.getData().addObject(plmDocumentChild2Ass1);
            request.getData().addObject(plmDocumentParentAss1);
          response = session.execute(PlmActionConstants.CREATE_SNAPSHOT, request);
            status = response.getStatus();
            if (status != null && response.getStatus().isError()) {
                System.out.println("create-snapshot failed. Error code = " +
status.getCode() + " " + status.getMessage());
            }
        } catch (PlmException e) {
            e.printStackTrace();
        }
        return null;
    }
```

This is the helper method used by document_createbaseline to check in files for a given document.

```
private void checkinFile(PlmObject plmDocument, PlmSession session, String
filepath, String filename) {
        try {
            PlmObject plmFile;
            plmFile = PlmFactory.createObject("file");
            PlmRelation plmRelation = plmDocument.createRelation("document-file-
relation", plmFile);
            File file = new File(filepath, filename);
            plmRelation.setAttributeValue("file-name", filename);
            plmRelation.setAttributeValue("file-path", filepath);
            plmRelation.setAttributeValue("file-size",
String.valueOf(file.length()));
            plmRelation.setAttributeValue("file-description", "Test");
            plmRelation.setAttributeValue("cad-last-view-date",
String.valueOf(file.lastModified()));
            PlmRequest plmRequest = PlmFactory.createRequest();
            PlmData plmData = plmRequest.getData();
            plmData.addObject(plmDocument);
          PlmResponse response = session.execute(PlmDocumentType.ACTION_CHECK_IN,
plmRequest);
            PlmStatus status = response.getStatus();
```

```
            if (status != null && response.getStatus().isError()) {
             System.out.println("check-in failed. Error code = " + status.getCode()
+ " " + status.getMessage());
            }
            plmDocument = (PlmObject)
response.getData().getObjects().iterator().next();
        } catch (PlmException e) {
            e.printStackTrace();
       }
  }
```

create-folder

This function creates a child folder inside a parent folder, which, in turn, is part of the structure of the current user's recently visited folder or home folder.

Action Constant

com.agile.plmapi.connector.sdk.service.UserDataService.createFolder(PlmContext context, PlmRequest request)

Action Entities

userdata

Input

The request data can be empty.

Parameters

PlmImplConstants.PARAMETER_PARENT_FOLDER: Contains the id of the folder within which the child folder will be created.

PlmImplConstants.PARAMETER_CHILD_FOLDER: Contains the name of the new folder to be created.

Options

No options of the object in the request are used when calling this function.

Result

The result of this operation will be summarized in the PlmStatus object associated with the PlmResponse. The PlmData in the response will be empty.

Samples

The following sample shows how to creates a folder under a users "Home Folder". These id's can be retrieved as shown in userdata.gethomefolder.

```
public void user_createfolder(PlmSession session) {
        try {
            PlmRequest plmRequest = PlmFactory.createRequest();
            //Id of home folder

plmRequest.getHeader().setParameter(PlmImplConstants.PARAMETER_PARENT_FOLDER,
"100");
```

```
plmRequest.getHeader().setParameter(PlmImplConstants.PARAMETER_CHILD_FOLDER,
"TestFolder");
          PlmResponse response = session.execute(PlmActionConstants.CREATE_FOLDER,
plmRequest);
             PlmStatus status = response.getStatus();
             if (status != null && response.getStatus().isError()) {
                 System.out.println("create-folder failed. Error code " +
status.getCode() + " " + status.getMessage());
             }
         } catch (PlmException e) {
             e.printStackTrace();
         }
     }
```

create-structure

This function scans the request's top-level PlmObjects looking for those of class type 'part' or 'document'and then creates or updates the relation with the PLM server for each PlmRelation it finds of the PlmObject. The value for this action constant is "create-structure".

Note If the child PlmObject of the relation represents an entity that does not exist in the server, an object is created based on the attributes of the child PlmObject (also of class 'part' or 'document').

Action Constant

com.agile.plmapi.connector.sdk.service.ItemCreateStructureService.createStructure (PlmContext context, PlmRequest request)

Action Entities

part, document

Input

The request data must contain at least one top-level object of class type 'part' or 'document' which can further have children of class type 'document' or 'part'.

Parameters

PlmImplConstants.PARAMETER_DELETE_ROWS: Flag to specify whether to delete previously existing relations. By default, the PLM connector will delete the CAD-integration-related relations of a part or document, which are no longer present in the request. If this parameter is set to 'false', then the existing CAD-integration-related relations will be preserved, even if they are no longer present according to the structure sent in the request.

Options

No options of the objects in the request are used when calling this function.

Result

The result of this operation will be summarized in the PlmStatus object associated with the PlmResponse. The PlmData will contain the same objects and relations as the request, no attributes will have been updated with the values in the server.

Samples

The samples below show how to create a single type of relation for an object but it is now possible to mix different relations in the same request to create all the relations shown below by this calling this single action.

The following sample creates a relation between a parent part and a child document, which are first created with the function provided as sample for the functions part.create and document.create:

```
public void part_createdocumentrelations(PlmSession session) {
        try {
            PlmObject parentPart = part_create(session);
            PlmObject childDocument = document_create(session);
            PlmRelation partDocumentRelation;
            partDocumentRelation = parentPart.createRelation("part-document-
relation",
                                                                childDocument);
            partDocumentRelation.setAttributeValue("child-number",
                                childDocument.getAttributeValue("number"));

            PlmRequest request = PlmFactory.createRequest();
            PlmData data = request.getData();
            data.addObject(parentPart);
       PlmResponse response =
session.execute(PlmActionConstants.CREATE_STRUCTURE, equest);
            PlmStatus status = response.getStatus();
            if (status.isError()) {
                System.out.println("Call to create-structure failed");
            }
        } catch (PlmException e) {
            e.printStackTrace();
        }
    }
```

The following sample creates a structure between a parent part and two children parts:

```
public PlmObject part_createstructure(PlmSession session) {
        try {
            PlmObject plmPartParent;
            PlmObject plmPartChild1;
            PlmObject plmPartChild2;

            plmPartParent = create_part(session);
            plmPartChild1 = create_part(session);
            plmPartChild2 = create_part(session);

            PlmRelation partPartRelation1;
            PlmRelation partPartRelation2;

            try {
                  partPartRelation1 = plmPartParent.createRelation("part-part-
  relation",
                                                                plmPartChild1);
                partPartRelation1.setAttributeValue("child-number",
                            plmPartChild1.getAttributeValue("number"));
                partPartRelation1.setAttributeValue("quantity", "1");
                partPartRelation2 = plmPartParent.createRelation("part-part-
relation",
                                                                plmPartChild2);
                partPartRelation2.setAttributeValue("child-number",
                                    plmPartChild2.getAttributeValue("number"));
```

```
                partPartRelation1.setAttributeValue("quantity", "1");
            } catch (PlmException e) {
                System.out.println("Failed to create part part relation.");
                e.printStackTrace();
            }

            PlmRequest request = PlmFactory.createRequest();
            request.getHeader().setParameter("delete-rows ", "false");
            PlmData data = request.getData();
            data.addObject(plmPartParent);
          PlmResponse response = session.execute("part.createstructure", request);
            PlmStatus status = response.getStatus();
            if (status.isError()) {
                System.out.println("Call to create-structure failed");
            }
            data = response.getData();
            // The returned PlmObject with only updated values from server, object
            // still not updated with structure value read from server
           PlmObject resultPart = (PlmObject) data.getObjects().iterator().next();
            return resultPart;
        } catch (PlmException e) {
            e.printStackTrace();
        }
        return null;
    }
```

The following sample creates a parent, adds 2 children documents to it and another further document to one of the children.

```
public void document_createrelations(PlmSession session) {
        try {
            // Create some documents to create document relations amongst.s
            PlmObject plmParentDoc = document_create(session);
            PlmObject plmChildDoc1 = document_create(session);
            PlmObject plmChildDoc2 = document_create(session);
            PlmObject plmChildDoc3 = document_create(session);

            PlmRelation plmRelation1 = plmParentDoc.createRelation("document-
document-relation", plmChildDoc1);
            plmRelation1.setAttributeValue("child-number",
plmChildDoc1.getAttributeValue("number"));
            PlmRelation plmRelation2 = plmParentDoc.createRelation("document-
document-relation", plmChildDoc2);
            plmRelation2.setAttributeValue("child-number",
plmChildDoc2.getAttributeValue("number"));
            PlmRelation plmRelation3 = plmChildDoc2.createRelation("document-
document-relation", plmChildDoc3);
            plmRelation3.setAttributeValue("child-number",
plmChildDoc3.getAttributeValue("number"));

            PlmRequest plmRequest = PlmFactory.createRequest();
            // Be careful to set this flag to false if you don't want your
            // previous cad-related relations to
            // be deleted that are excluded in the request.
            plmRequest.getHeader().setParameter("delete-rows", "false");
            PlmData plmData = plmRequest.getData();
            plmData.addObject(plmParentDoc);
            PlmResponse response =
session.execute(PlmActionConstants.CREATE_STRUCTURE, plmRequest);
            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError()) {
```

```
                System.out.println("Call to create-structure failed. Error code =
" + status.getCode() + " " + status.getMessage());
            }
        } catch (PlmException e) {
            e.printStackTrace();
        }
    }
```

delete

This function deletes one of more objects in the PLM server, according to the objects present in the request. The value of this action constant is “delete”.

Note In the a9 server, the server architecture allows an object to be soft-deleted, meaning that it’s moved to the ’recycle-bin’, from where it can be restored or ‘hard-deleted’ (permanently deleted from the server records). The PLM connector for EC 1.3 only performs a ‘soft-delete’ action on the objects.

Action Constant

com.agile.plmapi.connector.sdk.service.ItemService.delete(PlmContext context, PlmRequest request)

Action Entities

part, document

Input

The request data must contain at least one PlmObject of class type ‘part’ or ‘document’. Each of these PlmObjects represents the part or document that should be deleted.

Parameters

This action does not use parameters.

Options

No options of the objects in the request are used when calling this action.

Result

The result of this operation will be summarized in the PlmStatus object associated with the PlmResponse. The PlmData will contain the same objects that were passed in; no attributes values will be updated.

Samples

The following code deletes a part and a document that exist in the PLM server:

```
public void delete(PlmSession session)
{
    try {

        PlmObject plmPart = PlmFactory.createObject("part");
        // Use the number value of a part that already exists on the PLM
        // server
        plmPart.setAttributeValue("number", "P01334");

        PlmObject plmDocument = PlmFactory.createObject("document");
        // Use the doc number of a document that already exists on the PLM
        // server
```

```
            plmDocument.setAttributeValue("number", "D05022");
            PlmRequest request = PlmFactory.createRequest();
            PlmData data = request.getData();
            data.addObject(plmPart);
            data.addObject(plmDocument);
            PlmResponse response = session.execute(PlmActionConstants.DELETE,
request);
            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError()) {
                System.out.println("part.delete failed. Error code = "
                        + status.getCode() + " "
                        + status.getMessage());
            }
        } catch (PlmException e) {
            e.printStackTrace();
        }
    }
```

delete-folder-content

This function deletes an object from a folder, which is part of the structure of the current user's recently visited folder or home folder.

Action Constant

com.agile.plmapi.connector.sdk.service.UserDataService.deleteFolderContent (PlmContext context, PlmRequest request)

Action Entities

userdata

Input

The request data can be empty.

Parameters

PlmImplConstants.PARAMETER_PARENT_FOLDER: Contains the id of the folder whose content will be deleted.

PlmImplConstants.PARAMETER_CHILD_FOLDER: Contains the id of the content element to be removed from the parent folder.

Options

No options of the object in the request are used when calling this function.

Result

The result of this operation will be summarized in the PlmStatus object associated with the PlmResponse. The PlmData in the response will be empty.

Samples

The following sample shows how to delete a node under the "Recently Visited". These ids can be retrieved as shown in userdata.getrecentlyvisited folder. Analgously any node can be deleted under the home folder by retrieving the ids as shown in "GET_HOME_FOLDER".

```
public void user_deletefoldercontent(PlmSession session) {
```

```
        try {
            PlmRequest plmRequest = PlmFactory.createRequest();
            //Id of the recently visited folder

plmRequest.getHeader().setParameter(PlmImplConstants.PARAMETER_PARENT_FOLDER,
"6084433");

plmRequest.getHeader().setParameter(PlmImplConstants.PARAMETER_CHILD_FOLDER,
"6169308" );
            PlmResponse response =
session.execute(PlmActionConstants.DELETE_FOLDER_CONTENT, plmRequest);
            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError()) {
              System.out.println("userdata.deletefoldercontent failed. Error code
" +
status.getCode() + " " + status.getMessage());
            }
        } catch (PlmException e) {
            e.printStackTrace();
        }
    }
```

execute-query

This function executes a query as specified in the request and returns all objects found as a result of the query's execution.

Action Constant

com.agile.plmapi.connector.sdk.service.QueryService. executeQuery(PlmContext, PlmRequest, Logger)

Action Entities

part, change, document, filefolder

Input

The request data must contain one top-level object of entity type 'savedquery', which describes the query to be executed, either as a saved query (already existing in the server) or as a new query including a query criteria.

According to the statement above, in order to execute a new query, the request needs to contain a PlmObject of entity type 'savedquery', whose attributes need to be properly populated as follows:

<table>
<tr><th>Attribute Name</th><th>Value</th><th>Required</th></tr>
<tr><td>name</td><td>(Name to give to the query, mostly used to save a new query)</td><td>Not for this function</td></tr>
<tr><td>entity</td><td>The entity to query for, in this case "part"</td><td>Yes</td></tr>
<tr><td>criteria</td><td>The criteria for the query, specified using the following syntax:
[attribute-name] operand [parameter-numerical-order] [logical-operand [attribute-name] operand [parameter-numerical-order]]
For example, a query to search for all parts where the number attribute starts with 'P' and the creating-system attribute contains 'SW' would be specified as follows (together with the value of the 'parameters' attribute for the savedquery PlmObject specified below):
<code>[number] start with %0 and [creating-system] contains %1</code>
Figure 2-1: List of possible operands
<table><tr><th>Operand</th><th>Description</th></tr><tr><td>==</td><td>Equal to</td></tr><tr><td>!=</td><td>Not equal to</td></tr><tr><td>like</td><td>Like</td></tr><tr><td>not like</td><td>Not Like</td></tr><tr><td>></td><td>Greater than</td></tr><tr><td>>=</td><td>Greater than or equal to</td></tr><tr><td><</td><td>Less than</td></tr><tr><td><=</td><td>Less than or equal to</td></tr><tr><td>contains</td><td>Contains</td></tr><tr><td>does not contain</td><td>Doesn't contain</td></tr><tr><td>start with</td><td>Starts with</td></tr><tr><td>does not start with</td><td>Doesn't start with</td></tr><tr><td>is null</td><td>Is null</td></tr><tr><td>is not null</td><td>Is not null</td></tr></table>The possible logical operands are 'and' 'or'.</td><td>Yes</td></tr>
<tr><td>parameters</td><td>Delimited list of parameters that correspond to the numerical placeholders included in the criteria, separated by the delimiter |.| (a pipe sign followed by a dot and another pipe sign). As an example, to complement the value proposed for the criteria above, the value of the parameter attribute would be:
<code>P|.|SW</code></td><td>Yes, if the criteria contains at least one numeric placeholder.</td></tr>
<tr><td>case-sensitive</td><td>Specifies whether the query should be executed with case sensitivity or not. Possible values are 'true' and 'false'.</td><td>Yes</td></tr>
</table>

Parameters

This action does not use any parameters.

Options

No options of the objects in the request are used when calling this action.

Result

The result of this operation will be summarized in the PlmStatus object associated with the PlmResponse. The PlmData will contain a PlmObject for each object returned by the server as a result of executing the query.

Note The PlmObjects returned in the response will contain only the attributes defined in the PLM API for the entities, populated with the values currently stored in the server. No relations will be present in the response's data.

Samples

The following sample searches for parts whose number starts with 'P' and where the attribute 'creating-system' contains the letters "SW":

```
public Collection part_executequery(PlmSession session) {
        PlmObject plmObject;
        try {
            plmObject = PlmFactory.createObject("savedquery");
            plmObject.setAttributeValue("name", "TestQuery");
            plmObject.setAttributeValue("entity", "part");
            plmObject.setAttributeValue("criteria", "[number] start with %0 and
[creating-system] contains %1");
            plmObject.setAttributeValue("parameters", "P|.|SW");
            plmObject.setAttributeValue("case-sensitive", "true");
            PlmRequest request = PlmFactory.createRequest();
            PlmData data = request.getData();
            data.addObject(plmObject);
         PlmResponse response = session.execute(PlmActionConstants.EXECUTE_QUERY,
 Request);
            PlmStatus status = response.getStatus();
            if (status.isError()) {
           System.out.println("Call to part.executequery failed. Error code = " +
status.getCode() + " " + status.getMessage());
            }
         //Return the collection of PlmObjects found.
            return (PlmObject) response.getData().getObjects();
        } catch (PlmException e) {
            e.printStackTrace();
        }
        return null;
    }
```

The following sample searches for changes where the change number starts with 'C':

```
public Collection change_executequery(PlmSession session) {
        PlmObject plmObject;
        try {
            plmObject = PlmFactory.createObject("savedquery");
            plmObject.setAttributeValue("name", "Experiment2");
            plmObject.setAttributeValue("entity", "change");
            plmObject.setAttributeValue("criteria", "[number] start with %0");
            plmObject.setAttributeValue("parameters", "C000");
            plmObject.setAttributeValue("case-sensitive", "true");
            PlmRequest request = PlmFactory.createRequest();
            PlmData data = request.getData();
            data.addObject(plmObject);
          PlmResponse response = session.execute(PlmActionConstants.EXECUTE_QUERY,
request);
            PlmStatus status = response.getStatus();
            if (status.isError()) {
               System.out.println("Call to change.executequery failed. Error code
= " +
status.getCode() + " " + status.getMessage());
            }
            // return collection of objects found
            return response.getData().getObjects();
        } catch (PlmException e) {
            e.printStackTrace();
        }
        return null;
 }
```

The following sample searches for a document that starts with "D0000" and has a description that contains the text "caddrwngName".

```
public Collection document_executequery(PlmSession session) {
        PlmObject plmObject;
        try {
            plmObject = PlmFactory.createObject("savedquery");
            plmObject.setAttributeValue("name", "Experiment3");
            plmObject.setAttributeValue("entity", "document");
            plmObject.setAttributeValue("criteria", "[number] start with %0 and
[description] contains %1");
            plmObject.setAttributeValue("parameters", "D0000|.|caddrwngName");
            plmObject.setAttributeValue("case-sensitive", "true");
            PlmRequest request = PlmFactory.createRequest();
            PlmData data = request.getData();
            data.addObject(plmObject);
          PlmResponse response = session.execute(PlmActionConstants.EXECUTE_QUERY,
request);
            PlmStatus status = response.getStatus();
            if (status.isError()) {
                System.out.println("Call to document.executequery failed. Error
code = " +
status.getCode() + " " + status.getMessage());
            }
            //return collection of objects found
            return response.getData().getObjects();
        } catch (PlmException e) {
            e.printStackTrace();
        }
        return null;
    }
```

The following sample creates a query for a folder that starts with a certain folder number and its description contains a certain keyword we want to look for.

```
public Collection filefolder_executequery(PlmSession session) {
        PlmObject plmObject;
        try {
            plmObject = PlmFactory.createObject("savedquery");
            plmObject.setAttributeValue("name", "Experiment4");
            plmObject.setAttributeValue("entity", "filefolder");
            plmObject.setAttributeValue("criteria", "[number] start with %0 and
[description] contains %1");
            plmObject.setAttributeValue("parameters", "FOLDER0000|.|test");
            plmObject.setAttributeValue("case-sensitive", "true");
            PlmRequest request = PlmFactory.createRequest();
            PlmData data = request.getData();
            data.addObject(plmObject);
          PlmResponse response = session.execute(PlmActionConstants.EXECUTE_QUERY,
request);
            PlmStatus status = response.getStatus();
            if (status.isError()) {
                System.out.println("Call to filefolder.executequery failed");
            }
            //return collection of objects found
            return response.getData().getObjects();
        } catch (PlmException e) {
            e.printStackTrace();
        }
        return null;
    }
```

find

This function performs a query based on the attributes of the PlmObject given in the request and returns all objects of class type ‘part’ or ‘document’as a result of this query. The value of this constant is “find”.

Action Constant

com.agile.plmapi.connector.sdk.service. ItemFindService. findObject (PlmContext context, PlmRequest request, String entity, Integer baseClassId)

Action Entities

part, document

Input

The request data must contain at least one PlmObject of class type ‘part’ or ‘document’. Each of these PlmObjects will result in a query being executed and the objects returned by the query to be added to the response data, so if more than one object is present in the request, then the result will contain the union of the queries results.

Parameters

This function does not use any parameters.

Options

No options of the objects in the request are used when calling this function.

Result

The result of this operation will be summarized in the PlmStatus object associated with the PlmResponse. The PlmData will contain a PlmObject for each object returned by the server as a result of running the query corresponding to each object in the request.

Note The query criteria will be constructed using the Equal To operand ('==') for each attribute of the PlmObject with a non-empty value and using the 'and' logical operator to concatenate the conditions, therefore, the query will return all objects which exactly match all attributes specified for the PlmObject. Since the Equal To operand ('==') is used, it is worth mentioning that wildcards will not work with this function. Use the executequery function if you want to use wildcards.

The PlmObjects returned in the response will contain only the attributes defined in the PLM API for the entities, populated with the values currently stored in the server. No relations will be present in the response's data.

Samples

The following sample searches for part objects where the number starts with "P000" and where the value of the shippable-item attribute is "No":

```
public PlmObject part_findobject(PlmSession session) {
        PlmObject plmPart;
        try {
            plmPart = PlmFactory.createObject("part");
            plmPart.setAttributeValue("number", "P00001");
            // Additionally telling that the part should not be shippable
            plmPart.setAttributeValue("shippable-item","No");
            PlmRequest request = PlmFactory.createRequest();
            PlmData data = request.getData();
            data.addObject(plmPart);
          PlmResponse response = session.execute(PlmActionConstants.FIND, request);
            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError()) {
                System.out.println("find failed. Error code = " +
             status.getCode() + " " + status.getMessage());
            }
            data = response.getData();
            // The PlmObbject with values loaded in fields
            plmPart = (PlmObject) data.getObjects().iterator().next();
            // Load the attributes in a map
            Map m = plmPart.getAttributes();
            // Access some loaded attribute
            String value = plmPart.getAttributeValue("lifecycle-phase");
            return plmPart;
        } catch (PlmException e) {
            e.printStackTrace();
        }
        return null;
    }
```

The following sample also uses the document number and the description attribute of a document entity to locate it.

```
public PlmObject document_findobject(PlmSession session) {
        try {
            PlmObject plmDocument = PlmFactory.createObject("document");
            // Use the number value of a document that already exists on the PLM
            // server
            plmDocument.setAttributeValue("number", "D00002");
            plmDocument.setAttributeValue("description", "caddrwngName");
            PlmRequest request = PlmFactory.createRequest();
            request.getData().addObject(plmDocument);
          PlmResponse response = session.execute(PlmActionConstants.FIND, request);
            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError()) {
                System.out.println("document.findobject failed. Error code = " +
status.getCode() + " " + status.getMessage());
            }
            return (PlmObject) response.getData().getObjects().iterator().next();
        } catch (PlmException e) {
            e.printStackTrace();
        }
        return null;
    }
```

get-active-users

This function retrieves all the users defined in the PLM server, except those that have been set as inactive.

Action Constant

com.agile.plmapi.connector.sdk.service.UserDataService.getActiveUsers(PlmContext context, PlmRequest request)

Action Entities

userdata

Input

The request data can be empty.

Parameters

This function does not use any parameters.

Options

No options of the object in the request are used when calling this function.

Result

The result of this operation will be summarized in the PlmStatus object associated with the PlmResponse. The PlmData in the response will contain one PlmObject of type ‘user’, whose attributes ‘id’, ‘first-name’ and ‘last-name’ will be populated according to the values in the server. The value of the id attribute is the one to be used when executing functions that require a list of users’ ids, such as “part.getuserswithprivilege”, “change.sendtousers”, among others.

Samples

The following sample retrieves and prints a list of all users.

```
public void userdata_getactiveusers(PlmSession session) {
        try {
            PlmRequest request = PlmFactory.createRequest();
            PlmResponse response =
session.execute(PlmActionConstants.GET_ACTIVE_USERS, request);
            PlmStatus status = response.getStatus();
            if (status.isError()) {
                System.out.println("Call toget-active-users failed. "+
status.getCode() + " " + status.getMessage());
            }
            Collection topLevelCol = response.getData().getObjects();
            Iterator topLvliter = topLevelCol.iterator();
            while (topLvliter.hasNext()) {
                PlmObject userObj  = (PlmObject)topLvliter.next();
                System.out.println("The first name of the user is " +
userObj.getAttributeValue("first-name"));
                System.out.println("The last name of the user is "+
userObj.getAttributeValue("last-name"));
                System.out.println("The id of the user is "+
userObj.getAttributeValue("id"));
            }

        } catch (PlmException e) {
            e.printStackTrace();
        }
    }
```

get-affected-items

This function returns the affected items for an object of class type ‘change’.

Action Constant

com.agile.plmapi.connector.sdk.service.ChangeService.getItemsAffected(PlmContext context, PlmRequest request, Integer agileChangeClassId)

Action Entities

change

Input

The request data must contain atleast one PlmObject of entity type ‘change’. These PlmObject represent the ECO’s whose affected items will be retrieved.

Parameters

This function does not use any parameters.

Options

No options of the objects in the request are used when calling this function.

Result

The result of this operation will be summarized in the PlmStatus object associated with the PlmResponse. The PlmData will contain a PlmObject for each object present in the affected items table of the change.

Samples

This function reads the affected items of an existing change object:

```
public Collection change_affecteditems(PlmSession session) {
        PlmObject plmChange;
        try {
            plmChange = PlmFactory.createObject("change");
            // Use the number value of a change that already exists on the PLM
            // server
            plmChange.setAttributeValue("number", "C01021");
            PlmRequest request = PlmFactory.createRequest();
            PlmData data = request.getData();
            data.addObject(plmChange);
            PlmResponse response =
session.execute(PlmActionConstants.GET_AFFECTED_ITEM, request);
            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError())
                System.out.println("get affected items failed. Error code = " +
status.getCode() + " " + status.getMessage());
            data = response.getData();
            // The collection of returned affected items
            return data.getObjects();
        } catch (PlmException e) {
            e.printStackTrace();
        }
        return null;
    }
```

get-assigned-item

This function attempts to locate a part to which the object in the request is assigned in the context of the CAD integration and returns a PlmObject of entity type ‘part’ representing the parent part object if one is found, otherwise the function triggers an error. Tha value of this constant is “get-assigned-item”.

Action Constant

com.agile.ecp.plm.sdk.ObjectHandler.getParentItemAssignment(PlmContext context, PlmRequest request)

Action Entities

part, document

Input

The request data must contain only one PlmObject of entity type ‘part’ or ‘document’, which represents the object whose CAD assignment should be searched.

Parameters

This function does not use any parameters.

Options

PlmImplConstants.OPTION_CAD_CONFIGURATION: This option specifies the configuration of the component when assigned to the parent item. The default value of this option is "EMPTY", so if configurations are not being used in the CAD tool and the default value for configuration is other than EMPTY, the appropriate value should be specified in this option to allow the PLM API to find the assigned item.

Result

The result of this operation will be summarized in the PlmStatus object associated with the PlmResponse. If a part object was found, to which the object in the request is assigned in the context of the CAD integration, the PlmData will contain a PlmObject of type 'part' representing this object.

Samples

The following sample returns the item to which an existing part is assigned for the 'Default' CAD configuration:

```
public PlmObject part_getassigneditem(PlmSession session) {
        try {
            PlmObject part = PlmFactory.createObject("part");
            // Use the number value of a part that already exists on the PLM
            // server
            part.setAttributeValue("number", "P00175");
part.setOptionValue("cad-config", "Default");
            PlmRequest request = PlmFactory.createRequest();
            request.getData().addObject(part);
            PlmResponse response =
session.execute(PlmActionConstants.GET_ASSIGNED_ITEM, request);
            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError()) {
                System.out.println("get-assigned-item failed. Error code = " +
status.getCode() + " " + status.getMessage());
            }


            Collection col = response.getData().getObjects();
            if(col.size()==0)
                System.out.println("No assigned part was found");
            else
            {
                PlmObject resultPart = (PlmObject)col.iterator().next();
                String number = resultPart.getAttributeValue("number");
                System.out.println("Assigned part found with number: " + number);
                return resultPart;
            }
        } catch (PlmException e) {
            e.printStackTrace();
        }
        return null;
    }
```

This sample searches the given document for an assigned part type in the context of the CAD integration.

```
public PlmObject document_getassigneditem(PlmSession session) {
        try {
            PlmObject plmDocument = PlmFactory.createObject("document");
            // Use the number value of a document that already exists on the PLM
            // server
            plmDocument.setAttributeValue("number", "D05057");
            PlmRequest request = PlmFactory.createRequest();
            request.getData().addObject(plmDocument);
```

```
            PlmResponse response =
session.execute(PlmActionConstants.GET_ASSIGNED_ITEM, request);
            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError()) {
                System.out.println("get-assigned-item failed. Error code = " +
status.getCode() + " " + status.getMessage());
                return null;
            }
            return (PlmObject) response.getData().getObjects().iterator().next();
        } catch (PlmException e) {
            e.printStackTrace();
        }
        return null;
    }
```

get-cad-association

This function locates the CAD association object for each object in the request and returns the associated object. That is, for each object in the request, if the object represents a document in the a9 server, then the same object is returned. If, on the other hand, the object represents a part in the a9 server, the part's BOM table is scanned to locate a document, which corresponds to a CAD integration component, and the document representation is returned. In this latter case, the revision of the document is chosen based on the 'load-option' parameter as explained below.

Action Constant

com.agile.plmapi.connector.sdk.service. ItemService. getCADAssociation(PlmContext context, PlmRequest request)

Action Entities

part, document

Input

The request data must contain at least one PlmObject of entity type 'part' or 'document'. Each of these PlmObjects represents the object whose CAD association should be retrieved.

Parameters

PlmImplConstants.PARAMETER_LOAD_OPTION: This parameter specifies which revision of the associated document to consider when loading. The default value for this parameter is '0', so if this parameter is omitted or set to '0', the document will be retrieved based on its latest pending revision, if it's set to anything other that '0', then document(s) will be loaded based on the revision they had the last time that the cad association was saved.

Options

No options of the objects in the request are used when calling this function.

Result

The result of this operation will be summarized in the PlmStatus object associated with the PlmResponse. The PlmData will contain the CAD associated document for each object included in the request.

Samples

The following sample retrieves the cad association of an existing part:

```
public PlmObject part_getcadassociation(PlmSession session) {
```

```
        PlmObject plmPart;
        try {
            plmPart = PlmFactory.createObject("part");
            // Use the number value of a part that already exists on the PLM
            // server
            plmPart.setAttributeValue("number", "P00169");
            PlmRequest request = PlmFactory.createRequest();
            request.getHeader().setParameter("load-option", "1");
            PlmData data = request.getData();
            data.addObject(plmPart);
            PlmResponse response =
session.execute(PlmActionConstants.GET_CAD_ASSOCIATION, request);
            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError())
                System.out.println("get-cad-association failed. Error code = " +
                  status.getCode() + " " + status.getMessage());
            data = response.getData();
            // The returned PlmObject with complete structure
            plmPart = (PlmObject) data.getObjects().iterator().next();
            return plmPart;
        } catch (PlmException e) {
            e.printStackTrace();
        }
        return null;
    }
```

The following sample shows how to get the cad associations simply with a document object.

```
public PlmObject document_getcadassociation(PlmSession session)
    {
        try {
            PlmObject plmDocument;
            plmDocument = PlmFactory.createObject("document");
            // Use the doc number of a document that already exists on the PLM
            // server
            plmDocument.setAttributeValue("number", "D05041");
            PlmRequest plmRequest = PlmFactory.createRequest();
            plmRequest.getHeader().setParameter("load-option", "2");
            PlmData plmData = plmRequest.getData();
            plmData.addObject(plmDocument);
            PlmResponse response =
session.execute("PlmActionConstants.GET_CAD_ASSOCIATION, plmRequest);
            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError()) {
                System.out.println("get-cad-association failed. Error code = " +
status.getCode() + " " + status.getMessage());
            }
            plmDocument = (PlmObject)
response.getData().getObjects().iterator().next();
            return plmDocument;
        } catch (PlmException e) {
            e.printStackTrace();
        }
        return null;
    }
```

get-complete-structure

This function returns the complete structure of the given object(s) as currently stored in the PLM server, down to the lowest level of component relationships and including all the relations mapped in the server-side configuration file.

The behaviour is different for different types of entities present in the request.

For items the component relations are always retrieved recursively and other relations, such as History, Whereused, Pending Changes and Change History are included in the result for the top-level objects.

For objects of class change, the complete structure of the top-level change object(s) present in the request, as currently stored in the PLM server, including such relations as Affected Items, Attachments, Workflow, Reference and History is fetched.

For objects of class filefolders relations included are Files, References, Where Used, Workflow and History.

Lastly for object of class 'file', the relation between a file object and its containing FileFolder is fetched.

Action Constant

part, document :

com.agile.plmapi.connector.sdk.service. ItemService.getCompleteStructure(PlmContext context, PlmRequest request)

filefolder :

com.agile.plmapi.connector.sdk.service.DataObjectService.getCompleteStructure(PlmContext context, PlmRequest request)

change :

com.agile.plmapi.connector.sdk.service.DataObjectService.getCompleteStructure(PlmContext context, PlmRequest request)

file:

com.agile.plmapi.connector.sdk.service.FileService.getCompleteStructure(PlmContext context, PlmRequest request)

Action Entities

part, document, filefolder, change, file

Input

The request data must contain at least one PlmObject of entity class type 'part', 'document', 'change', 'filefolder' or 'file'. Each of these PlmObjects represents the plm object whose complete structure should be retrieved.

Parameters

PlmImplConstants.PARAMETER_LOAD_OPTION: This parameter is only valid for items (i.e. entities of class type 'part' and 'document') . This parameter specifies which revision of the components to consider when loading. The default value for this parameter is '0', so if this parameter is omitted or set to '0', the components will be loaded based on their latest pending revision, if it's set to '1', the components will be loaded based on their latest released revision. If it is set to '2', the components will be loaded based on the revision they had the last time that the structure was saved.

Options

No options of the objects in the request are used when calling this function.

Result

The result of this operation will be summarized in the PlmStatus object associated with the PlmResponse. The PlmData will contain each top-level object in the request, with its attributes and relations updated to show the complete structure as currently stored in the server. Note that the relations not only include the component relations (from the BOM table), but also other configured relations, such as History, Pending Changes, Change History, attachments, and so on depending on which entity the action is operating. For the "file" entity object only one relation of type 'file-container-relation', representing the relation between the file object and its containing FileFolder.

Note This function deletes all relations from the objects in the request before retrieving the structure, so any non top-level objects or any relations present in the request, which are not part of the structure of the top-level object(s), will not be present in the response.

Samples

The following sample reads the complete structure of an existing part:

```
public PlmObject part_getcompletestructure(PlmSession session) {
        PlmObject plmPart;
        try {
            plmPart = PlmFactory.createObject("part");
            // Use the number value of a part that already exists on the PLM
            // server
            plmPart.setAttributeValue("number", "P01766");
            PlmRequest request = PlmFactory.createRequest();
            request.getHeader().setParameter("load-option", "2");
            PlmData data = request.getData();
            data.addObject(plmPart);
            PlmResponse response = session.execute(
PlmActionConstants.GET_COMPLETE_STRUCTURE, request);
            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError())
             System.out.println("part.getcompletestructure failed. Error code = "+
                                    status.getCode() + " " + status.getMessage());
            data = response.getData();
            // The returned PlmObject with complete structure
            plmPart = (PlmObject) data.getObjects().iterator().next();
            return plmPart;
        } catch (PlmException e) {
            e.printStackTrace();
        }
        return null;
    }
```

The following sample reads the complete structure of a change object which exists in the server:

```
public PlmObject change_getcompletestructure(PlmSession session) {
        try {
            PlmObject plmChange = PlmFactory.createObject("change");
            // Use the number value of a change that already exists on the PLM
            // server
            plmChange.setAttributeValue("number", "C01021");
            PlmRequest plmRequest = PlmFactory.createRequest();
            PlmData plmData = plmRequest.getData();
            plmData.addObject(plmChange);
 PlmResponse response = session.execute(
PlmActionConstants.GET_COMPLETE_STRUCTURE, request);

            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError()) {
                System.out.println("get-complete-structure failed. Error code " +
```

```
status.getCode() + " " + status.getMessage());
            }
            plmChange = (PlmObject)
response.getData().getObjects().iterator().next();
            return plmChange;
        } catch (PlmException e) {
            e.printStackTrace();
        }
        return null;
    }
```

The following sample gets the complete structure of a given document.

```
public PlmObject document_getcompletestructure(PlmSession session) {
        try {
            PlmObject plmDocument;
            plmDocument = PlmFactory.createObject("document");
            // Use the doc number of a document that already exists on the PLM
            // server
            plmDocument.setAttributeValue("number", "D05041");
            PlmRequest plmRequest = PlmFactory.createRequest();
            plmRequest.getHeader().setParameter("load-option", "2");
            PlmData plmData = plmRequest.getData();
            plmData.addObject(plmDocument);
PlmResponse response = session.execute( PlmActionConstants.GET_COMPLETE_STRUCTURE,
request);

            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError()) {
                System.out.println("get-complete-structure failed. Error code = "
+ status.getCode() + " " + status.getMessage());
            }
            plmDocument = (PlmObject)
response.getData().getObjects().iterator().next();
            return plmDocument;
        } catch (PlmException e) {
            e.printStackTrace();
        }
        return null;
    }
```

The following sample retrieves the complete structure of the the file folder specified by the given number attribute.

```
public PlmObject filefolder_getcompletestructure(PlmSession session)
    {
        try {
            PlmObject plmFileFolder;
            plmFileFolder = PlmFactory.createObject("filefolder");
            // Use the File folder number that already exists on the PLM server
            plmFileFolder.setAttributeValue("number", "FOLDER00001");
            PlmRequest plmRequest = PlmFactory.createRequest();
            PlmData plmData = plmRequest.getData();
            plmData.addObject(plmFileFolder);
        PlmResponse response = session.execute(
PlmActionConstants.GET_COMPLETE_STRUCTURE, request);

            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError()) {
                System.out.println("get-complete-structure failed. Error code = "
+ status.getCode() + " " + status.getMessage());
            }
            plmFileFolder = (PlmObject)
response.getData().getObjects().iterator().next();
```

```
            return plmFileFolder;
        } catch (PlmException e) {
            e.printStackTrace();
        }
        return null;
    }
```

The following sample gets the relation of an existing file withs its container folder.

```
    public PlmObject file_getcompletestructure(PlmSession session) {
        try {
            PlmObject plmFile = PlmFactory.createObject("file");
            //Use the File folder number that already exists on the PLM server
            plmFile.setAttributeValue("folder-number", "FOLDER03021");
            PlmRequest plmRequest = PlmFactory.createRequest();
            PlmData plmData = plmRequest.getData();
            plmData.addObject(plmFile);
PlmResponse response = session.execute( PlmActionConstants.GET_COMPLETE_STRUCTURE,
request);
            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError()) {
              System.out.println("file.getcompletestructure failed. Error code " +
 status.getCode() + " " + status.getMessage());
            }
          plmFile = (PlmObject) response.getData().getObjects().iterator().next();
            return plmFile;
        } catch (PlmException e) {
            e.printStackTrace();
        }
        return null;
    }
```

get-files

This function retrieves the information of all the attached files for objects of type ‘document’ and returns it in the form of document-file-relations.

Action Constant

com.agile.plmapi.connector.sdk.service.DocumentFileRelationService. getFiles(PlmContext context, PlmRequest request)

Action Entities

document

Input

The request data must contain at least one PlmObject of entity type ‘document’. Each of these PlmObjects represents the document whose attachment information should be retrieved.

Parameters

This function does not use any parameters.

Options

No options of the objects in the request are used when calling this function.

Result

The result of this operation will be summarized in the PlmStatus object associated with the PlmResponse. The PlmData will contain each top-level object in the request, with as many document-to-file relations as the number of files currently associated to the object in the server, each one representing one file.

Samples

The following sample retireves the document file relations for a given document.

```
public PlmObject document_getfiles(PlmSession session) {
        try {
            PlmObject plmDocument;
            plmDocument = PlmFactory.createObject("document");
            // Use the doc number of a document that already exists on the PLM
            // server
            plmDocument.setAttributeValue("number", "D05029");
            PlmRequest plmRequest = PlmFactory.createRequest();
            PlmData plmData = plmRequest.getData();
            plmData.addObject(plmDocument);
            PlmResponse response = session.execute(PlmActionConstants.GET_FILES,
plmRequest);
            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError()) {
                System.out.println("document.getfiles failed. Error code = " +
 status.getCode() + " " + status.getMessage());
            }
            plmDocument = (PlmObject)
response.getData().getObjects().iterator().next();
            return plmDocument;
        } catch (PlmException e) {
            e.printStackTrace();
        }
        return null;
    }
```

get-home-folder

This function retrieves the contents of the current user's home folder, which can contain subfolders and saved queries.

Action Constant

com.agile.plmapi.connector.sdk.service.UserDataService.getHomeFolder(PlmContext context, PlmRequest request)

Action Entities

userdata

Input

The request data can be empty.

Parameters

This function does not use any parameters.

Options

No options of the object in the request are used when calling this function.

Result

The result of this operation will be summarized in the PlmStatus object associated with the PlmResponse. The PlmData in the response will contain the contents of the current user's recently visited folder. The contents will be represented by a structure of PlmObjects of type 'folder' or 'savedquery', with PlmRelations of type 'content' binding them together. The top level object returned in the response will in this case be of type "folder" and will be the user's home folder. All other PlmObjects will eb children of this plmObject. Additionally they will all have the attribute "id" set on them which can be used for later manipulation such as for createfolder.

Samples

The following sample shows how to get the home folder of the currently logged in user and navigate throught it's id and its' children ids.

```
public PlmObject user_gethomefolder(PlmSession session) {
        try {
            PlmRequest plmRequest = PlmFactory.createRequest();
            PlmResponse response =
session.execute(PlmActionConstants.GET_HOME_FOLDER, plmRequest);
            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError()) {
                System.out.println("get-home-folder failed. Error code " +
status.getCode() + " " + status.getMessage());
            }
           //Get the top level object which should be the home folder in this case.
            Collection topLevelCol = response.getData().getObjects();
            Iterator topLvliter = topLevelCol.iterator();
            while (topLvliter.hasNext()) {
                PlmObject parentObj  = (PlmObject)topLvliter.next();
 System.out.println("The id of the home folder called " +
     parentObj.getAttributeValue("name") +" is " +
parentObj.getAttributeValue("id"));
              System.out.println("Its children nodes and their respective ids are
as follows :");
                // Get at all the children of the home folder.
                Collection relationCol = parentObj.getRelations();
                Iterator relIter = relationCol.iterator();
                while (relIter.hasNext()) {
                    PlmChild child = ((PlmRelation) relIter.next()).getChild();
                    System.out.println("The id of this " +
   child.getAttributeValue("name") + " is " +
   child.getAttributeValue("id"));
                }
            }
        } catch (PlmException e) {
            e.printStackTrace();
        }
        return null;
    }
```

get-next-autonumber

This function returns the next autonumber from a given autonumber source.

Action Constant

getNextAutoNumber(PlmContext context, PlmRequest request)

Action Entities

part, document

Input

The request data needs just an empty plm object whose next auto number is desired to be fetched. Apart from this, the number source id must also be set as an option of the object included in the request.

Parameters

This function uses no parameters.

Options

PlmImplConstants.OPTION_NUMBERSOURCE_ID: This parameter specifies the id of the autonumber source to be used to retrieve the next number. This must be set on the object included in the request.

Result

The result of this operation will be summarized in the PlmStatus object associated with the PlmResponse. The PlmData will be empty, and the next number retrieved from the server will be stored in the parameter PlmImplConstants.OPTION_NEXT_AUTONUMBER of the object in the response.

Samples

The following sample reads the next autonumber from a given subclass and autonumber source:

```
public void part_getNextAutoNumber(PlmSession session)
    {
        PlmRequest request = PlmFactory.createRequest();
     PlmObject plmPart = PlmFactory.createObject("part");
      //The number-source-id of part's autonumber, can be retrieved by loadmetadata
sample shown.
plmPart.setOptionValue(PlmImplConstants.OPTION_NUMBERSOURCE_ID, "12416");
PlmData plmData = plmRequest.getData();
        plmData.addObject(plmPart);

        PlmResponse response;
        try {
            response = session.execute(PlmActionConstants.GET_NEXT_AUTO_NUMBER,
request);
            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError())
                System.out.println("get-next-autonumber failed. Error code = " +
 status.getCode() + " " + status.getMessage());
      try {
 PlmObject result = (PlmObject)response.getData().getObjects().iterator().next();
System.out.println(result.getOptionValue(PlmImplConstants.OPTION_NEXT_AUTONUMBER)
;
      } catch (Exception ex) {
         System.out.println(get-next-autonumber failed with Exception :" +
ex.getMessage());
      }

            System.out.println(response.getHeader().getParameterValue("next-
number"));
```

```
        } catch (PlmException e) {
            e.printStackTrace();
        }
    }
```

get-options

This function loads a summary of options and attributes of a part object, which are helpful to the CAD integration. The list of options and attributes loaded include: if the object exists in the server (internal.found option), the object's number, latest revision and lifecycle-phase, whether the current user has privilege to modify the part, whether the current user has privilege to create an ECO for the part, the object's latest pending revision, the object's attachment's checkout user and date if it's checked-out.

Action Constant

com.agile.plmapi.connector.sdk.service. ItemGetOptionService. getObjectOptions(PlmContext context, PlmRequest request)

Action Entities

part, document

Input

The request data must contain at least one PlmObject of entity type 'part' or 'document'. Each of these PlmObjects represents the part whose options should be retrieved.

Parameters

This function does not use parameters.

Options

No options of the objects in the request are used when calling this function.

Result

The result of this operation will be summarized in the PlmStatus object associated with the PlmResponse. The PlmData will contain the objects in the request, updated to include the values for the options and attributes mentioned above.

Samples

The following sample reads the options of an existing part:

```
public PlmObject part_getoptions(PlmSession session) {
        try {
            PlmObject plmPart = PlmFactory.createObject("part");
            // Use the number value of a part that already exists on the PLM
            // server
            plmPart.setAttributeValue("number", "P00023");

            PlmRequest plmRequest = PlmFactory.createRequest();
            PlmData plmData = plmRequest.getData();
            plmData.addObject(plmPart);
           PlmResponse response = session.execute PlmActionConstants.GET_OPTIONS,
plmRequest);
            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError()) {
```

```
                System.out.println("get-options failed. Error code " +
status.getCode() + " " + status.getMessage());
            }
          plmPart = (PlmObject) response.getData().getObjects().iterator().next();

            //To get a map of all the options
            Map m = plmPart.getOptions();
            //Or some individual option names and their values are printed below
            PlmOption plmOption = plmPart.getOption("internal.found");
           System.out.println(plmOption.getName() + " : " + plmOption.getValue());
            plmOption = plmPart.getOption("internal.has-modify-privilege");
           System.out.println(plmOption.getName() + " : " + plmOption.getValue());
            plmOption = plmPart.getOption("internal.latest-pending-revision");
           System.out.println(plmOption.getName() + " : " + plmOption.getValue());
            plmOption = plmPart.getOption("eco-number");
           System.out.println(plmOption.getName() + " : " + plmOption.getValue());
            plmOption = plmPart.getOption("internal.new-revision");
           System.out.println(plmOption.getName() + " : " + plmOption.getValue());
            plmOption = plmPart.getOption("internal.new-lifecycle-phase");
           System.out.println(plmOption.getName() + " : " + plmOption.getValue());
            plmOption = plmPart.getOption("internal.has-privilege");
           System.out.println(plmOption.getName() + " : " + plmOption.getValue());
            plmOption = plmPart.getOption("internal.has-eco-privilege");
           System.out.println(plmOption.getName() + " : " + plmOption.getValue());
            plmOption = plmPart.getOption("cad-config");
           System.out.println(plmOption.getName() + " : " + plmOption.getValue());
            plmOption = plmPart.getOption("parent-object-number");
           System.out.println(plmOption.getName() + " : " + plmOption.getValue());
            return plmPart;
        } catch (PlmException e) {
            e.printStackTrace();
        }
        return null;
    }
```

The following sample retrieves the options of an existing document (D00003):

```
public PlmObject document_getoptions(PlmSession session) {
        try {
            PlmObject plmDocument = PlmFactory.createObject("document");
            // Use the doc number of a document that already exists on the PLM
            // server
            plmDocument.setAttributeValue("number", "D00003");

            PlmRequest plmRequest = PlmFactory.createRequest();
            PlmData plmData = plmRequest.getData();
            plmData.addObject(plmDocument);
           PlmResponse response = session.execute(PlmActionConstants.GET_OPTIONS,
plmRequest);
            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError()) {
                System.out.println("get-options failed. Error code = " +
 status.getCode() + " " + status.getMessage());
            }
            plmDocument = (PlmObject)
response.getData().getObjects().iterator().next();
            // To get a map of all the options to iterate through
            Map m = plmDocument.getOptions();
            return plmDocument;
        } catch (PlmException e) {
            e.printStackTrace();
        }
```

```
            return null;
        }
```

get-properties

This function retrieves the properties, options and tables of a one or more plm objects present in the request. The tables included in the response are configured in the server, and typically include Attachments, History, Pending Changes and Change History.

Action Constant

com.agile.plmapi.connector.sdk.service. ItemGetPropertiesService. getProperties (PlmContext context, PlmRequest request)

Action Entities

part, document

Input

The request data must contain at least one PlmObject of entity type 'part' or 'document'. Each of these PlmObjects represents the entity whose attributes and options should be retrieved.

Parameters

This function does not use parameters.

Options

No options of the objects in the request are used when calling this function.

Result

The result of this operation will be summarized in the PlmStatus object associated with the PlmResponse. The PlmData will contain the objects in the request, updated to include the values for all the attributes defined for the entity and mapped to fields in the server side, plus the options and attributes returned by the action GET_OPTIONS and relations corresponding to the tables configured in the server side. It is important to note that this function also removes any relations present in the request, as only the top-level objects with their attributes and options will be included in the response.

Samples

The following sample reads the properties of an existing part:

```
public PlmObject part_getproperties(PlmSession session) {
        try {
            PlmObject plmPart;
            plmPart = PlmFactory.createObject("part");
            // Use the number value of a part that already exists on the PLM
            // server
            plmPart.setAttributeValue("number", "P00023");

 PlmObject plmDocument;
            plmDocument = PlmFactory.createObject("document");
            // Use the doc number of a document that already exists on the PLM
            // server
            plmDocument.setAttributeValue("number", "D01023");
```

```
            PlmRequest plmRequest = PlmFactory.createRequest();
            PlmData plmData = plmRequest.getData();
            plmData.addObject(plmPart);
 plmData.addObject(plmDocument);
        PlmResponse response = session.execute(PlmActionConstants.GET_PROPERTIES,
plmRequest);
            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError()) {
                System.out.println("get-properties failed. Error code = " +
                                    status.getCode()+ " " + status.getMessage());
            }
          plmPart = (PlmObject) response.getData().getObjects().iterator().next();
            // To get a map of all the options
            Map m = plmPart.getOptions();
            // Access some individual option values as follows
            PlmOption plmOption = plmPart.getOption("internal.found");
           System.out.println(plmOption.getName() + " : " + plmOption.getValue());
            // Get a map of all the attributes
            m = plmPart.getAttributes();
            // Access some individual attribute as follows
            String value = plmPart.getAttributeValue("lifecycle-phase");
            return plmPart;
        } catch (PlmException e) {
            e.printStackTrace();
        }
        return null;
    }
```

get-recently-visited-folder

This function retrieves the contents of the current user's recently visited folder.

Action Constant

com.agile.plmapi.connector.sdk.service.UserDataService. getRecentlyVisitedFolder(PlmContext context, PlmRequest request)

Action Entities

userdata

Input

The request data can be empty.

Parameters

This function does not use any parameters.

Options

No options of the object in the request are used when calling this function.

Result

The result of this operation will be summarized in the PlmStatus object associated with the PlmResponse. The PlmData in the response will contain the contents of the current user's recently visited folder. The recently visited folder object will have the PlmImplConstants.INTERNAL_OPTION_IS_RECENTLY_VISITED_FOLDER option set to true. The rest of the contents will be represented by a structure of PlmObjects of type 'folder', 'savedquery', 'part' or 'document', with PlmRelations of type 'content' binding them together. The top level object returned in the response will in this case be of type "folder" with the value of the "name" attribute set to "Recent Visits". It will than have children PlmObjects which will have have an option called "internal.node-id" set for them. Both these ids are of interest to us here as will be shown in later functions such as deletefoldercontent.

Samples

The following sample shows how to get the folder type contents of the "Recently Visited Folder" and how to navigate to the ids of all of its children.

```
public void user_getrecentlyvisitedfolder(PlmSession session) {
        try {
            PlmRequest plmRequest = PlmFactory.createRequest();
            PlmResponse response =
session.execute(PlmActionConstants.GET_RECENTLY_VISITED_FOLDER,
plmRequest);
            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError()) {
             System.out.println("get-recently-visited-folder failed. Error code "
+ status.getCode() + " " + status.getMessage());
            }
           //Get the top level object which should be the recently visited folder
     //in this case.
            Collection topLevelCol = response.getData().getObjects();
            Iterator topLvliter = topLevelCol.iterator();
            while (topLvliter.hasNext()) {
                PlmObject parentObj  = (PlmObject)topLvliter.next();
                if(parentObj.getAttributeValue("name").equals("Recent Visits"))
                {
               System.out.println("The id of the \"Recently visited Folder\" is " +
parentObj.getAttributeValue("id"));
                  System.out.println("Its children nodes and their respective ids
are as follows :");
                    //Get at all the children of the recently visited folder.
                    Collection relationCol = parentObj.getRelations();
                    Iterator relIter = relationCol.iterator();
                    while (relIter.hasNext()) {
                        PlmChild child = ((PlmRelation)relIter.next()).getChild();
                         System.out.println("The id of this " +
child.getAttributeValue("type") + " is " +
child.getOptionValue("internal.node-id"));
                    }
                }
            }
        } catch (PlmException e) {
            e.printStackTrace();
        }
    }
```

get-revision-to-cad

This function determines the revision of the objects in the request corresponding to the value of the 'load-option' parameter and updates the 'revision' attribute of them.

Action Constant

com.agile.plmapi.connector.sdk.service. ItemGetRevisionToCadService. getRevisionToCad (PlmContext context, PlmRequest request)

Action Entities

part, document

Input

The request data must contain at least one PlmObject of entity type 'part'. Each of these PlmObjects represents the object for which the appropriate revision will be loaded, based on the value of the parameter 'load-option' as explained below.

Parameters

PlmImplConstants.PARAMETER_LOAD_OPTION: This parameter specifies how the revision of the objects will be determined. T he default value for this parameter is '0', so if this parameter is omitted or set to '0', the objects' revision attribute will be updated to reflect their latest pending revision, if it's set to '1', the objects' revision attribute will be updated to reflect their latest released revision. Note that for this function, setting this parameter's value to '2' has the same effect of omitting it or setting it to '0', that is, the objects' revision attribute will be updated to reflect their latest pending revision.

Options

No options of the objects in the request are used when calling this function.

Result

The result of this operation will be summarized in the PlmStatus object associated with the PlmResponse. The PlmData of the response will contain the same objects present in the request, with the 'revision' attribute updated to the correct revision according to the load option.

Samples

The following sample retrieves the latest pending revision of an existing part:

```
      public PlmObject part_getrevisiontocad(PlmSession session) {
          try {
              PlmObject part = PlmFactory.createObject("part");
              // Use the number value of a part that already exists on the PLM
              // server
              part.setAttributeValue("number", "P00169");
    PlmObject plmDocument = PlmFactory.createObject("document");
          // Use the number value of a document that already exists
                 // on the PLM server
                  plmDocument.setAttributeValue("number", "D05057");

              PlmRequest request = PlmFactory.createRequest();
              request.getHeader().setParameter("load-option", "2");
              request.getData().addObject(part);
              request.getData().addObject(plmDocument);
              PlmResponse response =
  session.execute(PlmActionConstants.GET_REVISION_TO_CAD, request);
              PlmStatus status = response.getStatus();
              if (status != null && response.getStatus().isError()) {
                 System.out.println("part.getrevisiontocad failed. Error code = " +
                                        status.getCode() + " " + status.getMessage());
```

```
            }
            PlmObject resultPart =
                   (PlmObject)response.getData().getObjects().iterator().next();
            return resultPart;
        } catch (PlmException e) {
            e.printStackTrace();
        }
        return null;
    }
```

get-structure

This function returns the structure of a given object, optionally down to the lowest level of component relationships, as currently stored in the PLM server.

Action Constant

com.agile.plmapi.connector.sdk.service. ItemService. getStructure(PlmContext context, PlmRequest request, boolean includeAllRelations)

Action Entities

part, document

Input

The request data must contain at least one PlmObject of entity type 'part' or 'document'. Each of these PlmObjects represents the part whose structure should be retrieved.

Parameters

PlmImplConstants.PARAMETER_RECURSE: This parameter specifies whether or not include the component's structure recursively in the response. By default, the PLM connector will NOT include the structure of the components recursively, so the way to get this information in the response is to set this parameter to 'true'.

PlmImplConstants.PARAMETER_LOAD_OPTION: This parameter specifies which revision of the components to consider when loading the structure and it has effect only if the recurse parameter is set to 'true'. The default value for this parameter is '0', so if this parameter is omitted or set to '0', the components will be loaded based on their latest pending revision, if it's set to '1', the components will be loaded based on their latest released revision. If it is set to '2', the components will be loaded based on the revision they had the last time that the structure was saved.

Options

No options of the objects in the request are used when calling this function.

Result

The result of this operation will be summarized in the PlmStatus object associated with the PlmResponse. The PlmData will contain each top-level object in the request, with its attributes and relations updated to show the structure as currently stored in the server.

Note This function deletes all relations from the objects in the request before retrieving the structure, so any non top-level objects or any relations present in the request, which are not part of the structure of the top-level object(s), will not be present in the response.

Samples

The following sample reads the structure of an existing part:

```
public PlmObject part_getstructure(PlmSession session) {
            PlmObject plmPart;
        try {
            plmPart = PlmFactory.createObject("part");
            // Use the number value of a part that already exists on the PLM
            // server
            plmPart.setAttributeValue("number", "P01766");
            PlmRequest request = PlmFactory.createRequest();
            request.getHeader().setParameter("recurse", "true");
            request.getHeader().setParameter("load-option", "0");
            PlmData data = request.getData();
            data.addObject(plmPart);
         PlmResponse response = session.execute(PlmActionConstants.GET_STRUCTURE,
request);
            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError())
                System.out.println("part.getstructure failed. Error code = " +
                      status.getCode() + " " + status.getMessage());
            data = response.getData();
            //The returned PlmObject with complete structure
            plmPart = (PlmObject) data.getObjects().iterator().next();
            return plmPart;
        } catch (PlmException e) {
            e.printStackTrace();
        }
        return null;
    }
```

The following sample recursively retrieves the structure of the specified document. The last revision of all compoents is considered.

```
public PlmObject document_getstructure(PlmSession session) {
        try {
            PlmObject plmDocument;
            plmDocument = PlmFactory.createObject("document");
            // Use the doc number of a document that already exists on the PLM
            // server
            plmDocument.setAttributeValue("number", "D05041");
            PlmRequest plmRequest = PlmFactory.createRequest();
            plmRequest.getHeader().setParameter("recurse", "true");
            plmRequest.getHeader().setParameter("load-option", "2");
            PlmData plmData = plmRequest.getData();
            plmData.addObject(plmDocument);
         PlmResponse response = session.execute(PlmActionConstants.GET_STRUCTURE,
plmRequest);
            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError()) {
               System.out.println("document.getstructure failed. Error code = " +
   status.getCode() + " " + status.getMessage());
            }
            plmDocument = (PlmObject)
response.getData().getObjects().iterator().next();
            return plmDocument;
        } catch (PlmException e) {
            e.printStackTrace();
        }
        return null;
    }
```

get-users-with-privilege

This function checks which of the users given in the request have discover privilege on the object present in the request's data and returns only the users with discover privilege.

Action Constant

part, document

com.agile.plmapi.connector.sdk.service.ItemService.getUsersWithDiscoverPrivilege(PlmContext context, String userIds, PlmObject object)

filefolder :

com.agile.plmapi.connector.sdk.service.FilefolderService.getUsersWithDiscoverPrivilege(PlmContext context, String userIds, PlmObject object)

change :

com.agile.plmapi.connector.sdk.change.ChangeGetUsersWithPrivilegeHandler.
getUsersWithPrivilege(PlmContext context, PlmRequest request)

Action Entities

change, document, filefolder, part

Input

The request data must contain only one PlmObject of class type 'part', 'document', 'filefolder' or 'change'. This PlmObject represents the object, which will be queried to determine which of the given users have discover privilege on it. The users to consider are identified by their ids in the parameter 'users-ids' as explained below.

Parameters

PlmImplConstants.PARAMETER_USERS_IDS: This parameter is required and allows to specify the users to consider when determining which ones have discover privilege on the part object. This parameter must contain a list of the users' ids, separated by the delimiter |.| (a pipe sign followed by a dot and by another pipe sign).

Options

No options of the object in the request are used when calling this function.

Result

The result of this operation will be summarized in the PlmStatus object associated with the PlmResponse. The parameter 'users-ids' in the response header will contain the delimited list of users' ids of the user with discover privilege, separated by the delimiter |.| (a pipe sign followed by a dot and another pipe sign). The PlmData in the response will be empty.

Samples

The following sample returns an array of the users that have discover privilege on an existing part:

```
    public String[] part_getuserswithprivilege(PlmSession session)
    {
        PlmObject plmPart;
        try {
            plmPart = PlmFactory.createObject("part");
            // Use the number value of a part that you want to
//get user privileges for
```

```
            plmPart.setAttributeValue("number", "P00023");
            PlmRequest request = PlmFactory.createRequest();
            //A list of valid user-ids can be retrieved with
 //userdata.getactiveusers callable
          request.getHeader().setParameter(PlmImplConstants.PARAMETER_USERS_IDS,
"6169343|.|704|.|6163545");
            PlmData data = request.getData();
            data.addObject(plmPart);
 PlmResponse response =
session.execute(PlmActionConstants.GET_USERS_WITH_PRIVILEGE, request);
            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError()) {
              System.out.println("get-users-with-privilege failed. Error code = "
+ status.getCode() + " " + status.getMessage());
            }
            String resultStr =
response.getHeader().getParameterValue(PlmActionConstants.GET_USERS_WITH_PRIVILEG
E);
            String[] allowed_user_ids = resultStr.split("\\|\\.\\|");
            return allowed_user_ids;
        } catch (PlmException e) {
            e.printStackTrace();
        }
        return null;
    }
```

get-where-used

This function retrieves the Where Used information for the object(s) present in the request data.

Action Constant

com.agile.plmapi.connector.sdk.service.ItemGetWhereUsedService. getWhereUsed (PlmContext context, PlmRequest request)

Action Entities

part, document :

Input

The request data must contain at least one PlmObject of class type ‘part’ or ‘document’. Each of these PlmObjects represents the object whose ‘Where Used’ information should be retrieved.

Parameters

PlmImplConstants.PARAMETER_LOAD_OPTION: This parameter specifies how the revision of the parent objects will be determined. The default value for this parameter is ‘0’, so if this parameter is omitted or set to ‘0’ or ‘2’, the parent objects’ revision attribute will be updated to reflect their latest pending revision. If it’s set to ‘1’, the parent objects’ revision attribute will be updated to reflect their latest released revision.

Options

No options of the objects in the request are used when calling this function.

Result

The result of this operation will be summarized in the PlmStatus object associated with the PlmResponse. The PlmData will contain the same top-level objects included in the request; each of these objects will have a relation of type 'item-where-used-relation' for each item in which it's used as currently stored in the PLM server.

Note Since the response's data will contain only relations of type 'item-where-used-relation' for the top-level objects in the request, any other relations, as well as any other non-top-level objects present in the request will NOT be included in the response's data.

Samples

The following sample retrieves the where used information of two parts that exist in the server:

```
private void part_getwhereused(PlmSession session) {
        PlmObject plmPart1;
        PlmObject plmDocument1;
        try {
            plmPart1 = PlmFactory.createObject("part");
            // Use the number value of a part that already exists on the PLM
            // server
            plmPart1.setAttributeValue("number", "P00169");
            plmDocument1 = PlmFactory.createObject("document");
// Use the number value of a document that already exists on the PLM
               // server
                plmDocument1.setAttributeValue("number", "D00108");

            PlmObject plmPart2 = PlmFactory.createObject("part");
            // Use the number value of a part that already exists on the PLM
            // server
            plmPart1.setAttributeValue("number", "P00175");
            PlmRequest plmRequest = PlmFactory.createRequest();
            PlmData data = plmRequest.getData();
            data.addObject(plmPart1);
            data.addObject(plmPart2);
         PlmResponse response = session.execute(PlmActionConstants.GET_WHERE_USED,
plmRequest);
            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError())
                System.out.println("get-whereused failed. Error code = " +
                                   status.getCode() + " " + status.getMessage());

            Iterator result = response.getData().getObjects().iterator();
            for (; result.hasNext();) {
                PlmObject sourceObj = (PlmObject) result.next();
                Iterator relIterator = sourceObj.getRelations().iterator();
                for (; relIterator.hasNext();) {
                    PlmRelation relation = (PlmRelation)relIterator.next();
                    System.out.println("Plm Object no. "+
                              relation.getParent().getAttributeValue("number") +
                               " is used by item no. " +
                               relation.getAttributeValue("parent-number") + ".");
                }
            }
        } catch (PlmException e) {
            e.printStackTrace();
        }

    }
```

is-user-assigned

This function checks if the currently logged-in user is assigned to the change or filefolder class type object sent in the request, either as an approver or as an observer and returns 'true' or 'false' in the response's header's "user-is-assigned" parameter.

Action Constant

change:

com.agile.plmapi.connector.sdk.change. ChangeIsUserAssignedHandler. isUserAssigned (PlmContext context, PlmRequest request)

filefolder:

com.agile.plmapi.connector.sdk.filefolder.FilefolderIsUserAssignedHandler. isUserAssigned (PlmContext context, PlmRequest request)

Action Entities

change, filefolder

Input

The request data must contain only one PlmObject of class type 'change' or 'filefolder' representing the change to verify for the current user assignment .

Parameters

This function does not use any parameters.

Options

No options of the objects in the request are used when calling this function.

Result

The result of this operation will be summarized in the PlmStatus object associated with the PlmResponse. The parameter "user-is-assigned" in the response's header will contain either 'true' or 'false' depending on whether the current user is assigned to the given change or not. The PlmData will be empty.

Samples

The following sample checks if the current user is assigned to an existing change object:

```
public String change_isuserassigned(PlmSession session) {
        try {
            PlmObject plmChange = PlmFactory.createObject("change");
            // Use the number value of a change that already exists on the PLM
            // server
            plmChange.setAttributeValue("number", "C01021");
            PlmRequest plmRequest = PlmFactory.createRequest();
            PlmData plmData = plmRequest.getData();
            plmData.addObject(plmChange);
            PlmResponse response =
session.execute(PlmActionConstants.IS_USER_ASSIGNED,
 plmRequest);
            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError()) {
                System.out.println("is user assigned failed. Error code " +
```

```
status.getCode() + " " + status.getMessage());
            }
            return response.getHeader().getParameterValue("user-is-assigned");
        } catch (PlmException e) {
            e.printStackTrace();
        }
        return null;
    }
```

The following sample checks if the current user is assigned to an existing file folder object:

```
public String filefolder_isuserassigned(PlmSession session) {
        try {
            PlmObject plmFileFolder = PlmFactory.createObject("filefolder");
            //Use the File folder number that already exists on the PLM server
            plmFileFolder.setAttributeValue("number", "FOLDER03021");
            PlmRequest plmRequest = PlmFactory.createRequest();
            PlmData plmData = plmRequest.getData();
            plmData.addObject(plmFileFolder);
            PlmResponse response =
session.execute(PlmActionConstants.IS_USER_ASSIGNED,
plmRequest);
            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError()) {
                System.out.println("is user assigned failed. Error code " +
status.getCode() + " " + status.getMessage());
            }
            return response.getHeader().getParameterValue("user-is-assigned");
        } catch (PlmException e) {
            e.printStackTrace();
        }
        return null;
    }
```

load

This function reads the attributes of the part(s) or document(s) included in the request. Note that the returned object(s) include only the object's attributes and no relations; in fact, any relations included in the request will not be present in the response.

Action Constant

com.agile.plmapi.connector.sdk.service.DataObjectService. loadObjects(PlmContext context, PlmRequest request)

Action Entities

part, document

Input

The request data must contain at least one PlmObject of entity type 'part' or 'document'. Each of these PlmObjects represents the part or document whose attributes should be retrieved.

Parameters

This function does not use parameters.

Options

No options of the objects in the request are used when calling this function.

Result

The result of this operation will be summarized in the PlmStatus object associated with the PlmResponse. The PlmData will contain one object for each top-level object included in the request, whose attributes will be filled in.

Samples

The following code attempts to load an existing part:

```
public PlmObject part_load(PlmSession session) {
        PlmObject plmPart;
        try {
            plmPart = PlmFactory.createObject("part");
            //Use the number value of a part that already exists on the PLM server
            plmPart.setAttributeValue("number", "P01335");
            PlmRequest request = PlmFactory.createRequest();
            PlmData data = request.getData();
            data.addObject(plmPart);
         PlmResponse response = session.execute(PlmActionConstants.LOAD, request);
            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError()) {
                System.out.println("load failed. Error code = "
                        + status.getCode() + " "
                        + status.getMessage());
            }
            data = response.getData();
            // The PlmObbject with values loaded in fields
            plmPart = (PlmObject) data.getObjects().iterator().next();
            // Load the attributes in a map
            Map m = plmPart.getAttributes();
            // Access some loaded attribute
            String value = plmPart.getAttributeValue("lifecycle-phase");
            return plmPart;
        } catch (PlmException e) {
            e.printStackTrace();
        }
                return null;
    }
```

load-meta-data

This function loads the metadata of the class type entities part, document, change and filefolder and optionally of the given entity of the same class type if one was included in the request. The metadata information of the entity includes the subclasses that have been configured for it and the autonumber sources configured for each subclass, as well as all the lists of possible values for the attributes of the entity, which have been configured as single-selection or multiple-selection. For an item object included in the request, then the metadata information includes the list of revisions of the corresponding object in the PLM server, as well as the list of valid sites for it. For a change class type object included in the request, the metadata information includes the list of valid next status for the change. For a filefolder class type object included in the request, then the metadata information includes the list of versions of the corresponding object in the PLM server.

Action Constant

com.agile.plmapi.connector.sdk.service. ItemLoadMetaDataService. (PlmContext context, PlmRequest request, Integer classId)

filefolder :

com.agile.plmapi.connector.sdk.filefolder. loadMetaData(PlmContext context, PlmRequest request)

change :

com.agile.plmapi.connector.sdk.change. loadMetaData(PlmContext context, PlmRequest request)

Action Entities

part, change, document, filefolder

Input

Request data might be empty or contain one part object whose metadata should be retrieved. For convenience purposes, retrieving the metadata for an entity or an object can be achieved by executing one of the following methods:

com.agile.plmapi.api.PlmSession.getMetaData(PlmEntity entity);

com.agile.plmapi.api.PlmSession.getMetaData(PlmObject object);

Parameters

PlmImplConstants.SEARCH_ENTITY_TYPE: This parameter must be set to name of the entity whose metadata is required to be fetched. Meta data for only one entity at a time can be loaded.

Options

No options of the objects in the request are used when calling this function.

Result

The result will be returned by either of the functions mentioned above as a PlmMetaDataEntry object. This object represents a hierarchical arrangement of metadata entries, which can be navigated in a tree-like structure using the function PlmMetaDataEntry.getEntries(String entryName)

Samples

The following sample can be used to navigate the metadata structure to retrieve the list of subclasses and autonumber sources associated with the part entity.

```
        PlmEntity entity = session.getDefinition().getEntity("part");
        PlmMetaDataEntry root = session.getMetaData(entity);
        Collection subclasses = root.getEntries("subclasses");
        Iterator subclassesIt = subclasses.iterator();
        while (subclassesIt.hasNext()) {
           PlmMetaDataEntry subclassEntry = (PlmMetaDataEntry) subclassesIt.next();
           System.out.println("Subclass: '" + subclassEntry.getName() + "' ID: " +
subclassEntry.getValue());
           System.out.println("Autonumbers:");
           Collection autonumbers = subclassEntry.getEntries("autonumbers");
           Iterator autonumberIt = autonumbers.iterator();
           while (autonumberIt.hasNext()) {
              PlmMetaDataEntry autonumber = (PlmMetaDataEntry) autonumberIt.next();
              System.out.println("Name: " + autonumber.getName() + ", ID: " +
autonumber.getValue());
           }
        }
```

The following sample can be used to retrieve the revisions of a part object:

```
PlmObject part = PlmFactory.createObject("part");
part.setAttributeValue("number", "P00501");
PlmMetaDataEntry root = session.getMetaData(part);
Collection<PlmMetaDataEntry> revisions = root.getEntries("revisions");
System.out.println("Revisions for part 'P00501':");
for (PlmMetaDataEntry revision : revisions) {
System.out.println(revision.getName());
}
```

Samples

The following sample can be used to navigate the metadata structure to retrieve the list of subclasses and autonumber sources associated with the document entity.

```
PlmEntity entity = session.getDefinition().getEntity("document");
PlmMetaDataEntry root = session.getMetaData(entity);
Collection subclasses = root.getEntries("subclasses");
Iterator iter = subclasses.iterator();
while (iter.hasNext()) {
    PlmMetaDataEntry subclassEntry = (PlmMetaDataEntry) iter.next();
    System.out.println("Subclass: '" + subclassEntry.getName()
                    + "' ID: " + subclassEntry.getValue());
    System.out.println("Autonumbers:");
    Collection<PlmMetaDataEntry> autonumbers = subclassEntry
                    .getEntries("autonumbers");
    for (PlmMetaDataEntry autonumber : autonumbers) {
        System.out.println("Name: " + autonumber.getName() + ", ID: "
                        + autonumber.getValue());
    }
}
```

The following sample can be used to retrieve the revisions of a document object:

```
PlmObject part = PlmFactory.createObject("document");
part.setAttributeValue("number", "D00827");
PlmMetaDataEntry root = session.getMetaData(part);
Collection<PlmMetaDataEntry> revisions = root.getEntries("revisions");
System.out.println("Revisions for document 'D00827':");
for (PlmMetaDataEntry revision : revisions) {
    System.out.println(revision.getName());
}
```

The following sample can be used to navigate the metadata structure to retrieve the list of subclasses and autonumber sources associated with the change entity.

```
PlmEntity entity = session.getDefinition().getEntity("change");
PlmMetaDataEntry root = session.getMetaData(entity);
Collection subclasses = root.getEntries("subclasses");
Iterator iter = subclasses.iterator();
while (iter.hasNext()) {
    PlmMetaDataEntry subclassEntry = (PlmMetaDataEntry) iter.next();
    System.out.println("Subclass: '" + subclassEntry.getName()
                    + "' ID: " + subclassEntry.getValue());
    System.out.println("Autonumbers:");
    Collection<PlmMetaDataEntry> autonumbers = subclassEntry
                    .getEntries("autonumbers");
    for (PlmMetaDataEntry autonumber : autonumbers) {
        System.out.println("Name: " + autonumber.getName() + ", ID: "
                        + autonumber.getValue());
    }
}
```

The following sample can be used to retrieve the valid next statuses of a change object:

```
PlmObject change = PlmFactory.createObject("change");
change.setAttributeValue("number", "C00261");
PlmMetaDataEntry root = session.getMetaData(change);
Collection<PlmMetaDataEntry> entries = root.getEntries("lists/nextstatuses");
System.out.println("Valid next status for change 'C00261':");
for (PlmMetaDataEntry status : entries) {
    System.out.println(status.getName());
}
```

The following sample can be used to navigate the metadata structure to retrieve the list of subclasses and autonumber sources associated with the filefolder entity.

```
PlmEntity entity = session.getDefinition().getEntity("filefolder");
PlmMetaDataEntry root = session.getMetaData(entity);
Collection subclasses = root.getEntries("subclasses");
Iterator iter = subclasses.iterator();
while (iter.hasNext()) {
    PlmMetaDataEntry subclassEntry = (PlmMetaDataEntry) iter.next();
    System.out.println("Subclass: '" + subclassEntry.getName()
                    + "' ID: " + subclassEntry.getValue());
    System.out.println("Autonumbers:");
    Collection<PlmMetaDataEntry> autonumbers = subclassEntry
                    .getEntries("autonumbers");
    for (PlmMetaDataEntry autonumber : autonumbers) {
        System.out.println("Name: " + autonumber.getName() + ", ID: "
                        + autonumber.getValue());
    }
}
```

The following sample can be used to retrieve the versions of a filefolder object:

```
PlmObject ffolder = PlmFactory.createObject("filefolder");
ffolder.setAttributeValue("number", "FOLDER01522");
PlmMetaDataEntry root = session.getMetaData(ffolder);
Collection<PlmMetaDataEntry> versions = root.getEntries("versions");
System.out.println("Versions for file folder 'FOLDER01522':");
for (PlmMetaDataEntry version : versions) {
    System.out.println(version.getName());
}
```

locate-files

This function scans the request for objects of class type ‘document’. For each of these objects it scans the relations of type document-to-file and populates several attributes and options of the associated file represented by each relation, including the file’s id, which can be used to download the file from the FileServer.

Action Constant

com.agile.plmapi.connector.sdk.service.DocumentFileRelationService.locateFiles(PlmContext context, PlmRequest request)

Action Entities

document

Input

The request data must contain at least one top-level object of entity type ‘document’, which should have at least one child object of type ‘file’, with a document-to-file relation binding them. The relation’s ‘file-name’ attribute must be populated with the name of the file.

Parameters

This function does not use any parameters.

Options

No options of the objects in the request are used when calling this function.

Result

The result of this operation will be summarized in the PlmStatus object associated with the PlmResponse. The PlmData will contain the same objects and relations specified in the request; however, for each relation of type 'document-to-file', the following attributes and options will have been updated:

Attributes:

- file-id Set to the id of the file in the FileServer.
- cad-last-view-date Set to the timestamp of the last version of the file which was associated to the document.

Options:

- internal.found Set to 'true' or 'false' depending on whether a matching associated file was found or not.
- internal.checkedout Set to 'true' or 'false' depending on whether the associated file is currently checked-out or not.
- internal.check-out-user Set to the name of the user who checked-out the associated file if applicable.
- internal.checked-out-by-other-user If the user who checked-out the associated file is not the same as the currently logged-in user.

Samples

The following sample retrieves the information of the attached file for a given document.

```
public PlmObject document_locatefiles(PlmSession session) {
        try {
            PlmObject plmDocument;
            plmDocument = PlmFactory.createObject("document");
            // Use the doc number of a document that already exists on the PLM
            // server
            plmDocument.setAttributeValue("number", "D00002");
            PlmObject plmFile = PlmFactory.createObject("file");
            PlmRelation plmRelation = plmDocument.createRelation("document-file-
relation", plmFile);
            plmRelation.setAttributeValue("file-name", "1.txt");
            PlmRequest plmRequest = PlmFactory.createRequest();
            PlmData plmData = plmRequest.getData();
            plmData.addObject(plmDocument);
            PlmResponse response =
session.execute(PlmActionConstants.ACTION_LOCATE_FILES,
                    plmRequest);
            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError()) {
                System.out.println("document.locatefiles failed. Error code = "
                        + status.getCode() + " " + status.getMessage());
            }
            plmDocument = (PlmObject) response.getData().getObjects()
                    .iterator().next();
            return plmDocument;
        } catch (PlmException e) {
            e.printStackTrace();
        }
```

```
            return null;
        }
```

quick-search

This function performs a quick search based on a text value and returns all objects of entity type 'part' which result of the quick search. The quick search functionality itself is determined by the PLM server.

Action Constant

com.agile.plmapi.connector.sdk.service.QueryService.quickSearch(PlmContext context, PlmRequest request, int[] quickSearchtypes)

and

com.agile.plmapi.connector.sdk.callables.QuickSearchCallable. call(…)

Action Entities

part.quicksearch, document. quicksearch, change. quicksearch, filefolder. quicksearch.

Input

The request data needs not to contain any PlmObject, however, the search-text parameter and the search entity type mentioned below are required for this function.

Parameters

PlmImplConstants.SEARCH_ENTITY_TYPE: This parameter specifies the entities to be searched. This parameter can be added multilple times for each entity that needs to be searched.

PlmImplConstants.PARAMETER_SEARCH_TEXT: This parameter specifies the text to use for the quick search.

Options

No options of the objects in the request are used when calling this function.

Result

The result of this operation will be summarized in the PlmStatus object associated with the PlmResponse. The PlmData will contain a PlmObject for each object returned by the server as a result of the quick search function.

Note The PlmObjects returned in the response will contain only the attributes defined in the PLM API for the entity, populated with the values currently stored in the server. No relations will be present in the response's data.

Samples

The following sample performs a quick search using the text 'P001*', that is, of all parts whose number or description begin with 'P001':

```
private Collection quicksearch(PlmSession session) {
        try {
            PlmRequest request = PlmFactory.createRequest();
         request.getHeader().setParameter(PlmImplConstants.PARAMETER_SEARCH_TEXT,
"P00169");
            request.getHeader().setParameter(PlmImplConstants.SEARCH_ENTITY_TYPE,
"part");
```

```
            PlmResponse response = session.execute(PlmActionConstants.QUICK_SEARCH,
request);
              PlmStatus status = response.getStatus();
              if (status != null && response.getStatus().isError()) {
                  System.out.println("quick-search failed. Error code = " +
status.getCode() + " " + status.getMessage());
              }
              //return collection of objects found
              return response.getData().getObjects();
          } catch (PlmException e) {
              e.printStackTrace();
          }
          return null;
      }
```

reject

This function rejects the change or filefolder object present in the request.

Action Constant

change:

com.agile.plmapi.connector.sdk.service.ChangeService.approveReject(PlmContext context, PlmRequest request, String action)

filefolder:

com.agile.plmapi.connector.sdk.service.FilefolderService. approveReject(PlmContext context, PlmRequest request, String action)

Action Entities

change, filefolder

Input

The request data may contain any number of toplevel PlmObjects of entity type 'change' or 'filefolder' which need to be rejected.

Parameters

PlmImplConstants.PARAMETER_USERS_IDS: This parameter is required and allows to specify the users to send notification when approving the change object. This parameter must contain a list of the users' ids, separated by the delimiter |.| (a pipe sign followed by a dot and by another pipe sign).

PlmImplConstants.PARAMETER_APPROVE_REJECT_PASSWORD: This parameter is required and allows to send an encrypted version of the password of the currently logged-in user, required to reject a change. The password needs to be encrypted using a BlowFish encrypting mechanism.

PlmImplConstants.PARAMETER_COMMENT: This parameter is optional and allows to specify a comment to include when rejecting the change.

Options

No options of the object in the request are used when calling this function.

Result

The result of this operation will be summarized in the PlmStatus object associated with the PlmResponse. The PlmData in the response will be empty.

Samples

The following sample rejects an existing change object:

```
public void change_reject(PlmSession session) {
        try {
            PlmObject plmChange = PlmFactory.createObject("change");
            plmChange.setAttributeValue("number", "C01223");
            PlmRequest plmRequest = PlmFactory.createRequest();
            //A list of valid user-ids can be retrieved with
 //userdata.getactiveusers callable
            plmRequest.getHeader().setParameter
PlmImplConstants.PARAMETER_USERS_IDS, "6169343|.|704");
          plmRequest.getHeader().setParameter PlmImplConstants.PARAMETER_COMMENT,
"Change seems ok.");
            plmRequest.getHeader().setParameter
PlmImplConstants.PARAMETER_APPROVE_REJECT_PASSWORD,
com.agile.plmapi.util.BlowFishEncrypter.encrypt("agile"));
            PlmData plmData = plmRequest.getData();
            plmData.addObject(plmChange);
            PlmResponse response = session.execute(PlmActionConstants.REJECT,
plmRequest);
            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError()) {
               System.out.println("reject failed. Error code " + status.getCode()
+ " " + status.getMessage());
            }
        } catch (PlmException e) {
            e.printStackTrace();
        }
    }
```

The following sample rejects the file folder given in the request.

```
public void filefolder_reject(PlmSession session) {
        try {
            PlmObject plmFileFolder = PlmFactory.createObject("filefolder");
            plmFileFolder.setAttributeValue("number", "FOLDER03021");
            PlmRequest plmRequest = PlmFactory.createRequest();
            //A list of valid user-ids can be retrieved with
     //userdata.getactiveusers callable
            plmRequest.getHeader().setParameter
PlmImplConstants.PARAMETER_USERS_IDS, "6169343|.|704");
          plmRequest.getHeader().setParameter PlmImplConstants.PARAMETER_COMMENT,
"Change seems ok.");
            plmRequest.getHeader().setParameter
PlmImplConstants.PARAMETER_APPROVE_REJECT_PASSWORD,
com.agile.plmapi.util.BlowFishEncrypter.encrypt("agile"));

            PlmData plmData = plmRequest.getData();
            plmData.addObject(plmFileFolder);
            PlmResponse response = session.execute(PlmActionConstants.REJECT,
plmRequest);
            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError()) {
                System.out.println("reject failed. Error code " +
status.getCode() + " " + status.getMessage());
            }
```

```
        } catch (PlmException e) {
            e.printStackTrace();
        }
    }
```

remove-affected-items

This function scans the request for objects of class type 'change'. For each of these objects, the function scans the relations, looking for relations of type "change-item-relation". For each of these relations, the corresponding row it points to in the 'Affected Items' table is deleted.

Action Constant

com.agile.plmapi.connector.sdk.change.ChangeRemoveAffectedItemsHandler.
deleteFromChange(PlmContext context, PlmRequest request)

Action Entities

change

Input

The request data must contain at least one PlmObject of entity type 'change'. E ach of these PlmObjects represents the ECO whose affected items will be deleted.

Parameters

This function does not use any parameters.

Options

No options of the objects in the request are used when calling this function.

Result

The result of this operation will be summarized in the PlmStatus object associated with the PlmResponse. If a warning was triggered while attempting to execute this function, then the response's status will be of type Warning and the status' code will contain the code of the warning that was triggered. The PlmData of the response will be empty.

Samples

The following change removes one part (P01470) form the affected items table of an existing change (C01021):

```
public void change_deletefromchange(PlmSession session) {
        try {
            PlmObject plmChange = PlmFactory.createObject("change");
            // Use the number value of a change that already exists on the PLM
            // server
            plmChange.setAttributeValue("number", "C01021");
            //Create reference to a changeitemrealtion already on the server
 //for this change
            PlmObject part = PlmFactory.createObject("part");
            part.setAttributeValue("number", "P01470");
            plmChange.createRelation("change-item-relation", part);
            PlmRequest plmRequest = PlmFactory.createRequest();
            PlmData plmData = plmRequest.getData();
            plmData.addObject(plmChange);
            plmData.addObject(plmChange);
```

```
            //Now delete this change-item-relation
            PlmResponse response =
session.execute(PlmActionConstants.REMOVE_AFFECTED_ITEMS,
 plmRequest);
            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError()) {
                System.out.println("remove affected items failed. Error code = " +
  status.getCode() + " " + status.getMessage());
            }
        } catch (PlmException e) {
            e.printStackTrace();
        }
    }
```

rename-folder/rename_query

This function renames a folder or saved query object (respectively according to the action name), which is part of the structure of the current user's recently visited folder or home folder.

Action Constant

com.agile.plmapi.connector.sdk.service.UserDataService.renameObject(PlmContext context, PlmRequest request)

Action Entities

userdata

Input

The request data can be empty.

Parameters

PlmImplConstants.PARAMETER_OBJECT_ID: Contains the id of the object (either folder or saved query) to be renamed.

PlmImplConstants.PARAMETER_NEW_NAME: Contains the new name for the folder or saved query.

Options

No options of the object in the request are used when calling this function.

Result

The result of this operation will be summarized in the PlmStatus object associated with the PlmResponse. The PlmData in the response will be empty.

Samples

The following renames a saved query under a users Personal searches category located in his home folder.

```
public void userdata_rename(PlmSession session) {
        try {
            PlmRequest plmRequest = PlmFactory.createRequest();
            //Id of a saved query under the Personal Search category.
            //Can be retrieved through userdata.gethomefolder

plmRequest.getHeader().setParameter(PlmImplConstants.PARAMETER_OBJECT_ID,
"6136964");
```

```
          plmRequest.getHeader().setParameter(PlmImplConstants.PARAMETER_NEW_NAME,
"test rename query");
             PlmResponse response = session.execute("userdata.rename", plmRequest);
              PlmStatus status = response.getStatus();
              if (status != null && response.getStatus().isError()) {
                  System.out.println("userdata.rename failed. Error code " +
status.getCode() + " " + status.getMessage());
              }
         } catch (PlmException e) {
              e.printStackTrace();
         }
     }
```

save-query

This function saves a given query object inside a folder, which is part of the structure of the current user's recently visited folder or home folder.

Action Constant

com.agile.plmapi.connector.sdk.service.UserDataService. saveQuery(PlmContext context, PlmRequest request)

Action Entities

userdata

Input

The request data must have only one PlmObject of type 'savedquery', which represents the query to be saved. This object's attributes must be set according to the table below:

<table>
<tr><th>Attribute Name</th><th>Value</th><th>Required</th></tr>
<tr><td>name</td><td>(Name to give to the query when saved.</td><td>Yes</td></tr>
<tr><td>entity</td><td>The entity to query for, either "part", "document", "change" or "filefolder" as configured in the configuration file.</td><td>Yes</td></tr>
<tr><td>criteria</td><td>The criteria for the query, specified using the following syntax:
[attribute-name] operand [parameter-numerical-order] [logical-operand [attribute-name] operand [parameter-numerical-order]]
For example, a query to search for all parts where the number attribute starts with 'P' would be specified as follows (together with the value of the 'parameters' attribute for the savedquery PlmObject specified below):
<code>[number] start with %0 and [creating-system]contains %1</code>
Figure 2-2: List of possible operands
<table><tr><th>Operand¤</th><th>Description¤</th></tr><tr><td>==¤</td><td>Equal·to¤</td></tr><tr><td>!=¤</td><td>Not·equal·to¤</td></tr><tr><td>like¤</td><td>Like¤</td></tr><tr><td>not·like¤</td><td>Not·Like¤</td></tr><tr><td>>¤</td><td>Greater·than¤</td></tr><tr><td>>=¤</td><td>Greater·than·or·equal·to¤</td></tr><tr><td><¤</td><td>Less·than¤</td></tr><tr><td><=¤</td><td>Less·than·or·equal·to¤</td></tr><tr><td>contains¤</td><td>Contains¤</td></tr><tr><td>does·not·contain¤</td><td>Doesn't·contain¤</td></tr><tr><td>start·with¤</td><td>Starts·with¤</td></tr><tr><td>does·not·start·with¤</td><td>Doesn't·start·with¤</td></tr><tr><td>is·null¤</td><td>Is·null¤</td></tr><tr><td>is·not·null¤</td><td>Is·not·null¤</td></tr></table>The possible logical operands are: 'and' 'or'.</td><td>Yes</td></tr>
<tr><td>parameters</td><td>Delimited list of parameters which correspond to the numerical placeholders included in the criteria, separated by the delimiter |.| (a pipe sign followed by a dot and another pipe sign). As an example, to complement the value proposed for the criteria above, the value of the parameter attribute would be:
P|.|SW</td><td>Yes, if the criteria contains at least one numeric placeholder.</td></tr>
<tr><td>case-sensitive</td><td>Specifies whether the query should be executed with case sensitivity or not. Possible values are 'true' and 'false'.</td><td>Yes</td></tr>
</table>

Parameters

PlmImplConstants.PARAMETER_PARENT_FOLDER: Contains the id folder within which the query will be saved.

Options

No options of the object in the request are used when calling this function.

Result

The result of this operation will be summarized in the PlmStatus object associated with the PlmResponse. The PlmData in the response will be empty.

Samples

The following sample creates a saved query under the Personal Searches folder.

```
public void userdata_savequery(PlmSession session) {
        PlmObject plmObject;
        try {
            plmObject = PlmFactory.createObject("savedquery");
            plmObject.setAttributeValue("name", "Experiment1");
            plmObject.setAttributeValue("entity", "part");
            plmObject.setAttributeValue("criteria", "[number] start with %0 and
[description] contains %1");
            plmObject.setAttributeValue("parameters", "P|.|test");
            plmObject.setAttributeValue("case-sensitive", "true");
            PlmRequest request = PlmFactory.createRequest();
           //parent-folder is specified as the id of the Personal Search category.
            //Can be retrieved through userdata.gethomefolder

request.getHeader().setParameter(PlmImplConstants.PARAMETER_PARENT_FOLDER,
"6084430");
            PlmData data = request.getData();
            data.addObject(plmObject);
           PlmResponse response = session.execute(PlmActionConstants.SAVE_QUERY,
request);
            PlmStatus status = response.getStatus();
            if (status.isError()) {
               System.out.println("Call to save_query failed. "+ status.getCode()
+ " " + status.getMessage());
            }
        } catch (PlmException e) {
            e.printStackTrace();
        }
    }
```

sent-to-users

This function sends the object present in the request to the given users, optionally including a comment. The send functionality is server specific.

Action Constant

part, document:

> com.agile.plmapi.connector.sdk.service. ItemSendToUsersService. sendToUsers (PlmContext context, PlmRequest request)

filefolder:

com.agile.plmapi.connector.sdk.filefolder. FilefolderSendToUsersHandler. sendToUsers(PlmContext context, PlmRequest request)

change:

com.agile.plmapi.connector.sdk.change. ChangeSendToUsersHandler sendToUsers(PlmContext context, PlmRequest request)

Action Entities

part, change, document, filefolder

Input

The request data may have only one PlmObject of class type 'part', 'document', 'change' or 'filefolder'. This PlmObject represents the object which will be sent to the users identified by their ids in the parameter 'users-ids' as explained below.

Parameters

PlmImplConstants.PARAMETER_USERS_IDS: This parameter is required and allows to specify the users to which to send the part object. This parameter must contain a list of the users' ids, separated by the delimiter |.| (a pipe sign followed by a dot and by another pipe sign).

PlmImplConstants.PARAMETER_COMMENT: This parameter is optional and allows to include a comment when sending the object to the given users.

Options

No options of the object in the request are used when calling this function.

Result

The result of this operation will be summarized in the PlmStatus object associated with the PlmResponse. The PlmData in the response will be empty.

Samples

The following sample sends an existing part to the given users:

```
      public void part_sendtousers(PlmSession session)
      {
          PlmObject plmPart;
          try {
              plmPart = PlmFactory.createObject("part");
              // Use the number value of a part that you want to send to users
              plmPart.setAttributeValue("number", "P00023");
              PlmRequest request = PlmFactory.createRequest();
            //A list of valid user-ids can be retrieved with userdata.getactiveusers
 //callable
             request.getHeader().setParameter(PlmImplConstants.PARAMETER_USERS_IDS,
"6169343|.|704|.|6163545");
              request.getHeader().setParameter(PlmImplConstants.PARAMETER_COMMENT,
"This is a test send to you guys, don't worry.");
              PlmData data = request.getData();
              data.addObject(plmPart);
           PlmResponse response = session.execute(PlmActionConstants.SEND_TO_USERS,
request);
              PlmStatus status = response.getStatus();
              if (status != null && response.getStatus().isError()) {
```

```
                System.out.println("send-to-users failed. Error code = " +
status.getCode() + " " + status.getMessage());
            }
        } catch (PlmException e) {
            e.printStackTrace();
        }
    }
```

update

This function updates one of more part objects in the PLM server, according to the objects present in the request. Note that this function updates the values of the fields to which the entity is mapped; however, it does not modify relations or their attributes.

This action also updates one of more ECOs that already exist in the PLM server, according to the objects present in the request. This function also scans the relations of the change object(s) and updates the affected items table based on the relations of type “change-item-relation”.

For updating a PlmObject of type Userdata, it should only contain the new values for language and time zone.

Action Constant

part, document, change:

> com.agile.plmapi.connector.sdk.service. DataObjectService.update(PlmContext context, PlmRequest request)

userdata:

> com.agile.plmapi.connector.sdk. service. UserDataService. updateUserData(PlmContext context, PlmRequest request)

Action Entities

part, change, document, userdata

Input

The request data must contain at least one PlmObject of any entity type. Each of these PlmObjects represents the entities whose attributes should be saved (updated) in the server.

Parameters

PlmImplConstants.PARAMETER_CREATE_IF_DOES_NOT_EXIST: Set this to “true” in the request heasder if non-existent entities should also be created if they donot exist on the server. By default this has the value of “false”.

PlmImplConstants.PARAMETER_DISABLED_WARNINGS: This parameter is used to specify which warnings should be disabled before attempting to update the object. This value should be a separated list of the warning codes, using the following character sequence as separator: |.| (a pipe sign followed by a dot and followed by another pipe sign). The warning codes must be relevant to the PLM connector implementation.

Note Although this parameter can be set before the first time the request is sent to the server, a typical use of it is related to reattempting the function call when the result of its execution is of type warning. That is, when the response’ status after calling this function is of type ‘warning’ (which can be determined by calling PlmStatus.isWarning()), the status’ code contains the code of the warning, so this value can be added to the disabled-warnings parameter to call the function again.

Options

No options of the objects in the request are used when calling this function.

Result

The result of this operation will be summarized in the PlmStatus object associated with the PlmResponse. The PlmData will contain the same objects sent in the request; no attributes will be updated from the server side. Additionally, the PlmData will contain the representation of each entity that was created, updated to reflect the values of the newly created object's attributes as currently set in the server. However, it will not modify the relations of the objects sent in the request, meaning that they will be returned as they were in the request and not necessarily reflect all the relations present in the server for the enitities.

Samples

The following sample updates the value of the description of as existing part:

```
public PlmObject update(PlmSession session) {
        try {
            PlmObject plmPart = PlmFactory.createObject("part");
            // Load an exsiting part from the server
            plmPart = part_load(session);
            // Change some attribute values to update back to server
            plmPart.setAttributeValue("description", "This is the re-updated
description");
            PlmRequest plmRequest = PlmFactory.createRequest();
            // Uncomment the following lines for warning(s) you want suppressed
           // for this operation. Each warning is seperated by the "|.|" characters
            // plmRequest.getHeader().setParameter("disabled-warnings ",
"Integer_Value1|.|Integer_Value2"
            PlmData plmData = plmRequest.getData();
            plmData.addObject(plmPart);
            PlmResponse response = session.execute(PlmActionConstants.UPDATE,
plmRequest);
            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError()) {
               System.out.println("update failed. Error code " + status.getCode()
+ " " + status.getMessage());
            }
          plmPart = (PlmObject) response.getData().getObjects().iterator().next();
            return plmPart;
        } catch (PlmException e) {
            e.printStackTrace();
        }
        return null;
    }
```

This function updates the current user's language and time zone settings.

```
public PlmObject userdata_update(PlmSession session) {
        try {
            PlmObject plmUser = PlmFactory.createObject("userdata");
            // Put in some new values for the User data
            plmUser.setAttributeValue("language", "US-en");
            plmUser.setAttributeValue("timezone", "GMT-08:00");

            PlmRequest plmRequest = PlmFactory.createRequest();
            PlmData plmData = plmRequest.getData();
            plmData.addObject(plmUser);
            // Finally update the server side.
            PlmResponse response = session.execute(PlmActionConstants.UPDATE,
plmRequest);
            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError()) {
               System.out.println("update failed. Error code " + status.getCode()
+ " " + status.getMessage());
```

```
            }
          plmUser = (PlmObject) response.getData().getObjects().iterator().next();
            return plmUser;
        } catch (PlmException e) {
            e.printStackTrace();
        }
        return null;
    }
```

update-affected-items

This function scans the request's PlmData for objects of type 'change'. For each of these objects, it attempts to find the referenced change object in the PLM server and creates a new change object if it's not found. It then updates the affected items table based on the relations of type "change-item-relation" of the change object.

Note Since this function creates a change when it does not find it and it also updates the affected items, is a good alternative to change.create. In addition, the subclass of the change object to be created can be specified.

Action Constant

com.agile.plmapi.connector.sdk.service.ChangeService.updateAffectedItems(PlmContext context, PlmRequest request)

Action Entities

change

Input

The request data must contain at least one PlmObject of entity type 'change'. Each of these PlmObjects represents a change object, optionally to be created, whose affected items will be retrieved.

Parameters

PlmImplConstants.PARAMETER_CREATE_IF_DOES_NOT_EXIST: Set this to "false" in the request header if non-existent entities should not be created if they donot exist on the server. By default this has the value of "true".

Options

PlmImplConstants.OPTION_SUBCLASS: This option can be set for the PlmObjects of type 'change' to specify the id of the subclass to be used when creating a new change in the server.

Result

The result of this operation will be summarized in the PlmStatus object associated with the PlmResponse. The PlmData will contain all PlmObjects of type 'change' included in the request, updated to reflect all the properties and relations as currently stored in the PLM server.

Samples

The following sample creates a new change using the default subclass and autonumber source configured in the server and adds one existing part to the affected items table of the change:

```
public PlmObject updateitems(PlmSession session) {
        try {
            PlmObject plmChange = PlmFactory.createObject("change");
            //Here unlike change.update, we won't reference an existing object
            //instead we will create a new change for an existing part.
            plmChange.setAttributeValue("number", "#AUTO_NUMBER");
            //Uncomment the following and specify the value field to create
 //a custom change specified in the server
            //plmChange.setOptionValue("internal.class-id", value);
           plmChange.setAttributeValue("description", "This is the new description
of the change.");

            //Create a relation to existing part.
            PlmObject plmPart = PlmFactory.createObject("part");
            plmPart.setAttributeValue("number", "P01850");
            PlmRelation changeItemRelation;
           changeItemRelation = plmChange.createRelation("change-item-relation",
plmPart);
            changeItemRelation.setAttributeValue("item-number",
plmPart.getAttributeValue("number"));
            //Change an attribute of the item relation
            changeItemRelation.setAttributeValue("function", "Release");
            plmChange.addRelation(changeItemRelation);
            PlmRequest plmRequest = PlmFactory.createRequest();
            PlmData plmData = plmRequest.getData();
            plmData.addObject(plmChange);
            //Finally update the server side.
            PlmResponse response =
session.execute(PlmActionConstants.UPADTE_AFFECTED_ITEMS, plmRequest);
            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError()) {
                System.out.println("update-items failed. Error code " +
 status.getCode() + " " + status.getMessage());
            }
            plmChange = (PlmObject)
response.getData().getObjects().iterator().next();
            return plmChange;
        } catch (PlmException e) {
            e.printStackTrace();
        }
        return null;
    }
```

update-checkout-status

This function changes the checkout status of the file(s) associated with the given document or filefolder toplevel objects, which are represented by the relation of type document-to-file or container-file-relation respectively. The possible actions which will be executed for a document class type include check-in the attachment, check-out the attachment and cancel the check-out status of the attachment. Or in case of a filefolder type check-in the FileFolder, check-out the FileFolder and cancel the check-out status of the FileFolder.

Action Constant

com.agile.plmapi.connector.sdk.service.DataObjectService.changeCheckOutStatus (PlmContext context, PlmRequest request)

Action Entities

document, filefolder

Input

The request data must contain at least one top-level object of entity type ‘document’ or ‘filefolder’, which should have at least one child object of type ‘file’, with a document-to-file relation binding them.

Parameters

PlmImplConstants.PARAMETER_CHECK_OUT_MODE: This parameter specifies what action to execute on the file attachment(s). Its possible values are: PlmImplConstants.MODE_CHECKOUT, PlmImplConstants.MODE_CHECKIN and PlmImplConstants.MODE_CANCELCHECKOUT.

Options

No options of the objects in the request are used when calling this function.

Result

The result of this operation will be summarized in the PlmStatus object associated with the PlmResponse. The PlmData will contain the same objects and relations specified in the request; they will not be updated to show the updated status of the associated files.

Samples

The follwong sample checks out a file from the document specified.

```
public PlmObject document_updatecheckoutstatus(PlmSession session) {
        try {
            PlmObject plmDocument;
            plmDocument = PlmFactory.createObject("document");
            // Use the doc number of a document that already exists on the PLM
            // server
            plmDocument.setAttributeValue("number", "D05029");
            //Specify a file entity that is attached to the document in question
            PlmObject plmFile;
            plmFile = PlmFactory.createObject("file");
            plmFile.setAttributeValue("file-name", "testclient");
            plmFile.setAttributeValue("folder-number", "FOLDER03021");

            PlmRelation plmRelation = plmDocument.createRelation("document-file-
relation", plmFile);
            plmRelation.setAttributeValue("folder-number",
plmFile.getAttributeValue("folder-number"));

            PlmRequest plmRequest = PlmFactory.createRequest();

plmRequest.getHeader().setParameter(PlmImplConstants.PARAMETER_CHECK_OUT_MODE,
PlmImplConstants.MODE_CHECKOUT);
            PlmData plmData = plmRequest.getData();
            plmData.addObject(plmDocument);
            PlmResponse response =
session.execute(PlmActionConstants.UPDATE_CHECKOUT_STATUS, plmRequest);
            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError()) {
              System.out.println("update-checkout-status failed. Error code = " +
status.getCode() + " " + status.getMessage());
            }
```

```
            plmDocument = (PlmObject)
response.getData().getObjects().iterator().next();
            return plmDocument;
        } catch (PlmException e) {
            e.printStackTrace();
        }
        return null;
    }
```

The following sample checks out a given file folder. Checking in and as well as canceling the check out is then again a simple matter of changing the value of the "check-out-mode attribute".

```
public PlmObject filefolder_updatecheckoutstatus(PlmSession session) {
        try {
            //Use the File folder number that already exists on the PLM server
            PlmObject plmFileFolder;
            plmFileFolder = PlmFactory.createObject("filefolder");
            plmFileFolder.setAttributeValue("number", "FOLDER03021");

            PlmRequest plmRequest = PlmFactory.createRequest();

plmRequest.getHeader().setParameter(PlmImplConstants.PARAMETER_CHECK_OUT_MODE,
PlmImplConstants.MODE_CHECKOUT);
            PlmData plmData = plmRequest.getData();
            plmData.addObject(plmFileFolder);
            PlmResponse response = session.execute
PlmImplConstants.UPDATE_CHECKOUT_STATUS, plmRequest);
            PlmStatus status = response.getStatus();
            if (status != null && response.getStatus().isError()) {
                System.out.println("update-checkout-status failed. Error code = "
+ status.getCode() + " " + status.getMessage());
            }
            plmFileFolder = (PlmObject)
response.getData().getObjects().iterator().next();
            return plmFileFolder;
        } catch (PlmException e) {
            e.printStackTrace();
        }
        return null;
    }
```

CHAPTER 3

Details on Communication Options

This chapter presents the necessary steps in order to implement the asynchronous communication support. The chapter contains the following topic:

- *Firewall Support by allowing Asynchronous Communication*
- *Enabling HTTPS*

Firewall Support by allowing Asynchronous Communication

Asynchronous communication system in HTTP

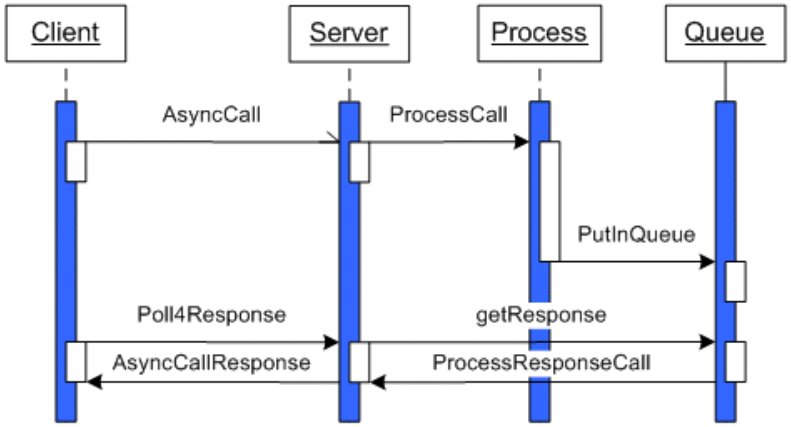
The asynchronous communication as depicted in the diagrams 3-1 and 3-2 below is a communication between two components where the client component sends a request to the server component and returns back instead of waiting for the response. The server receives the request and starts a process to do the task and be ready for the next request from the client instead of waiting until the response arrives and is thereby not blocking the communication pipe line.

The asynchronous communication supports two modes:

Pull mode

The first technique used is Pull model, where the client sends extra request on particular event, based on time or some listener triggers and the server sends the original response to this request. This is otherwise also called as polling scenario. As the HTTP itself is entirely based on the Request Response communication, it is optimal for the components to use this model as it adheres to the HTTP protocol standards.

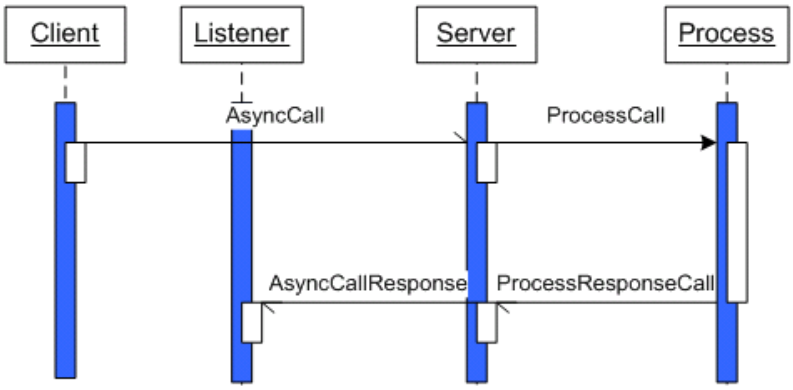
Figure 3-1: Pull Model for Asynchronous Communication



Push mode

The second technique used is a Push model, where the client sends the request and registers a listener in terms of listener URL and the server sends the response to the listener URL on finishing the task. Meanwhile the server is capable of receiving other requests from the client and thereby is not blocking the communication. To enable this, the client actually starts an HTTP server on the client side and listens for the HTTP request on a particular URL. This is the URL registered as listener URL during the asynchronous request. In HTTP this is possible mostly in an intranet (Trusted) environment where the server is allowed to make connection to the client machine.

Figure 3-2: Push Model for Asynchronous Communication



For more detailed information of how to switch between the modes, see the Getting Started section of the PLM API Technical Guide.

Client side response listener requirements:

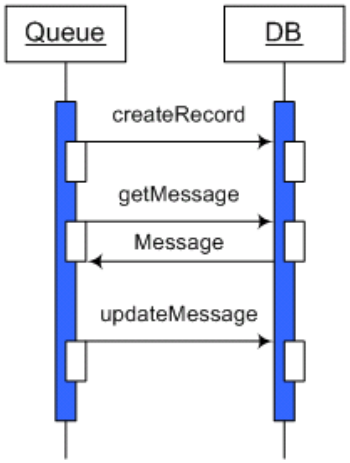
1 Implement the HTTP listener at the Client side to absorb the PLM response.

2 The listener should support HTTP post method and should answer HTTP-ok as response on success.

DB Messaging queue

Messaging is the communication technique used to enable asynchronous communication. The implemented messaging queue is the DB Messaging queue. Fig. 3-3 depicts the DB queue based messaging. The Queue component is the interface abstracting the DB behind, so that if a new message queue needs to be supported, it is easier to adapt by just replacing the DB part and writing only the access code, here the components till Queue in PLM API server should not be affected by this change. The queue component should basically provide methods for storing, updating, retrieving and cleaning.

Figure 3-3: Message Queuing with DB Queue



To enable DB queue messaging perform the following steps:

1. The asyncservice.sql should be executed for the first installation.
2. Make sure the agile database connection points to the database where the sql is executed.

Enabling HTTPS

To switch to HTTPS based communication certificates are needed for the system. A certificate is issued by a proper Certification Authority like Versign or Thawte. Certificates are what establish the identity of a person or a system that uses them. The certificates basically can contain roughly the following information:

- Public key
- Associated Private key
- Digital signature of the Issuing authority
- Timestamps

If someone gets a hold of your private key than the trust has been compromised and the perpetrator in procession of your private key could easily be masquerading about with your identity. In this case you need to get a new certificate issued and your previous certificate needs to be added to the Certificate revocation list, so everyone knows not to trust the entity presenting that compromised certificate anymore.

HTTPS uses the SSL protocol underneath which employs “public key cryptography” also known as “asymmetric cryptography” algorithm. What is “public key cryptography” is as follows. The private key and the public key present in a certificate, as discussed earlier, are mathematically related. The derivation of the private key from the public key is not at all practical. And the public key is freely distributed to all clients who want to talk to the owner of this public key. These clients can then encrypt the message they sent to the server using his public key. The encrypted message can only be decrypted by the server using his private key. No one else can decrypt the message.

A major problem with such key algorithms is the storing of keys. There are different ways about this problem but Java provides a system of vaults that are called as “stores”. There are 2 types of stores:

- Keystore
- Truststore

Java uses its own format for these stores which is called the "Java Key Store" or format or "JKS" in short. A "keytool" command line utility comes with all recent JDK's that can be used to manipulate the stores for e.g. importing, exporting or listing certificates.

So to support HTTPS, "truststores" and "keystores" need to be introduced in the system. A truststore is a vault where all certificates containing the public keys of people or entities that you want to talk to are stored. A keystore is a vault that is your private property and where you store the private and public key parts of your own certificate(s). You use the private key from this store to decrypt the message received from other parties. Additionally you can from this keystore export your own public key in a certificate to distribute it to other clients who wish to talk to you.

HTTPS can run in 2 modes:

1. Server Authentication mode

This mode dictates that the client will verify the identity of the server before a private communication line is opened.

2. Mutual Authentication mode

This mode dictates that in addition to the server authentication step by the client the server will also verify the identity of the client before it will start talking to it.

Client Side Settings for HTTPS

At the client side the first and foremost point to note is that only the "JKS" format for keystores and truststores will be supported. For **server authentication mode** to work the client machine needs to have the public key of the server in its truststore. If **mutual authentication mode** is planned to be used then the client should also have its own signed certificate stored in its keystore. Additionally the client's public key certificate needs to be inserted in the server's keystore.

Apart from this the client must know the location of its truststores and keystores for HTTPS to work. This needs to be additionally provided if the java default keystore and truststores are not used. This can be done programmatically by setting the following properties or starting the client with these java property values set:

- javax.net.ssl.keyStoreType (this will always be "jks")
- javax.net.ssl.keyStore
- javax.net.ssl.keyStorePassword
- javax.net.ssl.trustStoreType (this will always be "jks")
- javax.net.ssl.trustStore
- javax.net.ssl.trustStorePassword

Alternatively all of the above java properties can be set by convenience constants defined in PlmSession.java. So you can set these as parameters before opening your session with the PLM server and the PLM API will take care of initializing these system properties for you. You can set them as follows:

```
PlmSession session = PlmFactory.getSession();
      Map map = new HashMap();
      map.put(PlmSession.KEY_USER, "admin");
      map.put(PlmSession.KEY_PASSWORD, "agile");
      map.put(PlmSession.KEY_SYSTEM, PlmSession.SYSTEM_SDK);
      map.put(PlmSession.KEY_APPLICATION, "https://localhost:4343/Agile");
      map.put(PlmSession.KEY_URL, "https://localhost:4343/Agile/services");
      map.put(PlmSession.KEY_PROTOCOL, PlmSession.PROTOCOL_WEBSERVICE);
      //SSL parameters.
      map.put(PlmSession.KEY_KEYSTORE_TYPE, "jks");
      map.put(PlmSession.KEY_KEYSTORE_LOCATION, "C:\\keystore.jks")
```

```
map.put(PlmSession.KEY_KEYSTORE_PASSWORD, "agie123");
map.put(PlmSession.KEY_TRUSTSTORE_TYPE, "jks");
map.put(PlmSession.KEY_TRUSTSTORE_LOCATION, "C:\\keystore.jks")
map.put(PlmSession.KEY_TRUSTSTORE_PASSWORD, "agie123");
session.open(map);
```

Finally the SSL URL and port needs to be supplied to the PLM API over which it can then communicate with the server.

Server Side Settings for HTTPS

The server side configurations for HTTPS vary from server to server. But they need to be enabled explicitly for HTTPS mode to function.

With IBM WebSphere the server's keystores and truststores are in "jks" format. Their location needs to be provided in the administrative settings for the relevant application server. The truststores and keystores must of course contain the required certificates.

CHAPTER 4
Extensions Framework

This chapter presents the steps for extending the PLM-API Server with SDK Handlers and Macro Level Handlers. This chapter *contains the following topic:*

- *Extending the PLM-API Server with Handlers*

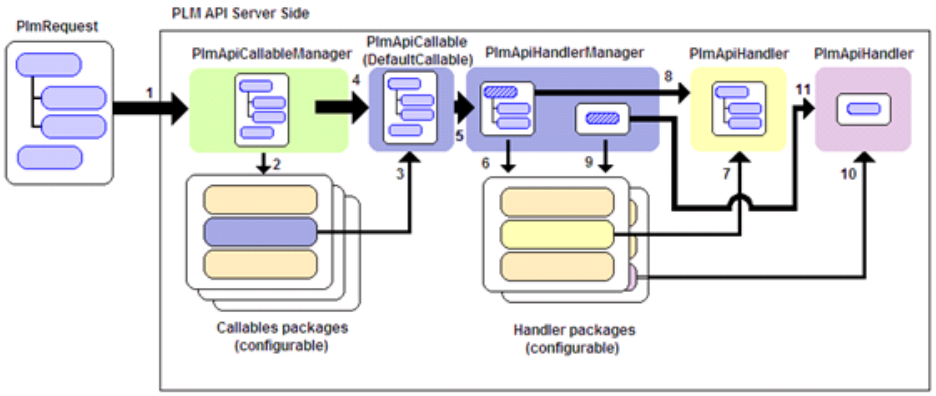
Extending the PLM-API Server with Handlers

The server side of the PLM API implementation can be extended to provide additional functionality through the PLM API or to customize the existing functionality.

This chapter explains the mechanism how the server side extensions work, how they can be extended and how they are configured to be used.

Callables and Handlers

The diagram below shows how this is handled in the PLM API:



This diagram represents the sequence of operations that take place when a request arrives at the server side of the PLM API, which are:

1. When a PLM API request arrives at the server side, it can contain one or more top level objects in the data, each of which can have relations binding it to child objects. The diagram shows a request with two top level objects, the first one of which has relations to two other objects.

2. The request is initially handled by the PlmApiCallableManager, which receives it and tries to locate a proper PlmApiCallable class to process the request based on the operation name. The lookup process consists on constructing the target class name from the operation name and then on locating a matching class in the configured packages. In order to assemble the class name, the first letter of the operation name is converted to upper case, as well as any letter that follows a hyphen (the hyphen itself is removed). For example, the operation "create-snapshot" will be converted to CreateSnapshot, and the operation "updatefolder" will be converted to Updatefolder. Finally, the word "Callable" is appended to the result string, so that the actual callable to look up for a function called "create-snapshot' is a class with the name CreateSnapshotCallable.

3. Once the class name to look for has been determined, the PlmApiCallableManager queries the connector configuration framework for a list of packages to scan in search for this class. This list is specified in the server side configuration file. The PlmApiCallableManager goes through the list in the same order that it has been specified in the configuration file and attempts to find a matching class in those packages. If a matching class is found in one of the packages the search stops. The class is validated to see if it implements the PlmApiCallable interface and if so, it is instantiated to process the request. In this case, namely when a specific callable is implemented for an operation, the next steps in the sequence depend entirely on the actual implementation of this callable and they may not match what is shown in the diagram above. For the case when a matching callable class is not found, the framework instantiates the DefaultCallable to handle the request.

4. Next, the PlmApiCallableManager sends the request to the callable for further processing. The sequence shown in the diagram above is based on the assumption that the default callable was instantiated to handle the request.

5. The default callable scans the incoming request and observes each top level object. For each of those objects, the callable creates a "mini" request, which contains this object only, and forwards it to the PlmApiHandlerManager for further processing. Note that the object in this stripped-down request still contains all of the hierarchy provided by its relations to child objects (see the diagram above).

6. When the request arrives at the PlmApiHandlerManager, it attempts to locate an appropriate handler to process it. To do this, the handler relies on the entity of the top level object that it can see in the request. This takes place as follows: The manager first uses the entity name and the operation name to construct the class name. It does this by first formatting the entity name and the operation name with the same approach as how the PlmApiCallableManager does at the callable level. Next it concatenates these values and appends the word "Handler" is appended at the end of the combined string. So, for example, in order to provide functionality for an operation called "create-snapshot" on an entity named "cad-model", the manager will try to look for a class called CadModelCreateSnapshotHandler in the packages specified in the configuration file.
If, after scanning all of the packages specified in the configuration, a matching class is not found, then the PlmApiHandlerManager attempts to locate a handler using the entity's class name and the operation name. That is, if the "cad-model" entity mentioned above is of class "document", then the PlmApiHandlerManager will scan the packages again, searching for a class called DocumentCreateSnapshotHandler.

7. If a matching handler class is found, then it is validated to verify that it implements the PlmApiHandler interface and if so, it is instantiated. If no matching handler class is found, then the PlmApiHandlerManager returns an error response stating that the operation is unknown for the object's entity.

8. When a matching handler is found, the stripped-down request is sent to the handler's instance to be processed and the response returned by it is temporarily stored in the Callable, so that later on, it can be merged with the responses that come from other handler instances which process the remaining top level objects present in the original request.

9. - 11. The DefaultCallable continues to scan the original incoming request and repeats steps 6, 7 and 8 for each top level object in the request.

12. The complete response is assembled from the individual responses of each handler's process and it is sent back to the PlmApiCallableManager, which, in turn, sends it back to the calling application.

Given the process described above, a user willing to provide an extension would have to do the following:

1. If the purpose of the extension is to provide support for a new function that is not handled by the standard PLM API, the user could:
   a. Implement a new PlmApiCallable class which gets the complete request and processes it according to the needs, or
   b. Rely on the DefaultCallable implementation to parse the request and implement proper PlmApiHandler classes to process the top level objects one at a time based either on the entity name or on the entity class.
2. If the purpose of the extension is to allow the standard PLM API to support an existing operation on an entity, which the standard PLM API implementation doesn't support, the user could:
   a. Implement a PlmApiHandler which is specific to one single entity and name it following the standard mentioned above, based on the entity name, or
   b. Implement a PlmApiHandler that is generic to all entities of the same class and name it following the standard mentioned above, based on the entity class.

If the PLM API handles the operation through a specific callable, instead of the default one, the request might not reach the handler level, depending on the actual implementation of the specific callable. This might require the user to also override the specific callable for the given operation (see option a. below).

3. If the purpose of the extension is to override the existing functionality in the PLM API for a given operation, the user could:
   a. Implement a new PlmApiCallable that gets the complete request and handle the request according to the custom needs, eventually calling handlers for each object and/or relation, or
   b. Implement a new PlmApiHandler class based on the entity's class name, so that it handles the operation in a custom way for every entity of the corresponding class, or
   c. Implement a new PlmApiHandler class based on the entity's name, so that it handles the operation in a custom way for objects on the given entity only.

In all of the cases mentioned above, the server side configuration file sould be modified after implementing the new classes, to include the package name(s) of the custom callables or handlers in the list, as high in the sequence as required so that the custom classes are located before a standard PLM API class (if one exists with the same name).

To do this, the user should locate the file plm-api-server.jar in the PLM API deployment. Inside this file, the user will find the server side configuration file. This file depends on the PLM Connector, so for the Agile 9 server, this file is plm-api-sdk.xml and for the Agile 6 server, this file is plm-api-eci.xml. This file should be extracted from the jar file, modified and then reinserted in the jar file.

For Agile 9 it is also possible to have the configuration files in a FileFolder object. The basic configuration file is called plm-api-confif-properties and is required to be attached to a FileFolder called PLMAPI_CONFIG. In this configuration file all other configuration files and their FileFolder name are configured.

Once extracted, the user should locate the "extensions" element. In the out-of-the-box configuration files, this element looks like this:

```
<extensions>
<callables package="com.agile.plmapi.connector.sdk.callables"/>
<handlers package="com.agile.plmapi.connector.sdk.part"/>
<handlers package="com.agile.plmapi.connector.sdk.document"/>
<handlers package="com.agile.plmapi.connector.sdk.filefolder"/>
<handlers package="com.agile.plmapi.connector.sdk.change"/>
<handlers package="com.agile.plmapi.connector.sdk.userdata"/>
</extensions>
```

The list of packages to scan when looking for callables or handlers is built from the “callables” and “handlers” elements located inside this “extensions” element. In order to add a package to scan when searching for PlmApiCallable classes, the user needs to add a new “callables” element. The value of the “package” attribute would be the new package to scan. Similarly, in order to add a package to scan when searching for PlmApiHandler classes, the user needs to add a new “handlers” element. Keep in mind that order inside the sequence is important, because the server side will scan the packages as listed in this node, from top to bottom until a matching class is found.

Debugging the server side extensions

There is a special connection mode for the PLM API to reduce the turn around time when developing server extensions. It will allow to locally debug the source code inside your IDE.

Instead of using the standard connection mode (PlmSession.PROTOCOL_WEBSERVICE), the connection mode PlmSessionImpl.PROTOCOL_DEBUG could be used for the session entry PlmSession.KEY_PROTOCOL.

If used with Agile 9, the SDK calls are mode from the client side into the Agile 9 server. Therefore, some system properties needs to be set depending on the Application Server.

Please refer to the Agile 9 documentation for further information.

This special connection mode should only be used for development purposes and not in production code.

Samples

Following are some samples to illustrate the use of the extension framework for working Agile 9 (and the Agile SDK). They are basically similar when working with Agile e6.

1 To override the function “create” at the callable level:

   a First, the user needs to create the callable class (CreateCallable.java). This callable will pre-process the request and then rely on handlers for each individual object. The class could look like this:

```
package com.mycompany.plmapi.cad;

import ...;

public class CreateCallable implements SdkCallable {

  public CreateCallable() {

  super(CreateCallable.class);
  }

   public PlmResponse call(PlmApiHandlerManager manager, PlmContext context, String
operation, PlmRequest request)
         throws PlmException {
      List responses = new ArrayList ();
      PlmRequest myRequest = preProcessRequest(request, operation);
      Collection objects = myRequest.getData().getObjects();
      Iterator it = object. .iterator () ;
     while (it. hasNext()) {
        PlmObject object = (PlmObject) it.next();
        // Forward the request to the PlmApiHandlerManager with one
        // object at a time, so that appropriate handlers are located
        // and instantiated to process it.
        responses.add(manager.call(context, operation, request.getHeader(),
object));
      }
      return mergeResponses(responses);
   }
```

```
   private PlmRequest preProcessRequest(PlmRequest request, String operation) {
      PlmRequest result = PlmImplHelper.createRequest(request, operation, false,
false);
      // Do some custom handling of the request's data or parameters
      return result;
   }

}
```

b Next, the user needs to edit the server side configuration file and add the package for the new callable:

```
<extensions>
<callables package="com.mycompany.plmapi.cad"/>
<callables package="com.agile.plmapi.connector.sdk.callables"/>
<handlers package="com.agile.plmapi.connector.sdk.part"/>
<handlers package="com.agile.plmapi.connector.sdk.document"/>
<handlers package="com.agile.plmapi.connector.sdk.filefolder"/>
<handlers package="com.agile.plmapi.connector.sdk.change"/>
<handlers package="com.agile.plmapi.connector.sdk.userdata"/>
</extensions>
```

c Finally, the user needs to compile and deploy the CreateCallable class in the proper location so that it is available to the server side of the PLM API. For the Agile 9 PLM Server series, this can be achieved by creating a jar file with the compiled class and copying this jar file to the WSX extensions folder, where the standard server side PLM API jar files are located.

2 To provide support for a function called “create-snapshot” for all entities of class “part” relying on the DefaultCallable to parse the request:

a First, the user needs to create a handler class (PartCreateSnapshotHandler.java). This class could look like this:

```
package com.mycompany.plmapi.cad;

import ...;
public class PartCreateSnapshotHandler implements SdkHandler {

  public PartCreateSnapshotHandler() {
        super(PartCreateSnapshotHandler.class);
     }

public PlmResponse call(PlmContext context, String operation, PlmRequest request)
throws PlmException {
      PlmResponse result = PlmImplHelper.createResponse(request, false);
      PlmObject object = (PlmObject)
request.getData().getObjects().iterator().next();
      // Do something with the object
      return result;
   }

}
```

b Next, the user needs to edit the server side configuration file and add the package for the new handler:

```
<extensions>
<callables package="com.agile.plmapi.connector.sdk.callables"/>
<handlers package="com.mycompany.plmapi.cad"/>
<handlers package="com.agile.plmapi.connector.sdk.part"/>
<handlers package="com.agile.plmapi.connector.sdk.document"/>
<handlers package="com.agile.plmapi.connector.sdk.filefolder"/>
<handlers package="com.agile.plmapi.connector.sdk.change"/>
<handlers package="com.agile.plmapi.connector.sdk.userdata"/>
```

```
</extensions>
```

c Finally, the user needs to compile and deploy the PartCreateSnapshotHandler class in the proper location so that it is available to the server side of the PLM API. For the Agile 9 PLM Server series, this can be achieved by creating a jar file with the compiled class and copying this jar file to the WSX extensions folder, where the standard server side PLM API jar files are located.

This page is blank