Agile Engineering Collaboration Services Extensions Guide

v1.3.1 for Agile 9

Part No. E12176-01

Make sure you check for updates to this manual at the Oracle Technology Network (OTN) Web site

Copyrights and Trademarks

February 21, 2008

REVISIONS

	Date	Pages Affected	Description
A	10/08/2007	All	New release of PLM API Technical Guide
B	25/02/2008	All	Updated release version EC to 1.3.1

PREFACE

The Agile documentation set includes Adobe® Acrobat™ PDF files. The Oracle Technology Network (OTN) Web site 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.

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 www.adobe.com.

The Oracle Technology Network (OTN) Web site 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 www.agile.com/support.

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

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.

Agile Training Aids

Go to the Agile Training Web page for more information on Agile Training offerings.

Chapter 1 Using the Extensions

Chapter 2 Translation and Mapping

CHAPTER 1
Using the Extensions

This chapter describes the different steps to follow regarding calling PLM-API standard or extension functions to complete each use case supported by the CAD Integration Framework. This chapter contains the following topics:

- *Introduction*
- *Installation*
- *Request Types used by the Extension Functions*
- *Use Cases Summary*
- *Use Cases*
- *PLM API XML Request Samples*

Introduction

This document describes how to use the Pre-packaged Extensions in EC 1.3.1, which are implemented under the PLM API extensions framework.

As mentioned in the PLM API User's Guide, each call to the PLM API functions takes an incoming request and returns a response. Typically, a request contains a header section with zero or more parameters in and a data section with zero or more objects on which the invoked function will act. The actual parameters and objects as well as object types depend on each particular function. This is also the case for the extension functions that have been implemented to conform the CAD Integration Framework.

Installation

The installation of the extensions is covered in the PLM API Technical Guide, Chapter 1, as part of the overall PLM API installation process.

Request types used by the extension functions

Standard Un-Mapped Request

This is a request that is created at the client side, either by using the PLM API Java Client Library or by generating a PLM API request from an XML file that conforms with the XML Schema defined in plm-api.xsd.

The entities recognized for this type of request are the following:

For objects: object and file.

For relations: object-object-relation and object-file-relation.

The parameters to include in the header of this type of request are those required by the MappingCallable to allow the selection of the desired Mapping Template, and they include:

- createdByUser
- createdDate
- createdByTool
- createdByToolVersion
- createdForProject

Note that all objects included in the request may make use of the key attribute to store human readable information, which is useful during debugging processes.

An example of an unmapped request can be found in Appendix A of this document.

Standard Mapped Request

This is a request that the client side will generate by copying the header and the data sections of the response sent back by the server after a call to the function "mapping". Typically, the response returned by this function call contains the same parameters in its header as the request did, and the data section of the response contains the objects that were generated from the objects present in the data section of the request.

The objects contained in the data section of the response have different entities as those in the request, and the entity names will typically match the subclass names of the objects in the a9 server. Each object will also have the correct Number attribute filled in.

The attributes of these objects will also match the attributes' names defined in a9 and they will have the internal-id and destination filled in.

Both objects and attributes will have the 'isdirty' and 'mapped' flags properly set depending on their status. These flags will be used by other callables in the sequence to complete a use case.

Valid entities for objects in the data section of this type of request are:

For objects: document (and the subclasses of the Documents class), part (and the subclasses of the Parts class), change (and the subclasses of the Changes class), file, filefolder (and the subclasses of the FileFolders class).

For relations: object-object-relation, object-change-relation, object-file-relation, object-filefolder-relation, file-filefolder-relation, change-object-relation, change-file-relation, change-filefolder-relation

Use Cases Summary

The following table contains an overview of all use cases supported by the extensions.

Table 1-1: Use Cases

Title	Description
Search	The goal of this use case is to find objects in the Agile server; this included searching for Items, Changes, FileFolders and Programs.
Browse (Get Object Detail)	The goal of this use case is to get the complete structure of Agile objects, including Items, Changes, FileFolders, Programs, Manufacturer Parts, Manufacturers.
Check-in	The goal of this use case is to synchronize a released CAD model from the PDM system to Agile.

Table 1-1: Use Cases

Title	Description
Check-out	The goal of this use case is to lock the objects in Agile and download the attached files.
Cancel Check-out	The goal of this use case is to cancel the lock status the objects in Agile.
Change Synchronization (Input) / Save Dialog	The goal of this use case is to create a change object in Agile of change the status of a change object in Agile to reflect the status of a LCR in the PDM system.

Use Cases

Search

Goal

The goal of this use case is to find objects in the Agile server; this includes searching for Items, Changes, FileFolders and Programs.

Flow

1 Call function “search”

- Request:
  - One or more query objects with a search criteria resembling Agile’s advanced search syntax, such as:
    - Mechanical Part.Number ‘starts with’ ‘MEP’
  - Parameters:
    - None required.
- Response:
  - The response’s data will contain one or more objects, each representing an object in the Agile server that matches the given criteria. The objects with include the attributes in the Title Block/Cover Page, Page Two and Page Three tabs.

Samples

The following sample searches for documents of subclass “Mechanical Part” whose Number attribute starts with the character sequence “MEP”:

```
public PlmResponse searchDocument(PlmSession session) {
      PlmResponse response = null;
      try {
         PlmQuery documentQuery = PlmFactory.createQuery("Mechanical Part");

         PlmAttributeCriteria attributeCriteriaPart =
PlmFactory.createQueryAttributeCriteria("Mechanical Part", "Number",
               PlmAttributeCriteria.OP_STARTS_WITH, "MEP");
         PlmExpressionCriteria criteriaPart =
PlmFactory.createQueryExpression("Mechanical Part", attributeCriteriaPart);
         documentQuery.setExpression(criteriaPart);
         PlmRequest request = PlmFactory.createRequest();
         request.addQuery(documentQuery);
         response = session.execute("search", request);
         try {
            System.out.println(PlmResponseHelper.marshal(response).toString());
         }
         catch (Exception ex) {
            throw new RuntimeException(ex);
```

```
            }
         }
         catch (PlmException ex) {
            ex.printStackTrace();
         }
         return response;
      }
```

Browse (Get Object Detail)

Goal

The goal of this use case is to get the complete structure of Agile objects, including Items, Changes, FileFolders, Programs, Manufacturer Parts, Manufacturers.

Flow

1 Call function "get-object-detail"

- Request:
  - This function takes a standard PLM API request, which although not exactly the same, is very similar to a standard mapped request as mentioned in Chapter 1. This request should have one or more objects in the data section, which the user is looking for in the a9 system. In order for the CIF framework to locate an object, the client side must provide enough information to uniquely identify the object, as described below:
    - For Parts and Documents:
    - Required: The attribute mapped to the TitleBlock.Number field (The name of this attribute depends on the mapping configuration file, it is usually 'Number').
    - Optional: The attribute mapped to the TitleBlock.Rev field. (Same as above, the attribute name depends on the mapping, and it is usually 'Rev'). Special values that can be used for this attribute include:
    - LATESTPENDING to retrieve the latest pending revision of the item.
    - LATESTRELEASED to retrieve the latest released revision of the item.

    - For Changes:
    - Required: The attribute mapped to the CoverPage.Number field (usually 'Number' but depends on the mapping)
    - For FileFolders:
    - Required: The attribute mapped to TitleBlock.Number (usually 'Number' but depends on the mapping).
    - Optional: The attribute mapped to TitleBlock.Version (usually 'Version' but depends on the mapping). If not specified, then the latest version of the FileFolder is retrieved.
    - For Programs:
    - Required: The attribute mapped to GeneralInfo.Number (usually 'Number' but depends on the mapping).
    - For Manufacturers:
    - Required: The attribute mapped to GeneralInfo.Name (usually 'Name' but depends on the mapping).
    - For ManufacturerParts:
    - Required: The attribute mapped to GeneralInfo.Manufacturer Name (usually 'ManufacturerName' but depends on the mapping).

- Required: The attribute mapped to GeneralInfo.Manufacturer Part Number (usually 'MAnufacturerPartNumber' but depends on the mapping).

❑ Response:

- One or more top level objects, each corresponding to one top level object in the request. Possible scenarios for each object are:
  - The object was found in the PLM server, so the complete structure of the object is returned, including all mapped attributes and relations. The option "internal.found" will be set to "true".
  - The object was not found in the PLM server, in this case, the object will be identical to the one sent in the request, but the "internal.found" option will be set to "false".

Samples

The following sample gets the details of a document in the a9 server without knowing the actual document's subclass. The top level object returned in the response will include the subclass information if a matching document object is found:

```
public PlmResponse getObjectDetail(PlmSession session) throws PlmException {
      PlmObject object = PlmFactory.createObject("document");
      object.setAttributeValue("Number", "ENG_11000-07");
      object.setAttributeValue("Rev", "LATESTPENDING");
      PlmRequest request = PlmFactory.createRequest();
      request.getData().addObject(object);
      return session.execute("get-object-detail", request);
   }
```

Check-in

Goal

The goal of this use case is to synchronize a released CAD model from the PDM system to Agile.

Flow

1 Call function "mapping"

❑ Request:

- A standard pre-mapped request as described in Chapter 1.
- Parameters:
  - logginglevels (possible values are: note, warn, noerrors, nofatals).
  - createdByUser
  - createdDate
  - createdByTool
  - createdByToolVersion
  - createdForProject

❑ Response:

- Mapped objects in the data section and same parameters from the request copied to the response's header.
- Status may contain notes, warnings, errors or fatals.

2 Verify that the mapping function returned a success response.

3 Call function "checkstatus"

This function processes objects of type "object" with isdirty=true, valid name=subclass, valid "Number" attribute. It also processes any objects of type "change" that are related to "object" via a valid object-change-relation entity.

- ❑ Request:
  - A mapped request created from the response to the call to “mapping”.
  - Parameters:
    - logginglevels (possible values are: note, warn, noerrors, nofatals).
- ❑ Response:
  - The same data sent in the request. Response status will include error elements if the check failed for one or more objects.

4 Verify that the response does not contain errors.

5 Call function “reserve”

- ❑ Request:
  - A standard mapped request created from the response to the call to “checkstatus”.
  - Parameters:
- ❑ Response:
  - The same data sent in the request. Response status will include error elements if the reserve function failed for one or more objects.

6 Verify that the response from the call to reserve does not contain errors.

7 Call function “cif validate”

- ❑ Request:
  - A standard mapped request created from the response to the call to “reserve”.
  - Parameters:
    - logginglevels (possible values are: note, warn, noerrors, nofatals).
    - save-axml (possible values are: true, false). If set to true then the aXML file that results from the conversion of the PLM API request to aXML format will be saved to the FileServer and the file Id will be returned.
- ❑ Response:
  - Identical to the request, the response’s status will include warning, error or fatal elements according to the results of the validation.
  - Parameters:
    - axml-file-id If the parameter “save-axml” was set to “true”, then this parameter will contain the id of the aXML file in the FileServer.

8 Verify that the response from the call to validate does not contain errors.

Optionally download the aXML file from the FileServer for debugging purposes (see standard PLM API documentation of function “download-files”).

9 Call function “upload-files”

This function will iterate through the objects of entity type ‘file’ in the request and for each of them, it will upload the corresponding file to the FileServer and will update the object to set its attribute ‘file-id’ to the id of the file assigned by the FileServer. If the file object is not a top level object, then it will be ignored unless the parameter recourse is set to true.

If a particular file present in the request should be skipped, then one should set its 'ignore' flag to 'true'.

- Request:
  - A standard mapped request created from the response to the call to "validate".
  - Parameters:
    - recourse needs to be set to "true" to allow file objects that are not top level to be uploaded.
- Response:
  - identical to the request, the response status will include warning or error elements according to the results of the upload task. Additionally, each object of type "file" in the request will contain two attributes additional to the ones it had before the function call.
    - file-id
    - file-size

10 Verify that the response from the call to upload-files does not contain errors.

11 Call function "update"

- Request:
  - A standard mapped request created from the response to the call to "upload-files".
  - Parameters:
- Response:
  - Identical to the request, the response status will include warning or error elements according to the results of the call to update.

12 Verify that the response from the call to update does not contain errors.

13 Call function "release"

- Request:
  - A standard mapped request created from the response to the call to "update".
  - Parameters:
- Response:
  - Identical to the request, the response status will include warning or error elements according to the results of the call to release.

Samples

The following sample walks through the check in use case:

```
public PlmResponse checkIn(PlmSession session) throws PlmException {
      PlmRequest request = null;
      try {
         request = PlmFactory.createRequest(new URL("file:/D:/Data/
ADW_PreMapped_Footprint_1.xml"));
      } catch (Exception ex) {
         throw new PlmException("Invalid URL or request format.", ex);
      }
      PlmResponse response = session.execute("mapping", request);
      // check to see if response is of type success
      if (!response.getStatus().isSuccess()) {
```

```
        System.out.println("Calling to 'mapping' function failed. Details are shown
below:");
          System.out.println(response.getStatus().getAllStatuses());
          return response;
       }
       request = PlmFactory.createRequest(response, "checkstatus");
       response = session.execute("checkstatus", request);
       // check to see if call to check-status has errors
       if (response.getStatus().hasErrors()) {
        System.out.println("Calling to 'check-status' has errors. Details are shown
below:");
          System.out.println(response.getStatus().getErrors());
          return response;
       }
       request = PlmFactory.createRequest(response, "reserve");
       response = session.execute("reserve", request);
       // check to see if call to reserve has errors
       if (response.getStatus().hasErrors()) {
          System.out.println("Calling to 'reserve' has errors. Details are shown
below:");
          System.out.println(response.getStatus().getErrors());
          return response;
       }
       request = PlmFactory.createRequest(response, "cif validate");
       response = session.execute("validate", request);
       // check to see if call to validate has errors
       if (response.getStatus().hasErrors()) {
        System.out.println("Calling to 'cif validate' has errors. Details are shown
below:");
          System.out.println(response.getStatus().getErrors());
          return response;
       }
       request = PlmFactory.createRequest(response, "upload-files");
       request.getHeader().setParameter("recurse", "true");
       response = session.execute("upload-files", request);
       // check to see if call to upload-files has errors
       if (response.getStatus().hasErrors()) {
        System.out.println("Calling to 'upload-files' has errors. Details are shown
below:");
          System.out.println(response.getStatus().getErrors());
          return response;
       }
       request = PlmFactory.createRequest(response, "update");
       response = session.execute("update", request);
       // check to see if call to update has errors
       if (response.getStatus().hasErrors()) {
          System.out.println("Calling to 'update' has errors. Details are shown
below:");
          System.out.println(response.getStatus().getErrors());
          return response;
       }
       request = PlmFactory.createRequest(response, "release");
       response = session.execute("release", request);
       return response;
    }
```

Check-out

Goal

The goal of this use case is to lock the objects in Agile and download the attached files.

Flow

1 Call function “mapping”

- ❑ Request:
  - A standard pre-mapped request as described in Chapter 1.
  - Parameters:
    - loggінglevels (possible values are: note, warn, noerrors, nofatals).
    - createdByUser
    - createdDate
    - createdByTool
    - createdByToolVersion
    - createdForProject
- ❑ Response:
  - Mapped objects in the data section and same parameters from the request copied to the response’s header.
  - Status may contain notes, warnings, errors or fatals.

2 Verify that the mapping function returned a success response.

3 Call function “checkstatus”

This function processes objects of type "object" with isdirty=true, valid name=subclass, valid "Number" attribute. It also processes any objects of type "change" that are related to "object" via a valid object-change-relation entity.

- ❑ Request:
  - A mapped request created from the response to the call to “mapping”.
  - Parameters:
    - loggінglevels (possible values are: note, warn, noerrors, nofatals).
- ❑ Response:
  - The same data sent in the request. Response status will include error elements if the check failed for one or more objects.

4 Verify that the response does not contain errors.

5 Call function “reserve”

- ❑ Request:
  - A standard mapped request created from the response to the call to “checkstatus”.
  - Parameters:
- ❑ Response:
  - The same data sent in the request. Response status will include error elements if the reserve function failed for one or more objects.

6 Verify that the response from the call to reserve does not contain errors.

7 Call function “get-object-detail”

- Request:
  - A standard mapped request created from the response to the call to “reserve”.
  - Parameters:
    - loggінglevels (possible values are: note, warn, noerrors, nofatals).
- Response:
  - The response’s data will contain the complete structure of the objects in the request, including the file-ids of the files attached to them.

8 Verify that the response from the call to get-object-detail does not contain errors.

(Should the user here go through the request and provide a file-path attribute for the files to be downloaded? This attribute is required for download to work so the other option is that it would have already been populated by the call to mapping)

9 Call function “download-files”

This function will iterate through the objects of entity type ‘file’ in the request and for each of them, it will download the corresponding file from the FileServer to the location specified by the “file-path” attribute. If the file object is not a top level object, then it will be ignored unless the parameter recourse is set to true.

If a particular file present in the request should be skipped, then one should set its ‘ignore’ flag to ‘true’.

- Request:
  - A standard mapped request created from the response to the call to “get-object-detail”.
  - Parameters:
    - recurse needs to be set to “true” to allow file objects that are not top level to be downloaded.
- Response:
  - Identical to the request, the response status will include warning or error elements according to the results of the download task.

Samples

The following sample walks through the check out use case:

```
public PlmResponse checkOut(PlmSession session) throws PlmException {
      PlmRequest request = null;
      try {
         request = PlmFactory.createRequest(new URL("file:/D:/Data/
ADW_PreMapped_Footprint_1.xml"));
```

Is this request also valid for checkOut? If not, please provide a new request in the Appendix A.

```
} catch (Exception ex) {
         throw new PlmException("Invalid URL or request format.", ex);
      }
      PlmResponse response = session.execute("mapping", request);
      // check to see if response is of type success
      if (!response.getStatus().isSuccess()) {
       System.out.println("Calling to 'mapping' function failed. Details are shown
below:");
         System.out.println(response.getStatus().getAllStatuses());
         return response;
      }
      request = PlmFactory.createRequest(response, "checkstatus");
      response = session.execute("checkstatus", request);
```

```
      // check to see if call to check-status has errors
      if (response.getStatus().hasErrors()) {
       System.out.println("Calling to 'check-status' has errors. Details are shown
below:");
         System.out.println(response.getStatus().getErrors());
         return response;
      }
      request = PlmFactory.createRequest(response, "reserve");
      response = session.execute("reserve", request);
      // check to see if call to reserve has errors
      if (response.getStatus().hasErrors()) {
         System.out.println("Calling to 'reserve' has errors. Details are shown
below:");
         System.out.println(response.getStatus().getErrors());
         return response;
      }
      request = PlmFactory.createRequest(response, "get-object-detail");
      response = session.execute("get-object-detail", request);
      // check to see if call to get-object-detail has errors
      if (response.getStatus().hasErrors()) {
        System.out.println("Calling to 'get-object-detail' has errors. Details are
shown below:");
         System.out.println(response.getStatus().getErrors());
         return response;
      }
```

Should something be done here?

```
request = PlmFactory.createRequest(response, "download-files");
      request.getHeader().setParameter("recurse", "true");
      response = session.execute("download-files", request);
      // check to see if call to download-files has errors
      if (response.getStatus().hasErrors()) {
         System.out.println("Calling to 'download-files' has errors. Details are
shown below:");
         System.out.println(response.getStatus().getErrors());
         return response;
      }
      return response;
   }
```

Cancel Check-out

Goal

The goal of this use case is to cancel the lock status the objects in Agile.

Flow

1 Call function “mapping”

- Request:
  - A standard pre-mapped request as described in Chapter 1.
  - Parameters:
    - loggінglevels (possible values are: note, warn, noerrors, nofatals).
    - createdByUser
    - createdDate
    - createdByTool

- createdByToolVersion
- createdForProject

❑ Response:

- Mapped objects in the data section and same parameters from the request copied to the response's header.
- Status may contain notes, warnings, errors or fatals.

2 Verify that the mapping function returned a success response.

3 Call function "checkstatus"

This function processes objects of type "object" with isdirty=true, valid name=subclass, valid "Number" attribute. It also processes any objects of type "change" that are related to "object" via a valid object-change-relation entity.

❑ Request:

- A mapped request created from the response to the call to "mapping".
- Parameters:
  - logginglevels (possible values are: note, warn, noerrors, nofatals).

❑ Response:

- The same data sent in the request. Response status will include error elements if the check failed for one or more objects.

4 Verify that the response does not contain errors.

5 Call function "cancelreserve"

This function processes objects of type "object" with isdirty=true, valid name=subclass and valid "Number" attribute.

❑ Request:

- A standard mapped request created from the response to the call to "checkstatus".
- Parameters:

❑ Response:

- The same data sent in the request. Response status will include error elements if the cancelreserve function failed for one or more objects.

6 Verify that the response from the call to cancelreserve does not contain errors.

Samples

The following sample walks through the cancel check out use case:

```
public PlmResponse cancelCheckOut(PlmSession session) throws PlmException {
      PlmRequest request = null;
      try {
         request = PlmFactory.createRequest(new URL("file:/D:/Data/
ADW_PreMapped_Footprint_1.xml"));
```

Is this request also valid for Cancel CheckOut? If not, please provide a new request in the Appendix A.

```
} catch (Exception ex) {
         throw new PlmException("Invalid URL or request format.", ex);
      }
      PlmResponse response = session.execute("mapping", request);
      // check to see if response is of type success
      if (!response.getStatus().isSuccess()) {
```

```
        System.out.println("Calling to 'mapping' function failed. Details are shown
below:");
          System.out.println(response.getStatus().getAllStatuses());
          return response;
       }
       request = PlmFactory.createRequest(response, "checkstatus");
       response = session.execute("checkstatus", request);
       // check to see if call to check-status has errors
       if (response.getStatus().hasErrors()) {
        System.out.println("Calling to 'check-status' has errors. Details are shown
below:");
          System.out.println(response.getStatus().getErrors());
          return response;
       }
       request = PlmFactory.createRequest(response, "cancelreserve");
       response = session.execute("cancelreserve", request);
       // check to see if call to cancelreserve has errors
       if (response.getStatus().hasErrors()) {
          System.out.println("Calling to 'cancelreserve' has errors. Details are
shown below:");
          System.out.println(response.getStatus().getErrors());
          return response;
       }
       return response;
    }
```

Change Synchronization (Input) / Save Change

Goal

The goal of this use case is to create a change object in Agile of change the status of a change object in Agile to reflect the status of a LCR in the PDM system.

Flow

1 Call function “createchange”

This function creates or updates a change object and creates entries for any file attachments (which need to have been previously uploaded to the DFM) and adds or updates any affected items of the change.

The function processes objects of type "change" with isdirty=true, valid name=subclass, valid "Number" attribute. It also processes any objects of type "object" or "file" that are related to "change" via a valid change-object-relation or change-file-relation.

Request:

- A "change oriented" mapped request which consists of changes and files and objects and relations from the changes to the file and objects. These relations are of the type “change-object-relation” and “change-file-relation”.
  For each change object in the request:
  - If the change object includes the attribute mapped to CoverPage.Number (usually called “Number”), then an attempt is made to find a change object with that number and if one is found, then it is updated.
  - If the change object doesn’t include the “Number” attribute (or the one mapped to CoverPage.Number according to the mapping configuration), then a new change object is created.
- Parameters:
  - logginglevels (possible values are: note, warn, noerrors, nofatals).

❑ Response:

- The same data included in the request with extra header parameters and status section. Also, adds "Number" attribute to all objects representing created changes.
- Status may contain notes, warnings, errors or fatals.

2 Verify that the createchange function returned a success response.

3 Call function “movechange”

This step is optional, and is usually executed if the change object(s) created or updated in the previous steps need to be moved to a new status in the Workflow.

This function moves the specified change object to the specified "NextStatus"; it processes objects of type "change" with isdirty=true, valid name=subclass, valid "Number" attribute, and valid "NextStatus" attribute.

- Request:
  - A "change oriented" mapped request which consists of objects of type = "change" only.
  - Parameters:
    - loggінglevels (possible values are: note, warn, noerrors, nofatals).
- Response:
  - The same data sent in the request. Response status will include error elements if the movechange function failed for one or more change objects.

4 Verify that the response does not contain errors.

Samples

The following sample walks through the change synchronization use case:

```
public PlmResponse synchronizeChange(PlmSession session) throws PlmException {
      PlmRequest request = PlmFactory.createRequest();
      PlmObject change = PlmFactory.createObject("Eng Lib ECO");
      change.setAttributeValue("Description", "Synchronized change order");
      change.setAttributeValue("NextStatus", "Submitted");
      PlmObject mechPart = PlmFactory.createObject("Mechanical Part");
      mechPart.setAttributeValue("Number", "MEP_35-0008-07_P");
      PlmRelation affItem = change.createRelation("change-object-relation",
mechPart);
      affItem.setAttributeValue("Revision", "$NN");
      affItem.setAttributeValue("Lifecycle Phase", "Prototype");
      PlmResponse response = session.execute("createchange", request);
      // check to see if response is of type success
      if (!response.getStatus().isSuccess()) {
        System.out.println("Calling to 'createchange' function failed. Details are
shown below:");
         System.out.println(response.getStatus().getAllStatuses());
         return response;
      }
      request = PlmFactory.createRequest(response, "movechange");
      response = session.execute("movechange", request);
      // check to see if call to movechange has errors
      if (response.getStatus().hasErrors()) {
        System.out.println("Calling to 'movechange' has errors. Details are shown
below:");
         System.out.println(response.getStatus().getErrors());
         return response;
      }
      return response;
   }
```

Change Synchronization (Output)

In order the extract the change information from the Agile system, you need to use ACS (Refer to "Configuring Agile Content Service" in the Agile Administrator Guide). You will want to set up the following:

- a destination that will receive the change information extracted from Agile
- a workflow event that will cause the changes to be extracted when they are released
- a filter to indicate the specific data from the change that you want to extract. Note that if you are extracting the affected items of the changes, you may also need to set up an item filter.
- a subscriber that pulls that all together and sets up the extraction of the changes.

Bulk Data Export

To periodically extract the item data from the Agile system, you will also need to use ACS. For this type of extract, you will want to set up the following:

- a destination that will receive the item information extracted from Agile
- a scheduled event that will cause the items to be extracted on a periodic basis. Note that after an item is extracted once by a specific subscriber, it will not be extracted again, unless it is modified.
- a filter to indicate the specific data from the item that you want to extract. Note that since you are extracting all items, when you specify the BOM tab options, you will want to specific BOM Tab Only (as opposed to any option that will extract the BOM and item data).
- a subscriber that pulls that all together and sets up the extraction of the items.

PLM API XML Request Samples

This section contains samples of XML files that can be used to create a request. The files are used as feeders for several of the samples provided in this document. When this is the case, the URL in the function PlmFactory.createRequest() will include the file name as shown below.

ADW_PreMapped_Footprint_1.xml

The following is a sample of the XML representation of an ADW footprint.

```
<plm:root xmlns:plm="http://www.agile.com/api/plm" version="2.0.0">
<plm:header type="request">
<plm:session id="9cc72bd2-cff0-43be-b2d7-9429f6db73ea"/>
<plm:operation name="translate"/>
<plm:param name="createdByUser" value="CAD User"/>
<plm:param name="createdDate" value="1177530554015"/>
<plm:param name="createdByTool" value="ADW2Agile"/>
<plm:param name="createdByToolVersion" value="0.0.001"/>
</plm:header>
<plm:data>
<!-- Layout Footprint before mapping -->
<plm:object name="" key="t5010_2_mech.psm" head="true" source="t5010_2_mech.psm"
sourceTool="Allegro 15.7" id="ef8f6cda-1694-4354-acf3-86a192568111">
<plm:attribute source="ADWXL attribute" isdirty="true"
name="ADW_MODEL_NAME">t5010_2_mech</plm:attribute>
<plm:attribute source="ADWXL attribute" isdirty="true"
name="ADW_LIFECYCLE_STATUS">RELEASED</plm:attribute>
<plm:attribute source="ADWXL attribute" isdirty="true" name="ADW_REVISION">1.0</
plm:attribute>
<plm:attribute source="ADWXL attribute" isdirty="true"
name="ADW_ORIGINATOR">mkoots</plm:attribute>
```

```
<plm:attribute source="ADWXL attribute" isdirty="true"
name="ADW_CREATION_DATE">1177530554015</plm:attribute>
<plm:attribute source="ADWXL attribute" isdirty="true"
name="ADW_MODIFICATION_DATE">1177530554015</plm:attribute>
<plm:attribute source="ADWXL attribute" isdirty="true"
name="ADW_MODIFICATION_USER">cad_user</plm:attribute>
<plm:attribute source="ADWXL attribute" isdirty="true" name="ADW_COMMENT">Some
Comment about part</plm:attribute>
</plm:object>
<!-- Padstack before mapping -->
<plm:object name="" key="t_pad2.pad" head="true" source="t_pad2.pad"
sourceTool="Allegro 15.7" id="ef8f6cda-1694-4354-acf3-86a192568112">
<plm:attribute source="ADWXL attribute" isdirty="true"
name="ADW_MODEL_NAME">t_pad2</plm:attribute>
<plm:attribute source="ADWXL attribute" isdirty="true"
name="ADW_LIFECYCLE_STATUS">RELEASED</plm:attribute>
<plm:attribute source="ADWXL attribute" isdirty="true" name="ADW_REVISION">1.0</
plm:attribute>
<plm:attribute source="ADWXL attribute" isdirty="true"
name="ADW_ORIGINATOR">mkoots</plm:attribute>
<plm:attribute source="ADWXL attribute" isdirty="true"
name="ADW_CREATION_DATE">1177530554015</plm:attribute>
<plm:attribute source="ADWXL attribute" isdirty="true"
name="ADW_MODIFICATION_DATE">1177530554015</plm:attribute>
<plm:attribute source="ADWXL attribute" isdirty="true"
name="ADW_MODIFICATION_USER">cad_user</plm:attribute>
<plm:attribute source="ADWXL attribute" isdirty="true" name="ADW_COMMENT">Some
Comment about part</plm:attribute>
</plm:object>
<!-- Flash before mapping -->
<plm:object name="" key="t_pad2_n.fsm" head="true" source="t_pad2_n.fsm"
sourceTool="Allegro 15.7" id="ef8f6cda-1694-4354-acf3-86a192568113">
<plm:attribute source="ADWXL attribute" isdirty="true"
name="ADW_MODEL_NAME">t_pad2_n</plm:attribute>
<plm:attribute source="ADWXL attribute" isdirty="true"
name="ADW_LIFECYCLE_STATUS">RELEASED</plm:attribute>
<plm:attribute source="ADWXL attribute" isdirty="true" name="ADW_REVISION">1.0</
plm:attribute>
<plm:attribute source="ADWXL attribute" isdirty="true"
name="ADW_ORIGINATOR">mkoots</plm:attribute>
<plm:attribute source="ADWXL attribute" isdirty="true"
name="ADW_CREATION_DATE">1177530554015</plm:attribute>
<plm:attribute source="ADWXL attribute" isdirty="true"
name="ADW_MODIFICATION_DATE">1177530554015</plm:attribute>
<plm:attribute source="ADWXL attribute" isdirty="true"
name="ADW_MODIFICATION_USER">cad_user</plm:attribute>
<plm:attribute source="ADWXL attribute" isdirty="true" name="ADW_COMMENT">Some
Comment about part</plm:attribute>
</plm:object>
<!-- Shape before mapping -->
<plm:object name="" key="t_pad2_n.ssm" head="true" source="t_pad2_n.ssm"
sourceTool="Allegro 15.7" id="ef8f6cda-1694-4354-acf3-86a192568114">
<plm:attribute source="ADWXL attribute" isdirty="true"
name="ADW_MODEL_NAME">t_pad2_n</plm:attribute>
<plm:attribute source="ADWXL attribute" isdirty="true"
name="ADW_LIFECYCLE_STATUS">RELEASED</plm:attribute>
<plm:attribute source="ADWXL attribute" isdirty="true" name="ADW_REVISION">1.0</
plm:attribute>
<plm:attribute source="ADWXL attribute" isdirty="true"
name="ADW_ORIGINATOR">mkoots</plm:attribute>
```

```
<plm:attribute source="ADWXL attribute" isdirty="true"
name="ADW_CREATION_DATE">1177530554015</plm:attribute>
<plm:attribute source="ADWXL attribute" isdirty="true"
name="ADW_MODIFICATION_DATE">1177530554015</plm:attribute>
<plm:attribute source="ADWXL attribute" isdirty="true"
name="ADW_MODIFICATION_USER">cad_user</plm:attribute>
<plm:attribute source="ADWXL attribute" isdirty="true" name="ADW_COMMENT">Some
Comment about part</plm:attribute>
</plm:object>
<!-- Footprint-Padstack Relation -->
<plm:relation name="" key-parent="t5010_2_mech.psm" key-child="t_pad2.pad"
isdirty="true" mapped="true" source="\\ADW\psm\t5010_2_mech.psm"
sourceTool="Allegro 14.2" parent="ef8f6cda-1694-4354-acf3-86a192568111"
child="ef8f6cda-1694-4354-acf3-86a192568112"/>
<!-- Padstack-Flash Relation -->
<plm:relation name="" key-parent="t_pad2.pad" key-child="t_pad2_n.fsm"
isdirty="true" mapped="true" source="\\ADW\psm\t_pad2.pad" sourceTool="Allegro
14.2" parent="ef8f6cda-1694-4354-acf3-86a192568112" child="ef8f6cda-1694-4354-
acf3-86a192568113"/>
<!-- Padstack-Shape Relation -->
<plm:relation name="" key-parent="t_pad2.pad" key-child="t_pad2_n.ssm"
isdirty="true" mapped="true" source="\\ADW\psm\t_pad2.pad" sourceTool="Allegro
14.2" parent="ef8f6cda-1694-4354-acf3-86a192568112" child="ef8f6cda-1694-4354-
acf3-86a192568114"/>
</plm:data>
</plm:root>
```

CHAPTER 2
Translation and Mapping

This chapter describes the configuration required to convert the CAD structures to Agile PLM structures, which is called "structural mapping". It also shows the possible transformations that can be applied to the names and values of the attributes to adapt them to the Agile PLM Server settings. This chapter contains the following topics:

- *Introduction*
- *Structure Mapping*
- *Subclass Mapping*
- *Attribute Mapping*
- *Translation*

Introduction

Mapping is a process where a CAD structure (with different types of objects) is converted to Agile destination data. Mapping from CAD to Agile is handled by the MappingCallable on the server side, which is an extension of the PLM API, invoked through the call to the function "mapping" from within a PLM API application.

The actual structural mapping must be defined in a mapping file, which is an XML file compliant with the map.xsd schema definition file, whose contents are shown at the end of this section.

There is no limit in the number of XML mapping files that the user may create, so they can be used to accommodate the needs of different CAD integrations. During the call to the mapping function, the appropriate mapping file is selected based on the data included in the header of the request when this PLM API function is called.

Because different mapping files are associated with different CAD related projects inside a CAD integration application, these mapping files are called Project Mappings.

In order to select the appropriate Project Mapping file, the MappingCallable relies on a set of rules defined in one file, called mastermapping.txt. This "Master Mapping" file is stored in a predetermined FileFolder in the target A9 PLM Server; The name of this FileFolder is, in turn, defined in the general PLM API configuration file (see the PLM API User's Guide for more details about the PLM API configuration file).

Inside this file, the keys: mapping.folder and mapping.file specify the number of the FileFolder and the name of the Master Mapping file attached to it. The default folder is PLMAPI_MAPPING.

The Master Mapping file consists of a sequence of rules which are evaluated from top to bottom until one of them matches the entries in the request's header, at which time, the corresponding project mapping is selected. The rules in the Master Mapping are of this form:

```
PLMAPI_MAPPING,ProjectMapping_CF.xml: createdByUser="*" && createdByTool="adw*" &&
createdByToolVersion="*" && createdForProject="project*"
```

The first two entries determine the number of the FileFolder and the name of the file that is attached to it, separated by a comma (,) and followed by a colon (:).

To the right side of the colon is a logical expression that refers to the parameters in the request's header and the values to which to match them. The rule shown above, as an example, states that the Project Mapping file called "ProjectMapping_CF.xml", which is attached to the FileFolder number "PLMAPI_MAPPING", should be used when the parameter createdByTool starts with "adw" and the parameter "createdForProject" starts with "project", regardless of the values of the parameters "createdByUser" and "createdByToolVersion", (note that the star character (*) means that any value of these two parameters will match the rule).

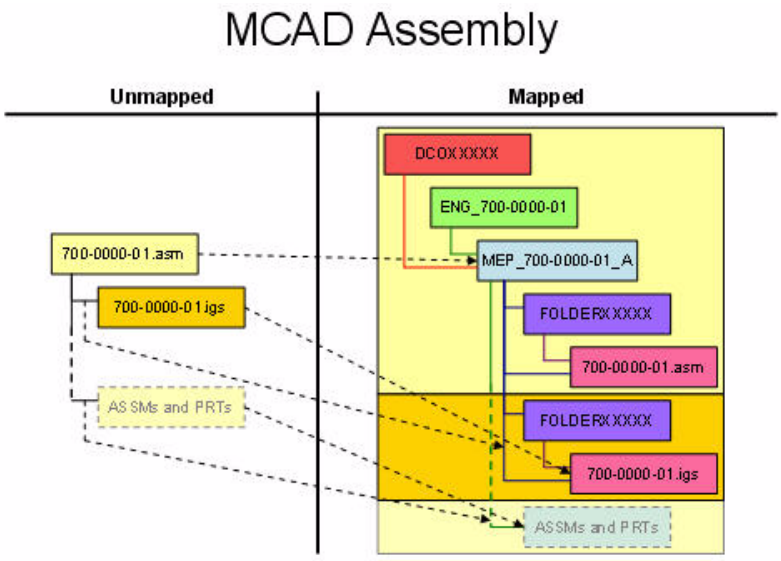
Once the project mapping file has been determined, it is loaded into memory to drive the actual mapping process. As mentioned above, the mapping process will take the original PlmRequest that was sent to the PLM API "mapping" function and will return a PlmResponse whose data contains a structure of objects that arise from applying the following three steps to the incoming request's data:

1. **Structure mapping**: Each object in the request's data is analyzed and converted to a structure of objects with relations binding them together.
2. **Subclass mapping**: Each object in the resulting structure is assigned a subclass based on the rules specified in the mapping file.
3. **Attribute mapping**: The attributes of the objects in the incoming request's data are translated to attributes of the objects in the generated structure, optionally modifying their name and/or value. (The process of modifying the name and value of the attributes is called "Translation" and is discussed in greater detail in the following section).

Structure Mapping

The first step in the mapping process is the structure mapping, which consists on analyzing the incoming "unmapped" structure and generating a "mapped" structure which contains objects related to the A9 Data Model. The diagram below shows a sample of the unmapped structure and the resulting mapped structure for a Mechanical CAD assembly.

Figure 2-1: Unmapped and mapped structures of a mechanical CAD Assembly



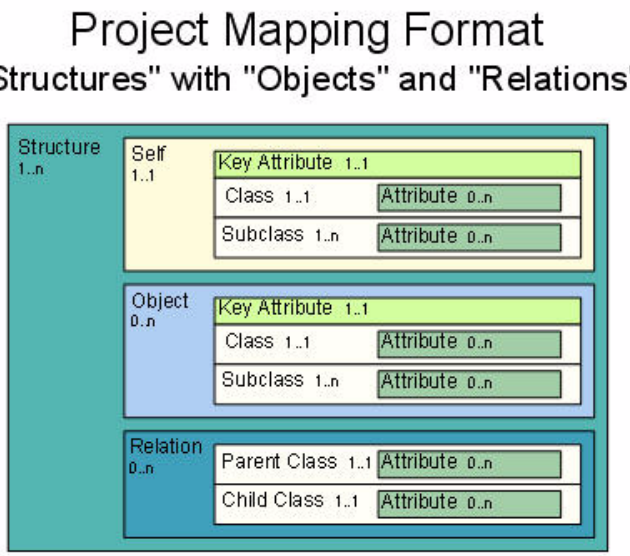
The diagram above shows three colored sections from top to bottom, starting with a light yellow section on top, then a mustard section in the middle and then a light yellow section at the bottom. These sections illustrate the fact that each single object of the unmapped structure may be converted to a structure of objects with relations binding them. These structures will be part of the complete mapped structure that will be returned in the PlmResponse, so to differentiate these structures from the complete mapped structure, they will be referred to as "sub-structures".

The diagram also shows that each sub-structure includes one object which is the direct map of the original object itself, and optionally a number of additional objects with relations between them. In the picture above, the object identified as 700-000-01.asm is mapped itself to MEP_700-0000-01_A (light blue), with four additional objects and their relations.

Once each object has been converted to a sub-structure, the MappingCallable analyzes the relations between the objects in the original, unmapped structure and creates relations between the corresponding objects in the mapped structure. Note in the diagram above that there are arrows connecting the relations between the objects in the unmapped structure to the corresponding relations in the mapped structure.

The way in which each sub-structure is created from an object in the unmapped request is determined by the project mapping file. This file includes one or more nodes of type “map:structure”, whose structure is shown in the following diagram:

Figure 2-2: Structure Element Diagram



As shown above, each structure has one element describing how the original object will map to an object in the mapped structure. This is represented by the “self” element. Each structure must have one and only one “self” element.

A structure element can have 0 or more “object” elements, each of which represent one additional object in the mapped sub-structure. The relations between these objects will be defined by the relation elements.

As mentioned above, each project mapping file will most likely contain more than one structure element; to define how each different object in the unmapped structure will be converted to a sub-structure of the mapped structure, each structure element has a nested matchRule element. For each object in the unmapped structure, the mapping callable will scan the Project Mapping file searching for a structure element whose matchRule matches the current object. When this is the case, the sub-structure will be created based on the self, object(s) and relation(s) elements nested inside the structure element.

Below is shown an excerpt of a project mapping file defining one structure element. This structure element will convert the object present in the unmapped structure to a total of five objects (one “map:self” element plus four “map:object” elements). There are also six map:relation elements included in this structure, which will result in the same number of relations between the five objects in the mapped structure.

As defined by the map:matchRule element, this structure will be applied if the concatenation of the values of the attributes “filename” and “related_change” of the object in the unmapped structure matches the regular expression: ^.*\.*psmnull$*, that is, if the filename ends with “.psm” and the related_change attribute is not present (its value is “null”).

```
<map:structure name="Allegro Footprint Structure wo/LCR">
<map:matchRule source="filename+related_change" match="^.*\.psmnull$"/>
<map:self name="Allegro Object" head="true">
<map:key page="Title Block" name="Number" id="1001">
<map:expression source="filename" match="^(.*[\\/]|)([^\\/]+)\.psm$"
replace="DSFV_$2" toCase="upper"/>
</map:key>
<map:class name="Documents">
<map:target page="Title Block" name="Description" id="1002" isdirtyDefault="true">
<map:expression source="&amp;quot;typ=&amp;quot;+type+&amp;quot;,cls=&amp;quot;+class+&amp;quot;,
pkg=&amp;quot;+package+&amp;quot;,sct=&amp;quot;+Socket"/>
</map:target>
<map:target page="Title Block" name="Rev" id="1014">
<map:expression source="Title Block.Rev" refresh="true"/>
</map:target>
…
</map:class>
<map:subclass name="PCB Footprint">
<map:target page="Page Three" name="Designer" id="1540">
<map:expression source="Page Three.Designer" refresh="true"/>
</map:target>
…
</map:subclass>
</map:self>
<map:object name="ECPN" head="true" isdirty="false">
<map:key page="Title Block" name="Number" id="1001" split=";">
<map:expression source="part" match="([^;]+)" replace="ENG_$1"/>
</map:key>
<map:class name="Documents" limitRefresh="true">
<map:target name="role" option="true">
<map:expression replace="parent"/>
</map:target>
</map:class>
<map:subclass name="ECPN" limitRefresh="true"/>
</map:object>
<map:object name="Design Change Order" head="true">
<map:key page="Cover Page" name="Number" id="1047">
<map:expression replace=""/>
</map:key>
<map:class name="Change Orders">
<map:target name="autonumbersource" option="true">
<map:expression replace="Eng Lib"/>
</map:target>
<map:target name="ownerattribute" option="true">
<map:expression replace="1099"/>
</map:target>
…
</map:class>
<map:subclass name="Eng Lib ECO">
<map:target page="Page Three" name="Eng Change Category" id="1539">
<map:expression source="Page Three.Eng Change Category" refresh="true"/>
</map:target>
</map:subclass>
</map:object>
<map:object name="Allegro Footprint Folder" head="false">
<map:key page="Title Block" name="Number" id="6173">
```

```
<map:expression replace=""/>
</map:key>
<map:class name="File Folders">
<map:target name="autonumbersource" option="true">
<map:expression replace="File Folder Number"/>
</map:target>
<map:target page="Title Block" name="Type" id="6175">
<map:expression source="Title Block.Type" refresh="true"/>
</map:target>
…
</map:class>
<map:subclass name="Library Files"/>
</map:object>
<map:object name="Allegro Footprint File" head="false">
<map:key page="" name="file-name" id="">
<map:expression source="filename" match="^(.*[\\/]|)([^\\/]+)$" replace="$2"/>
</map:key>
<map:class name="Files">
<map:target page="" name="file-path" id="">
<map:expression source="filename" match="^(.*[\\/]|)([^\\/]+)$" replace="$1"/>
</map:target>
<map:target page="" name="file-id" id="">
<map:expression source="fileId"/>
</map:target>
</map:class>
<map:subclass name="file"/>
</map:object>
<map:relation parent="Allegro Object" child="Allegro Object" self="true">
<map:childClass name="Documents">
<map:target page="BOM" name="UOM" id="1343">
<map:expression source="BOM.UOM" refresh="true"/>
</map:target>
<map:target page="BOM" name="Item Number" id="1011">
<map:expression source="BOM.Item Number" refresh="true"/>
</map:target>
…
</map:childClass>
<map:parentClass name="Documents"/>
</map:relation>
<map:relation parent="ECPN" child="Allegro Object" self="false">
<map:childClass name="Documents" limitRefresh="true"/>
<map:parentClass name="Documents" limitRefresh="true"/>
</map:relation>
<map:relation parent="Allegro Object" child="Design Change Order" self="false">
<map:childClass name="Change Orders">
<map:target page="Affected Items" name="New Rev" id="1056">
<map:expression replace="$NN"/>
</map:target>
…
</map:childClass>
<map:parentClass name="Documents"/>
</map:relation>
<map:relation parent="ECPN" child="Design Change Order" self="false">
<map:childClass name="Change Orders" limitRefresh="true"/>
<map:parentClass name="Documents" limitRefresh="true"/>
</map:relation>
<map:relation parent="Allegro Object" child="Allegro Footprint File" self="false">
<map:childClass name="Files"/>
<map:parentClass name="Documents">
<map:target page="Attachments" name="File Description" id="1045">
<map:expression source="Attachments.File Description" refresh="true"/>
```

```
</map:target>
…
</map:parentClass>
</map:relation>
<map:relation parent="Allegro Footprint File" child="Allegro Footprint Folder"
self="false">
<map:childClass name="File Folders">
<map:target page="Files" name="File Type" id="6305">
<map:expression source="Files.File Type" refresh="true"/>
</map:target>
…
</map:childClass>
<map:parentClass name="Files"/>
</map:relation>
     </map:structure>
```

By defining an appropriate number of structure elements in a Project Mapping file, the user can create a complex structural mapping to cover the needs of the CAD integration for a complete CAD project.

Subclass mapping

As mentioned before, each of the objects in the mapped structure that results of the call to the mapping function will be a close match to an object in the A9 data model. This includes the fact that the subclass of these objects will be already defined.

The PLM API uses the A9 subclass names as the entity names for the objects in the data section of a PlmRequest or a PlmResponse. Because of this, the mapped structure's objects names match the A9 subclass names, such as, for example:

```
<plm:object name="PCB Footprint " …>
```

or:

```
<plm:object name="ECPN" …>
```

where “PCB Footprint” and “ECPN” are subclasses of the Documents or Parts classes in a9. The mapping callable determines the name of the subclass from the subclass element inside the “self” or “object” elements of a structure.

In the excerpt shown below, one can notice that the object in the unmapped structure will itself be converted to an object of subclass “PCB Footprint”, according to the subclass element underneath the self element, as shown below:

```
<map:self name="Allegro Object" head="true">
…
<map:subclass name="PCB Footprint">
…
</map:subclass>
</map:self>
```

Attribute mapping

The last step in the structural mapping is the attribute mapping. Once the mapping callable has determined the objects that will be included in the mapped structure, including their subclass and the relations that will bind them together, the mapping will assign attributes to both objects and relations of the mapped sub-structure according to the target elements in the project mapping file. The target elements are the attributes shown on the previous diagram.

The attributes of the objects and relations in the mapped structure come in most of the cases from the objects and relations in the unmapped structure, however, there are certain special cases where an attribute may originate from a special process, such as, for example, when an object number should be generated by using the autonumber feature or when a change status should be changed within a workflow.

Additionally, the attributes of the objects and relations in the unmapped structure may go through a translation process where their name and/or value may be modified before they become attributes of the objects or relations of the mapped structure.

All aspects of the attribute mapping are governed by two types of elements present in the project mapping file: “key” and “target”; the special attributes are handled by the use of predefined constants, such as “autonumbersource” or “ownerattribute” and the translation is defined by the use of regular expressions which are parsed and evaluated by the mapping callable.

The following excerpt shows the use of the map:key element and map:target element to demonstrate how the attribute mapping/translation is defined:

```
<map:self name="Allegro Object" head="true">
<map:key page="Title Block" name="Number" id="1001">
<map:expression source="filename" match="^(.*[\\/]|)([^\\/]+)\.psm$"
replace="DSFV_$2" toCase="upper"/>
</map:key>
<map:class name="Documents">
<map:target page="Title Block" name="Description" id="1002" isdirtyDefault="true">
<map:expression source="&amp;quot;typ=&amp;quot;+type+&amp;quot;,cls=&amp;quot;+class+&amp;quot;,
pkg=&amp;quot;+package+&amp;quot;,sct=&amp;quot;+Socket"/>
</map:target>
...
</map:class>
...
</map:self>
```

Note how the map:key and map:target elements are defined. These XML elements have attributes which specify what the target attribute in the mapped structure is. In the first case, the name of the target attribute is “Number” and In the second case, the name of the target attribute is “Description”. Their IDs and table names are also populated in the mapped structure to 1001, 1002 and TitleBlock (in both cases) respectively.

The translation is defined in the enclosed map:expression element. This element defines how the value of the attribute in the mapped structure will be calculated, either from one or more attributes of the object in the unmapped structure or by using special predefined constants that will be evaluated at runtime.

In general, the value of the target attribute could be understood as:

```
target <- function(source)
```

where “function” is the Java implementation of String.replaceAll(match, replace), which uses Regular Expressions. For more information about the use of Regular Expression in Java, please refer to: http://java.sun.com/j2se/1.4.2/docs/api/java/util/regex/Pattern.html#sum.

Usually, the expression element will contain the following XML attributes: source, match and replace. The value of the source XML attribute can point to the name of an attribute in the source object, or a predefined constant or a concatenation of names of attributes in the source object. The match XML attribute contains a regular expression that will be sought for and the value of replace will determine what to replace the matching expression with. In the java String.replaceAll(match,replace) method "source" from the map:expression becomes the String, "match" from the map:expression becomes match, and "replace" becomes replace. The details of this behavior are discussed in detail in the following section on Translation.

Translation

Basic Translation

In many cases unmapped attributes already have the correct value for A9 and simply need to be assigned to an A9 attribute. In this case only the source is needed in the expression in the target. Here is an example:

```
<map:target page="Title Block" name="Description" id="1002">
```

```
<map:expression source="desc"/>
</map:target>
```

In this example the value of the CAD attribute "desc" will copied to the target A9 attribute Description. Note that the expression does not have a match value or a replace value.

The value of the source in the map:expression is itself a simple expression. The source expression can concatenate CAD attribute values and literal strings. A CAD attribute value is referenced in the expression using the CAD attribute's name like "desc" shown above. Literal strings in the source expression must be contained in "quotes". In a source expression a literal string looks like this source=""My Literal String"". The strange looking character sequence " is how a double quote (") is represented in an xml value. When this literal string is assigned to an A9 attribute it will be "My Literal String". The plus sign is used to delimit two values that should be concatenated. For example to concatenate the value of the CAD attribute desc and string literal "My Literal String" you would use source="desc+"My Literal String"". If the value of the CAD attribute desc is "Bracket" then the resulting value of the source expression will be "BracketMy Literal String".

The resulting value of the expression can also be forced to upper or lower case or truncated to a specific length.

To change the case add the xml attribute "toCase" to the expression. The attribute toCase can have the value of "upper" or "lower". To make the resulting value upper case you would add toCase="upper" to the expression like this:

```
<map:expression source="desc" toCase="upper"/>
```

If the value of the CAD attribute desc is "Bracket" then the resulting value of the above expression would be "BRACKET".

To limit the length of an attribute you can add the xml attribute "maxLength" to the expression. The value of maxLength must be a positive whole number. To truncate the desc attribute to 5 characters you would use this expression:

```
<map:expression source="desc" maxLength="5"/>
```

If the value of the CAD attribute desc is "Bracket" then the resulting value of the above expression would be "Brack".

If you simply want to assign a string literal to an A9 attribute you should use replace instead of source. In a similar way to the source string literal example above, you can also specify a string literal in a map expression using the xml attribute "replace". Both of these expressions will have the same resulting value.

```
<map:expression source="&amp;quot;My Literal String&amp;quot;"/>
<map:expression replace="My Literal String"/>
```

The second expression is preferable because it is easier to read and less complicated. Note that when replace is specified without source and match, replace's literal value will be the value of the expression.

Advanced Translation

The map expression has three primary attributes source, match, and replace and looks like this when they are all included:

```
<map:expression source="filename" match="^(.*[\\/]|)([^\\/]+)\.psm$"
replace="PSM_$2"/>
```

Each of these xml attributes source, match, and replace are optional and the behavior will change depending on which of the three attributes are defined in the map expression.

Here are the steps that are taken to get the value of the expression:

1. If source is not defined then the literal value of replace is used. If neither source nor replace is defined then the resulting value will be null.
2. If source is defined and match is not defined then source is evaluated as a simple expression described above in Basic Translation.
3. If source and match are defined but replace is not then the resulting value is null.

4 If all three xml attributes source, match, and replace are defined then the java method String.replaceAll(match, replace) is used to evaluate the expression, where String is the result of source evaluated as a simple expression described above in Basic Translation.

If we use the above map expression as an example and assume that the value of the filename attribute in the CAD data is c:\data\resistor.psm then the resulting value will be PSM_resistor. Here is a more detailed description of how the map expression was evaluated. Because all three xml attributes are defined the expression will be evaluated with step 4. When an expression is evaluated using this step the following sub-steps are performed:

1 First the simple source expression is evaluated. In this case evaluating the source attribute involves only getting the value of the CAD attribute "filename".

2 Second this value is the string used to execute String.replaceAll(match,replace) and the match and replace xml attributes are used for the replaceAll match and replace parameters. If we break the match string "^(.*[\\/])([^\\/]+)\.psm$" into several smaller pieces it will be easier to understand. The '^' matches the beginning of the string and the '$' matches the end of the string, so we are trying to match and replace the entire string. (.*[\\/]) matches any characters followed by either a back slash '\' or a forward slash '/'. It matches the path portion of the filename which is "c:\data\". The enclosing parentheses mark this as a Capture Group which can be used later for replacement. ([^\\/]+) matches one or more characters that are not a back slash '\' or a forward slash '/'. This matches the filename portion of the filename which is "resistor". \.psm matches a period '.' followed by "psm". This is the extension portion of the filename. At this point both the path and the filename have been captured and can be used in replacement.

3 Finally the replace value is used to replace any matched sections of the source string. Since we matched the entire string the entire string will be replaced. Our replacement string is "PSM_$2". "PSM_" is a string literal and is placed as in the result. $2 is a special marker telling java to use the second Capture Group at this point in the string. The second capture group is the filename "resistor". Putting these together we have the final result "PSM_resistor".

Appendix A: map.xsd

Following are the contents of the file map.xsd, which is used to verify the strucure of the project mapping XML files:

<?xml version="1.0" encoding="UTF-8"?>

<xs:schema targetNamespace="http://www.agile.com/api/map" xmlns:map="http://www.agile.com/api/map" xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="unqualified">

```
<xs:simpleType name="toCaseType">
<xs:restriction base="xs:string">
<xs:enumeration value="upper"/>
<xs:enumeration value="lower"/>
</xs:restriction>
</xs:simpleType>
<xs:complexType name="expressionType">
<xs:attribute name="source" type="xs:string" use="optional"/>
<xs:attribute name="match" type="xs:string" use="optional"/>
<xs:attribute name="replace" type="xs:string" use="optional"/>
<xs:attribute name="toCase" type="map:toCaseType" use="optional"/>
<xs:attribute name="maxLength" type="xs:integer" use="optional"/>
<xs:attribute name="delimiter" type="xs:string" use="optional"/>
<xs:attribute name="quote" type="xs:string" use="optional"/>
<xs:attribute name="ancestorFlag" type="xs:string" use="optional"/>
<xs:attribute name="refresh" type="xs:boolean" use="optional"/>
<!-- "source" names an attribute from the source object to use for this expression.
-->
<!-- "match" defines the java regular expression matching string. -->
<!-- "replace" defines the java regular expression replacement string. -->
```

```
<!-- "toCase" will force the expression result to the specified case. -->
<!-- "maxLength" will truncate the expression result to the length specified. -->
<!-- "delimiter" is used to concatonate values when building a more complex source
string. The default is + -->
<!-- "quote" is used to mark the beginning and end of explicit values in the source.
The default is " -->
<!-- "ancestorFlag" marks an attribute name that is an xml attribute on the parent
object element. -->
<!-- "refresh" equal to true says this expression should be updated automatically
by refresh. -->
</xs:complexType>
<xs:complexType name="expressionOptionType">
<xs:sequence>
<xs:element name="expression" type="map:expressionType" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="targetType">
<xs:choice>
<xs:element name="expression" type="map:expressionType"/>
<xs:element name="expressionOption" type="map:expressionOptionType"/>
</xs:choice>
<xs:attribute name="page" type="xs:string" use="optional"/>
<xs:attribute name="name" type="xs:string" use="optional"/>
<xs:attribute name="id" type="xs:string" use="optional"/>
<xs:attribute name="keepIfErr" type="xs:boolean" use="optional"/>
<xs:attribute name="option" type="xs:boolean" use="optional"/>
<xs:attribute name="isdirtyDefault" type="xs:boolean" use="optional"/>
<xs:attribute name="isdirty" type="xs:boolean" use="optional"/>
<!-- "page" the tab name for the target attribute in Agile. Title Block for example.
-->
<!-- "name" the name of the target attribute in Agile. Number for example. -->
<!-- "id" the base id of the target attribute in Agile. 1001 for example. -->
<!-- "keepIfErr" include the attribute in the mapped data even if there is an error.
The default is false -->
<!-- "option" create an option in the mapped request instead of an attribute. The
default is false -->
<!-- "isdirtyDefault" value to use for isDirty. The default default is false -->
<!-- "isdirty" value to use for isDirty. -->
</xs:complexType>
<xs:complexType name="keyType">
<xs:complexContent>
<xs:extension base="map:targetType">
<xs:attribute name="split" type="xs:string" use="optional"/>
</xs:extension>
</xs:complexContent>
</xs:complexType>
<xs:complexType name="subclassType">
<xs:sequence>
<xs:element name="matchRule" type="map:expressionType" minOccurs="0"/>
<xs:element name="target" type="map:targetType" minOccurs="0"
maxOccurs="unbounded"/>
</xs:sequence>
<xs:attribute name="name" type="xs:string" use="required"/>
<xs:attribute name="limitRefresh" type="xs:boolean" use="optional"/>
<!-- "name" of the Agile subclass of the target object. Must match a subclass in
Agile -->
<!-- "limitRefresh" turns off the addition of attributes to this object during
refresh -->
</xs:complexType>
<xs:simpleType name="classNameType">
<xs:restriction base="xs:string">
```

```
<xs:enumeration value="Activities"/>
<xs:enumeration value="Change Orders"/>
<xs:enumeration value="Change Requests"/>
<xs:enumeration value="Documents"/>
<xs:enumeration value="File Folders"/>
<xs:enumeration value="Files"/>
<xs:enumeration value="Manufacturers"/>
<xs:enumeration value="Manufacturer Parts"/>
<xs:enumeration value="Parts"/>
</xs:restriction>
</xs:simpleType>
<xs:complexType name="classType">
<xs:sequence>
<xs:element name="target" type="map:targetType" minOccurs="0"
maxOccurs="unbounded"/>
</xs:sequence>
<xs:attribute name="name" type="map:classNameType" use="required"/>
<xs:attribute name="limitRefresh" type="xs:boolean" use="optional"/>
<!-- "name" of the class of the target object. Must match a class in list above -->
<!-- "limitRefresh" turns off the addition of attributes to this object during
refresh -->
</xs:complexType>
<xs:complexType name="objectType">
<xs:sequence>
<xs:element name="key" type="map:keyType"/>
<xs:element name="secondaryKey" type="map:keyType" minOccurs="0"/>
<xs:element name="class" type="map:classType"/>
<xs:element name="subclass" type="map:subclassType" maxOccurs="unbounded"/>
</xs:sequence>
<xs:attribute name="name" type="xs:string" use="required"/>
<xs:attribute name="head" type="xs:boolean" use="optional"/>
<xs:attribute name="isdirtyDefault" type="xs:boolean" use="optional"/>
<xs:attribute name="isdirty" type="xs:boolean" use="optional"/>
<xs:attribute name="ignore" type="xs:boolean" use="optional"/>
<!-- "name" is used by the relations' parent and child attributes -->
<!-- "head" can be used to force a value for the head flag. The default is true -->
<!-- "isdirtyDefault" value to use for isdirty. The default default is false -->
<!-- "isdirty" value to use for isdirty. -->
<!-- "ignore" value to use for ignore -->
</xs:complexType>
<xs:complexType name="relationType">
<xs:all>
<xs:element name="childClass" type="map:classType"/>
<xs:element name="parentClass" type="map:classType"/>
</xs:all>
<xs:attribute name="parent" type="xs:string" use="required"/>
<xs:attribute name="child" type="xs:string" use="required"/>
<xs:attribute name="sibling" type="xs:string" use="optional"/>
<xs:attribute name="self" type="xs:boolean" use="optional"/>
<xs:attribute name="isdirtyDefault" type="xs:boolean" use="optional"/>
<xs:attribute name="isdirty" type="xs:boolean" use="optional"/>
<!-- "parent" must match an object name in the same structure -->
<!-- "child" must match an object name in the same structure or the self in another
structure -->
<!-- "sibling" names a object other than self in structure of the child -->
<!-- "self" true if the relation exists in the original request. The default is
false -->
<!-- "isdirtyDefault" default value to use for isDirty. The default default is false
-->
<!-- "isdirty" value to use for isDirty. -->
</xs:complexType>
```

```
<xs:complexType name="structureType">
<xs:sequence>
<xs:element name="matchRule" type="map:expressionType" minOccurs="0"/>
<xs:element name="self" type="map:objectType"/>
<xs:element name="object" type="map:objectType" minOccurs="0"
maxOccurs="unbounded"/>
<xs:element name="relation" type="map:relationType" minOccurs="0"
maxOccurs="unbounded"/>
</xs:sequence>
<xs:attribute name="name" type="xs:string" use="required"/>
</xs:complexType>
<xs:complexType name="addTypeOptionType">
<xs:attribute name="value" type="xs:boolean" use="required"/>
<!-- "value" true to include Type options on objects. -->
</xs:complexType>
<xs:complexType name="forceIsDirtyType">
<xs:attribute name="value" type="xs:boolean" use="required"/>
<xs:attribute name="attribute" type="xs:string" use="required"/>
<xs:attribute name="split" type="xs:string" use="optional"/>
<!-- "value" to the given value. must be accompanied by a value with a list of
attribute ids or names -->
<!-- "attributes" give an attribute ID that should always be forced to isDirty=False
-->
<!-- "split" is flag and delimiter used to specify that the attribute should be
split into several values -->
</xs:complexType>
<xs:complexType name="globalRulesType">
<xs:sequence>
<xs:element name="addTypeOption" type="map:addTypeOptionType" minOccurs="0"/>
<xs:element name="forceIsDirty" type="map:forceIsDirtyType" minOccurs="0"
maxOccurs="2"/>
</xs:sequence>
</xs:complexType>
<xs:element name="root">
<xs:complexType>
<xs:sequence>
<xs:element name="globalRules" type="map:globalRulesType" minOccurs="0"/>
<xs:element name="structure" type="map:structureType" minOccurs="0"
maxOccurs="unbounded"/>
</xs:sequence>
<xs:attribute name="version" type="xs:string" use="required"/>
<!-- "version" is the mapping xml version -->
</xs:complexType>
</xs:element>
</xs:schema>
```