

# Connector for Mentor Graphics Expedition Enterprise

Administrator's Guide

3DEXPERIENCE R2020x



**3DEXPERIENCE** Platform is based on the V6 Architecture © Copyright Dassault Systèmes, 2014-2019. All rights reserved.

The **3DEXPERIENCE** Platform for 2020x is protected by certain patents, trademarks, copyrights, and other restricted rights, the full list of which is available at the 3DS support site: <http://help.3ds.com/>.

Certain portions of the **3DEXPERIENCE** Platform R2020x contain elements subject to copyright owned by third party, the full list of which is also available at the 3DS support site mentioned above.

You will require an account with support in order to view this page. From the support page, select your desired product version and language to launch the appropriate help. Select **Legal Notices** from left frame. This displays the full list of patents, trademarks and copyrights for this product.

Any copyrights not listed belong to their respective copyrights owners.

# Table of Contents

<b>TABLE OF CONTENTS</b> .....	<b>3</b>
<b>LIST OF FIGURES</b> .....	<b>6</b>
<b>SERVER CONNECTOR INSTALLATION</b> .....	<b>7</b>
EXECUTE SETUP .....	7
<i>3DEXPERIENCE Products</i> .....	7
INSTALLATION STEPS .....	8
SILENT INSTALLATION .....	8
UPDATE CONNECTOR STRING PROPERTIES .....	9
<b>CLIENT CONNECTOR INSTALLATION</b> .....	<b>10</b>
SYSTEM REQUIREMENTS .....	10
<i>Mentor Environment</i> .....	10
<i>3DEXPERIENCE Products</i> .....	10
<i>Connector for Mentor Graphics Expedition Enterprise</i> .....	10
<i>Java Environment</i> .....	10
EXECUTE SETUP .....	11
<i>Windows Platforms</i> .....	11
INSTALLATION STEPS – CONNECTOR FOR MENTOR GRAPHICS EXPEDITION ENTERPRISE .....	11
<i>SITE based Local Connector Configuration</i> .....	12
INSTALLATION STEPS - CONNECTOR FOR MENTOR GRAPHICS EXPEDITION ENTERPRISE DXDATABOOK .....	12
<b>GLOBAL CONFIGURATION OBJECT CLIENT SETTINGS</b> .....	<b>13</b>
GROUP ID .....	13
DERIVED OUTPUT PARAMETER FILE .....	14
CHECKIN REVISION BEHAVIOR .....	16
ROLE BASED COMMANDS .....	17
ENFORCE FOLDER ON SAVE .....	18
GCO ATTRIBUTES USED FOR USER SETTINGS .....	19
<b>BOM DATA MANAGEMENT</b> .....	<b>20</b>
UI ELEMENTS .....	20
<i>Component Search Dialog</i> .....	20
Type Search Dialog .....	21
Policy Search Dialog .....	22
State Search Dialog .....	22
Operator search .....	23
<i>Webform Element Definition</i> .....	23
typechooser .....	23
policychooser .....	24
statechooser .....	24
vaultchooser .....	24
revision .....	24
latestrevision .....	24

textfield .....	24
checkbox .....	24
Table Columns .....	25
UI Element cache refresh .....	25
BILL OF MATERIAL CREATION .....	26
MATERIAL EXISTENCE CHECK .....	26
<b>BILL OF MATERIAL CLIENT CONFIGURATION .....</b>	<b>27</b>
[GENERAL] SECTION .....	28
Design BOM creation .....	28
BOM Settings .....	28
Missing Part Strategy .....	30
Checkin Behavior after Save to 3DEXPERIENCE .....	31
[UI] SECTION .....	32
BOM Table Settings .....	32
[CAD] SECTION .....	32
CAD Model Webform Settings .....	32
[ASM] SECTION .....	32
Assembly Form Settings .....	32
[PCB] SECTION .....	33
Bareboard Reference Designator Setting .....	33
Bareboard Form Settings .....	33
[PART] SECTION .....	33
Part Filter Settings .....	33
Part Form settings .....	34
[SEARCH] SECTION .....	34
<b>COMPONENT DATA MANAGEMENT .....</b>	<b>35</b>
3DEXPERIENCE WEB SERVICES .....	35
Publishing REST Web Service .....	35
CONNECTION SETTINGS FOR 3DEXPERIENCE WEB SERVICES .....	36
Connection Settings .....	36
DXDATABOOK CONFIGURATION .....	37
JDBC CONFIGURATION .....	38
DATA MAPPING LAYER .....	39
CDM Setup Wizard Settings Files .....	39
General Structure of the XML Configuration .....	40
Regular Expressions .....	40
PROCESSING INSTRUCTIONS .....	41
pemappings .....	41
import .....	41
namespace .....	42
define .....	42
userref .....	43
alternatives .....	43
break .....	44
ignore .....	44
commit .....	44
Test Instructions .....	45
test .....	45
Alias Instructions .....	46
Operations .....	52
duplicate .....	52
add .....	52
modify .....	52
set .....	53
setSpecialAttribute .....	53
setPartNumber .....	54
setEnoviaLink .....	54

remove .....	54
removeSpecialAttribute .....	55
concatAttribute .....	55
log .....	56
PART AND IP CLASSIFICATION MAPPING CONFIGURATION.....	57
<i>Upload of Parts</i> .....	57
<i>Download of Parts</i> .....	57
<i>Multiclassification</i> .....	58
Upload of Parts .....	58
Download of Parts.....	59
BACKGROUND SYNCHRONIZATION .....	60

---

## List of Figures

FIGURE 1 COMPONENT SEARCH DIALOG.....	20
FIGURE 2 TYPE SEARCH DIALOG.....	21
FIGURE 3 VAULT SEARCH DIALOG .....	22
FIGURE 4 POLICY SEARCH DIALOG .....	22
FIGURE 5 STATE SEARCH DIALOG .....	23
FIGURE 6 OPERATOR SEARCH: REAL.....	23
FIGURE 7 OPERATOR SEARCH: STRING.....	23
FIGURE 8 3DEXPERIENCE WEB SERVICES SETUP .....	36
FIGURE 9 LIBRARY PROJECT DIRECTORY.....	39
FIGURE 10 PART FAMILIES IN PART AND IP CLASSIFICATION .....	57

---

# Server Connector Installation

## Execute Setup

Login as a user with administrative privileges and follow the instructions. If you have no administrative privileges, the installation may fail.

### 3DEXPERIENCE Products

- X-CAD Design Foundation V6R2020x (mandatory)
- Engineering BOM Management Foundation V6R2020x (optional)
- IP Classification Foundation V6R2020x (optional)

## Installation Steps

1. Copy the contents of the Connector Server installer folder provided (ConnectorforMentorGraphicsExpeditionEnterpriseServer-V6R2020x.AllOS) to the Apps folder of the 3DEXPERIENCE server installation directory.

Example:

```
C:\DassaultSystemes\R2020x\3DSpace\Apps\  
<ConnectorforMentorGraphicsExpeditionEnterpriseServer>
```

2. Open the mql.exe from "C:\DassaultSystemes\R2020x\3DSpace\win\_b64\code\bin\mql.exe"
3. Set the path of the Connector server installer folder to execute the tcl file as shown below.

```
MQL<1>tcl;  
% cd  
C:/DassaultSystemes/R2020x/3DSpace/Apps/ConnectorforMentorGraphicsExpeditionE  
nterpriseServer;  
% exit
```

4. Run the ECADIntegInstallSchemaMain.tcl under C:\DassaultSystemes\R2020x\3DSpace\Apps\ConnectorforMentorGraphicsExpeditionEnterpriseServer to install the schema. Set the context user and vault before installing.

```
MQL<2>set context user creator password "password" vault "eService  
Administration";  
MQL<3>run ECADIntegInstallSchemaMain.tcl;
```

The installation is completed with all Schema installed.

## Silent Installation

Connector supports silent or unattended installation of server schema.

Follow the steps stated below.

1. Copy the contents of the Connector Server installer folder provided (ConnectorforMentorGraphicsExpeditionEnterpriseServer-V6R2020x.AllOS) to the Apps folder of the 3DEXPERIENCE server installation directory.

Example:

```
C:\DassaultSystemes\V6R2020x\3DSpace\Apps\  
<ConnectorforMentorGraphicsExpeditionEnterpriseServer>
```

2. Login to mql with an administrative user into a non-ADMINISTRATION vault.  
set context person creator password 'password' vault 'eService Administration';
3. Set the following environment variables directly in mql.  
set env silent\_mode YES;  
set env base\_directory = C:\DassaultSystemes\R2020x\3DSpace\Apps\  
ConnectorforMentorGraphicsExpeditionEnterpriseServer
4. Run the main installation script ECADIntegInstallSchemaMain.  
run ECADIntegInstallSchemaMain.tcl;
5. You can use this method also directly from a command line or a shell by passing over the commands to mql.

Example:

```
C:\DassaultSystemes\V6R2020x\3DSpace\win_b64\code\bin\mql.exe -c "set context  
person creator vault 'eService Administration';set env silent_mode YES;set env  
base_directory C:\DassaultSystemes\V6R2020x\3DSpace\Apps\  
ConnectorforMentorGraphicsExpeditionEnterpriseServer;run
```



```
C:\DassaultSystemes\V6R2020x\3DSpace\Apps\  
ConnectorforMentorGraphicsExpeditionEnterpriseServer\ECADIntegInstallSchemaMain.  
tcl"
```

The installation is completed with all Schema installed.

## Update Connector String Properties

After the installation of the server connector, user needs to update the connector specific string properties in 3DEXPERIENCE environment. User needs to manually merge the contents of the file "emxFrameworkStringResource.properties" under

```
"C:\DassaultSystemes\R2020x\3DSpace\Apps\ConnectorforMentorGraphicsExpeditionEnterp  
riseServer\emxFrameworkStringResource.properties" with the one under
```

```
"C:\DassaultSystemes\R2020x\3DSpace\STAGING\ematrix\properties\emxFrameworkStringRe  
source.properties"
```

Once the properties are added, the .war file has to be re-deployed by executing the following batch files,

```
C:\DassaultSystemes\R2020x\3DSpace\win_b64\code\command\BuildDeploy3DSpace_CAS.bat
```

---

# Client Connector Installation

This manual guides through the installation of Connector for Mentor Graphics Expedition Enterprise for Windows based systems.

## System Requirements

### Mentor Environment

- Mentor Graphics Expedition Enterprise VX2.3
- Mentor Graphics Expedition Enterprise VX2.4
- Mentor Graphics Expedition Enterprise VX2.5

### 3DEXPERIENCE Products

- XCADDesignConnector-V6R2020x

### Connector for Mentor Graphics Expedition Enterprise

- ConnectorforMentorGraphicsExpeditionEnterpriseDxDatabook32Bit
- ConnectorforMentorGraphicsExpeditionEnterpriseDxDatabook64Bit
- ConnectorforMentorGraphicsExpeditionEnterpriseVX32BitClient
- ConnectorforMentorGraphicsExpeditionEnterpriseVX64BitClient

### Java Environment

- Java JRE 11 64bit
- Java JRE 11 32bit

# Execute Setup

Login as a user with administrative privileges and follow the instructions. If you have no administrative privileges, the installation may fail.

## Windows Platforms

Please run the desired executable:

- ConnectorforMentorGraphicsExpeditionEnterpriseDxDatabookClient32bit.exe
- ConnectorforMentorGraphicsExpeditionEnterpriseDxDatabookClient64bit.exe
- ConnectorforMentorGraphicsExpeditionEnterpriseVX32BitClient.exe
- ConnectorforMentorGraphicsExpeditionEnterpriseVX64BitClient.exe

## Installation Steps – Connector for Mentor Graphics Expedition Enterprise

### 1. Introduction

It is the first step of the installation. It shows an overview of the product, its version and additional recommendations for the proper installation of Connector Client. Choose *Next* to proceed with the installation or *Cancel* will quit the installation.

### 2. Select Destination Directory

Choose or specify the path of the directory for installing the Connector. Select *Next* to proceed.

### 3. Select Mentor Installation

Please specify the directory where the Mentor Version is installed.

**Example:** C:\MentorGraphics\EEVX.2.3  
Proceed with the installation by selecting *Next*.

### 4. Select Installation Type

The user can choose between two different installation types. This alters the installation path of the connector commands. Selecting Local, alters the local Mentor installation. This is useful for single user access. In case of multiple user access, Admin type is used so that the commands can be placed at a shared location and each user can manipulate the local script to point to the shared location.

These settings define where the Connector specific menu entries should be placed.

Please select the type of your installation.

- **LOCAL**- Installation will add the commands to the local Mentor installation scope
- **ADMIN**- Installation will add the commands to the Connector installation folder.

The menu entries will be placed under the connector installation directory

**Example:** C:\ConnectorforMentorGraphicsExpeditionEnterpriseClient\bin\VX

A manual step needs to be performed to add the path of the dxStartup.vbs to the scripts.ini under "C:\MentorGraphics\EEVX.1.2\SDD\_HOME\standard\scripts.ini".

**Example:**  
[ViewDraw]  
Script#0 =

C:\ConnectorforMentorGraphicsExpeditionEnterpriseClient\bin\VX\dxStartup.vbs

**5. Choose Shortcut Folder**

Please specify where the product icons need to be created. After the installation, the *Start Menu* entries are created under the specified location. Proceed with the installation by selecting *Next*.

**6. Install Complete**

The final step of the installation which shows the final summary of the installation. A restart of the system is required to complete the installation. Select *Done* to complete the installation.

## SITE based Local Connector Configuration

A new environment variable 3DS\_ECAD\_SITE is introduced to support SITE based local connector configurations.

The variable points to a shared folder where the Scripts and Configs reside.

3DS\_ECAD\_SITE = <path of the SITE\_DIRECTORY>

The structure of the SITE folder should be

<path of the SITE\_DIRECTORY>/ENOVIA/MentorEE/script

<path of the SITE\_DIRECTORY>/ENOVIA/MentorEE/config

## Installation Steps - Connector for Mentor Graphics Expedition Enterprise DxDatabook

Login as a user with administrative privileges and follow the instructions. If you have no administrative privileges, the installation may fail.

**1. Introduction**

It is the first step of the installation. It shows an overview of the product, its version and additional recommendations for the proper installation of Connector Client. Choose *Next* to proceed with the installation or *Cancel* will quit the installation.

**2. Select Destination Directory**

Choose or specify the path of the directory for installing the Connector. Select *Next* to proceed.

**3. Choose Shortcut Folder**

Please specify where the product icons need to be created. After the installation, the *Start Menu* entries are created under the specified location. Proceed with the installation by selecting *Next*.

**4. Install Complete**

The final step of the installation which shows the final summary of the installation. A restart of the system is required to complete the installation. Select *Done* to complete the installation.

---

# Global Configuration Object Client settings

## Group ID

This feature provides the functionality to group different kinds of object types, so that they could be checked in always at the same time. This is especially necessary for the Project/Variant combination.

All cadTypes which have the same GroupId will be grouped together. All objects whose CadTypes are not referenced in this attribute will carry an individual GroupId and therefore do not belong to a group.

During Login process, the attribute ECADInteg-GroupId is received from 3DEXPERIENCE and is honored in the checkin dialog. An object selection in checkin dialog, evaluates the GroupId received and all Objects which belong to the same GroupId are either selected or deselected in top down or bottom up direction.

**Important note: The setting for Select Children is honoured with the standard functionality.**

Considering the above GCO setting and if “Select Children” is set to false, selecting a Variant object results in the selection of the Project as they belong to the same GroupId. This is a bottom up functionality.

In the same case, if the Select Children is set to true, selecting a Variant object results in the selection of the Project. This results in the selection of all the objects under the Project since the select children is set to true. This is top down functionality.

To set the Group ID, you update the ECADInteg\_GroupId attribute for the ECADInteg-GlobalConfig object. The syntax for the value of this attribute is <CADTYPE>|<GroupID> and the GroupId must be an integer value.

### Steps to configure Group ID:

1. Open an MQL window.
2. Execute these commands:

```
set context user creator;
modify businessobject "ECADInteg-GlobalConfig" MentorEEGlobal TEAM
"ECADInteg_GroupId" <CADTYPE1>|<GROUPID0> <CADTYPE2>|<GROUPID0>
<CADTYPE3>|<GROUPID0>;
```

Where <CADTYPE1>, <CADTYPE1>, and <CADTYPE1> and the object names, and <GROUPID0> is the ID they will share. For example:

```
modify businessobject "ECADInteg-GlobalConfig" MentorEEGlobal TEAM
"ECADInteg_GroupId" CadenceAllegroCISDesignType|0
CadenceAllegroCISVariantType|0;
```

## Derived Output Parameter File

The Connector supports the download and the execution of so called Derived Output Parameter Files prior to Derived Output Creation. Derived Output Parameter Files allow the Administrator to provide customized scripts to the end user. Parameter object could be used to provide additional information to create Derived Outputs while checkin operation. Once the user has selected a Parameter Object for a specific Derived Output, the Connector downloads the java script and executes it.

There are specific conventions which need to be obeyed.

The content of the parameter object can consist of the following files:

- Mandatory control script
- Additional and optional support scripts
  - a. The mandatory script needs to have the following Naming convention  
**<sampleParameterName>-<mappedDerivedOutputName>.js**  
**Example: myParameter-mill.js**
  - b. There can be multiple other files in this object which do not need to follow this naming convention. These other files can then i.e. be used inside the java script file
  - c. The java script needs to be checked into the corresponding Derived Output Format of the Parameter Object

In order to setup the parameter file for its corresponding derived output, the following setting in the GCO should be set by the user.

The attribute **IEF-DerivedOutputParameterObjTypeMapping** defines which type of Derived Output Parameter types are available for which specific mapped type of the Derived Output

<b>Attribute Name</b>	IEF-DerivedOutputParameterObjTypeMapping
<b>Attribute Values</b>	<DerivedOutput MappedType1> <ParameterObjecttype1> <DerivedOutput MappedType2> <ParameterObjecttype2>.... e.g. mill Derived Output Parameter

The attribute **IEF- DerivedOutputTypeDefaultParameterObjectMapping** defines the default Derived Output Parameter Object for the mapped Derived Output type. This Object is shown as default in Derived Output page while Checkin or This Value is directly used in case of checkin no option.

<b>Attribute Name</b>	IEF- DerivedOutputTypeDefaultParameterObjectMapping
<b>Attribute Values</b>	<DerivedOutput MappedType1>,<Default Parameter Object1> <DerivedOutput MappedType2>,<Default Parameter Object1> e.g. mill Derived Output Parameter,DOP-1,A

**Important note:** *The format of the Derived Output Parameter (DOP) file checked in to DOP Object should be of the type "Generic".*

**Example steps to configure Derived Output Parameter File:**

1. Open an MQL window.

2. Execute these command:

```
add bus "Derived Output Parameter" <DOP-NAME> <REVISION> policy "Derived  
Output Parameter Policy" vault "eService Production";
```

```
modify businessobject "ECADInteg-GlobalConfig" MentorEEGlobal TEAM "IEF-  
DerivedOutputTypeDefaultParameterObjectMapping" "<DO-NAME>|Derived Output  
Parameter,<DOP-NAME>,<REVISION>";
```

```
modify businessobject "ECADInteg-GlobalConfig" MentorEEGlobal TEAM "IEF-  
DerivedOutputParameterObjTypeMapping" "<DO-NAME>|Derived Output Parameter";
```

```
tcl;  
cd <path to DOP file>  
exit;
```

```
checkin businessobject "Derived Output Parameter" <DOP-NAME> <REVISION> format  
generic append "<DOP-Name>-<DO-Name>.js";
```

# Checkin Revision Behavior

The Revision behavior in the checkin is now configurable through GCO setting. Three different ways are supported for controlling the revisioning. Based on the selected setting, the user can create any revision, select NEXT revision or can be disallowed to create a new revision in the Connector.

## GCO Attribute

Specify which of the behaviors you want to configure.

- Attribute
  - Name: ECADInteg-CheckRevisionBehaviour
  - Type: String
  - Ranges:
    - “Allow Revisioning”
    - “Disallow Revisioning”
    - “Manual Revisioning”
  - Default: “Allow Revisioning”
- References:
  - Referenced in Type: ECADInteg-GlobalConfig
- ECADInteg-CheckinRevisionBehavior = [AllowRevisioning|DisallowRevisioning|ManualRevisioning]
  - Defines the Checkin revision behavior in CSE

## Allow Revisioning:

This allows the user to create only NEXT revision. The menu for creating NEXT revision will be available from the context menu of the design object in the checkin dialog. The user will not be able to create a manual revision. All the available revisions will be listed. The Revision list box is non-editable.

## Disallow revisioning:

This denies revisioning for the user. The user will not be able to create any manual revision nor a NEXT revision but all the available revisions will be listed. The Revision list box is non-editable.

**Important note: If the design object does not exist in 3DEXPERIENCE and user does not have the access for Revisioning, then the user will not be able to create the design object in 3DEXPERIENCE.**

## Manual Revisioning:

This allows the user to create NEXT revision as well specify an own revision. All the available revisions will be listed. The Revision list box is editable and the user can specify any revision. This user specified input will be used as a new revision.

To set the Group ID, you update the `ECADInteg_CheckinRevisionBehavior` attribute for the `ECADInteg-GlobalConfig` object.

## Steps to configure Revisioning:

1. Open an MQL window.
2. Execute these commands:

```
set context user creator;
modify businessobject "ECADInteg-GlobalConfig" MentorEEGlobal TEAM
"ECADInteg_CheckinRevisionBehavior" "Allow Revisioning"|"Disallow
Revisioning"|"Manual Revisioning";
```



# Role based commands

The administrator can control the access to Connector commands for any specific **role/roles** or even **disable** them completely using this GCO setting.

The GCO attribute which controls this is “ECADInteg-CSEDisabledCommands”.

During the Login to 3DEXPERIENCE from connector, the value specified in this GCO attribute is received from 3DEXPERIENCE. The execution of the Connector commands will be based on the configured setting.

Following is an overview of the available commands:

Command	Meaning
createworkspace	Workspace creation
deleteworkspace	Delete workspace
renameworkspace	Rename Workspace
createproject	Project creation
renameproject	Rename Project
deleteproject	Delete Project
createfolder	Workspace/Project folder creation
renamefolder	Rename Workspace/ Project Folder
deletefolder	Delete Workspace / Project Folder
movefolder	Move Workspace Project Folder
deletecadobject	delete CADObject
lifecycle	Promote/demote CADObject
assignfolder	Manually assign Folder to CAD Object
removefromfolder	Remove CADObject from Folder

All commands which are disabled will be greyed out and disabled in the Connector. A tooltip text shows that the command is disabled for the current role.

### Examples:

- createworkspace|VPLMCreator,VPLMViewer
- movefolder|VPLMCreator
- createfolder|disabled
- assignfolder|enabled

To set role-based access to commands, you update the ECADInteg-CSEDisabledCommands attribute for the ECADInteg-GlobalConfig object.

### Steps to configure role based commands:

1. Open an MQL window.
2. To set the access roles that cannot execute commands, execute these commands:

```
set context user creator;
modify businessobject "ECADInteg-GlobalConfig" MentorEEGlobal TEAM
"ECADInteg_CSEDisabledCommands" <COMMAND1>|<ACCESSROLE>
<COMMAND2>|<ACCESSROLE>;
```

Where:

- • <COMMAND> is one of the commands listed in Role-Based Access to Commands.
- • <ACCESSROLE> is a comma-separated list of the access roles that cannot access that command, enabled, or disabled.

For example:

```
modify "ECADInteg-GlobalConfig" MentorEEGlobal TEAM
"ECADInteg_CSEDisabledCommands" createworkspace|VPLMCreator,VPLMViewer
movefolder|VPLMCreator createfolder|disabled assignfolder|enabled;
```

## Enforce Folder on Save

The administrator can enforce if a CAD object needs to be assigned to a workspace folder during Save to 3DEXPERIENCE.

The GCO attribute which controls this is “ECADInteg-EnforceFolderOnSave”. The default value is set to **FALSE**.

Allowed Values: TRUE|FALSE

If the GCO is configured to **TRUE**, the user needs to choose the workspace folder before performing a Save using the browse button under *Save To* field.

To configure if users must save CAD objects to folders, you update the ECADInteg-EnforceFolderOnSave attribute for the ECADInteg-GlobalConfig object.

### Steps to configure Enforce Folder on Save:

1. Open an MQL window.
2. To enable or disable the enforcement, execute these commands:

```
set context user creator;
modify businessobject "ECADInteg-GlobalConfig" MentorEEGlobal TEAM
"ECADInteg-EnforceFolderOnSave" TRUE | FALSE;
```

Enter **TRUE** to force users to select folders when saving CAD objects, or **FALSE** to allow users to save CAD objects without selecting a folder.

## GCO Attributes used for User Settings

The user settings are read during the login to 3DEXPERIENCE. CSE reads the DEC login response and updates the setting. The existing CSE settings are overwritten by the 3DEXPERIENCE settings.

The user can modify these settings at runtime but it lacked saving it back to 3DEXPERIENCE. Currently this feature enables the user to save back the user modified settings to 3DEXPERIENCE.

The settings would be sent to 3DEXPERIENCE only when the client preferences are committed.

**Important note:** *The Settings will not be saved to 3DEXPERIENCE when the user commits the modifications done in Options dialog under Checkin dialog and also from the context menu for Derived output selection in Checkin dialog.*

The following user settings from the 3DEXPERIENCE Options could be saved back to 3DEXPERIENCE.

Attribute Name	Corresponding Setting Name in Connector
IEF-Pref-MCADInteg-CreateIterationOnCheckin	Create Iteration On Checkin
IEF-Pref-MCADInteg-LockObjectOnCheckin	Retain Lock After Checkin
IEF-Pref-IEF-DefaultConfigTables	Default View
IEF-Pref-MCADInteg-SelectChildItems	Select Children
IEF-Pref-MCADInteg-LockObjectOnCheckout	Lock Objects on Checkout
IEF-Pref-MCADInteg-CheckOutDirectory	Checkout Directory
IEF-Pref-IEF-DefaultLateralView	Default Table View
IEF-Pref-IEF-SelectedManualDerivedOutputs	Manual
IEF-Pref-MCADInteg-SelectedDerivedOutputs	Automatic

# BOM Data Management

The BOM Management provides a client driven User Interface. The User Interface as well as the business logic can be partially controlled through a definition in 3DEXPERIENCE. This functionality is provided through the means of 3DEXPERIENCE Webforms and 3DEXPERIENCE tables.

## UI Elements

### Component Search Dialog

By default, the Connector uses the `ECAD-ComponentSearchForm` to display the search criteria. The Part Search Dialog is configurable regarding the criteria shown below in the left frame of the figure.

Name	Type	Revision	State	Policy
?	Part	1	Create	Developer
4931144	Part	1	Create	Developer
0698-3572	Part	1	Create	Developer
1818-4699	Part	1	Create	Developer
1826-2831	Part	1	Create	Developer
26184	Part	1	Create	Developer
41103	Part	1	Create	Developer
101206-056	Part	1	Create	Developer
102100-038	Part	1	Create	Developer
117223-001	Part	1	Create	Developer
137486-011	Part	1	Create	Developer
24100070	Part	1	Create	Developer
A6968A	Part	1	Create	Developer
A9217A	Part	1	Create	Developer
A9222A	Part	1	Create	Developer
A9321A	Part	1	Create	Developer
A9341A	Part	1	Create	Developer
A4446593001	Part	1	Create	Developer
A24161310...	Part	1	Create	Developer
A24610015...	Part	1	Create	Developer
A28075581...	Part	1	Create	Developer
America	Part	1	Create	Developer
Asia	Part	1	Create	Developer
B0637A	Part	1	Create	Developer
B5586A	Part	1	Create	Developer
B7374A	Part	1	Create	Developer
CM2041	Part	1	Create	Developer

**Figure 1 Component Search Dialog**

The `ECAD-ComponentSearchForm` form includes these fields where users can enter search criteria:

- `emxFramework.Basic.Type`
- `emxFramework.Basic.Name`
- `emxFramework.Basic.Revision`
- `emxFramework.GlobalSearch.LatestRevisionOnly`
- `emxFramework.Basic.Policy`
- `emxFramework.Basic.Current`
- `emxFramework.Basic.Vault`
- `emxFramework.Basic.Description`

When designing the criteria form, you can remove any of the above fields, and add any additional fields. For complete details about modifying forms, see *MQL Command Reference: form Command*. Use that information

with the information here to determine how to update the `ECAD-ComponentSearchForm` form to meet your business requirements.

Below is an example to configure *Target Cost*.

```
modify form ECAD-ComponentSearchForm
field label targetcost
expression label "emxFramework.Attribute.Target_Cost"
setting "Query Attribute" "Target Cost"
setting "Query Comparitor" smatchlist
setting "Registered Suite" Framework
setting " client typeClass" operatorfield
setting format real
```

The `_client_typeClass` `operatorfield` and `format real` settings are required for configuring a field that uses operators.

Use string resources for labels to support localization.

Most fields allow text entry or selection from lists. You can use the `typeClass` setting for a field to define chooser or other types of selectors.

**Important note:** *The default Part Search webform 'ECAD-ComponentSearchForm' can be used as a reference to create your own Search Webforms.*

## Type Search Dialog

The Type Search Dialog will present the user with an expandable Tree in which the user can make the selection of exactly one entry.

The Tree displays the type hierarchy based on an input type. By default, the first Level of the tree components is expanded.

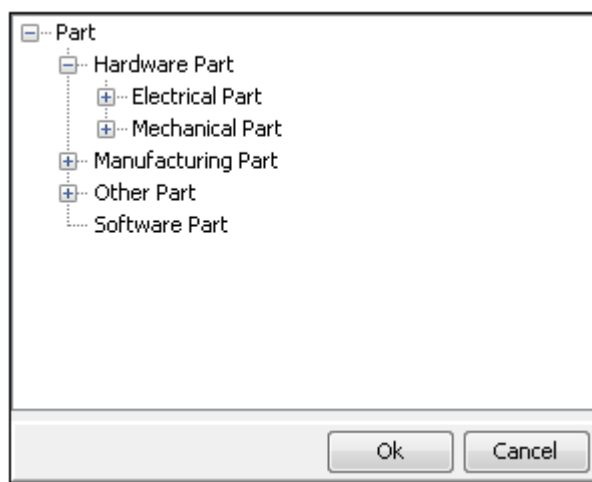
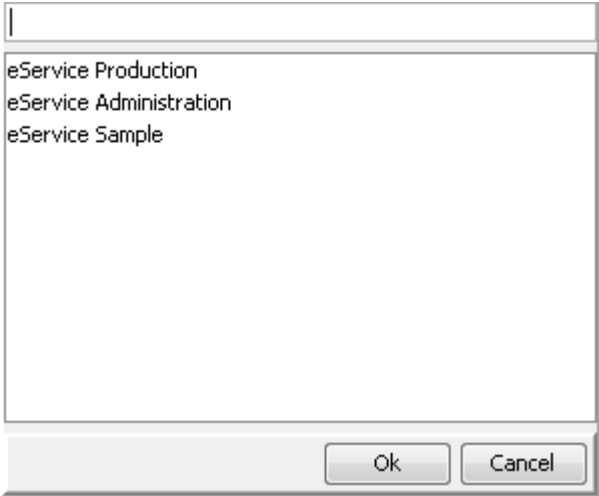


Figure 2 Type Search Dialog

## Vault Search Dialog

The Vault Search Dialog will present the user with a list of vault entries in which the user can de-/select one or more entries.

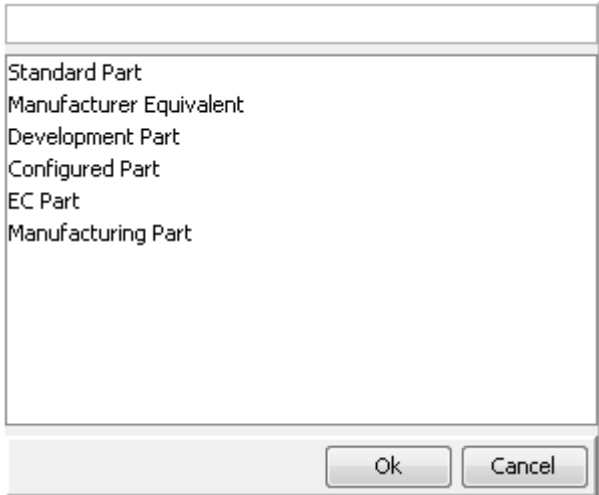
The upper entry field is used to do a fast selection based on some input. This means, if the user is typing some characters, the possible selection will be highlighted.



**Figure 3 Vault Search Dialog**

### Policy Search Dialog

The Policy Search Dialog will present the user with a list of policy entries in which the user can de-/select one or more entries. The upper entry field is used to do a fast selection based on some input. This means, if the user is typing some characters, the possible selection will be highlighted.



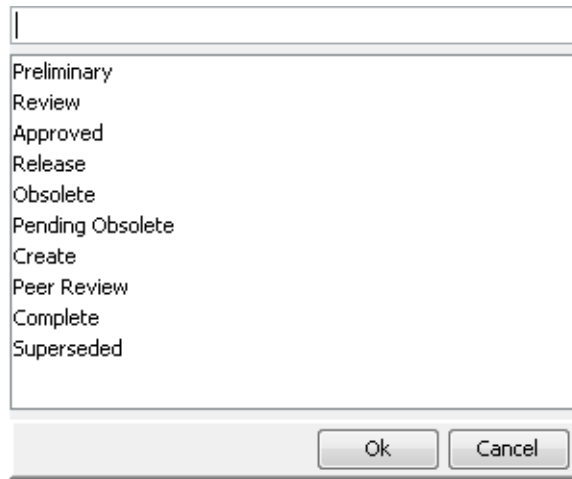
**Figure 4 Policy Search Dialog**

### State Search Dialog

The State Search Dialog will present the user with a list of state entries in which the user can de-/select one or more entries.

The list of available states is defined by the currently selected policies. Only states which are available for the selected policy will be shown.

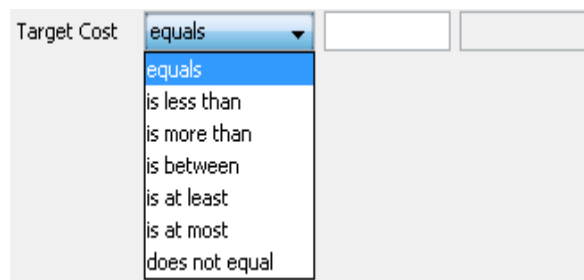
The upper entry field is used to do a fast selection based on some input. This means, if the user is typing some characters, the possible selection will be highlighted.



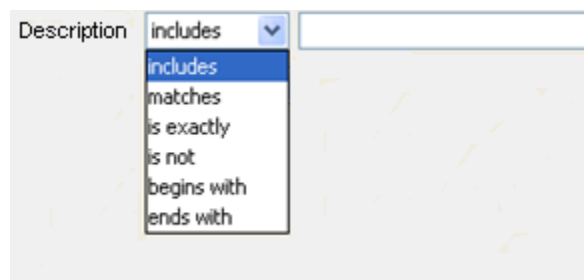
**Figure 5 State Search Dialog**

## Operator search

The user has the functionality to search based on additional operators. The operators are shown in a drop down menu. Based on the type of the attribute different operators are available. These can be configured through 3DEXPERIENCE Webforms. The figures below shows the configured *Target Cost* and *Description* as an example.



**Figure 6 Operator search: Real**



**Figure 7 Operator search: String**

## Webform Element Definition

The webform element definition defining the must have attributes. The following Webform elements exist and have a specific implementation:

### typechooser

The typechooser element will bring up a hierarchical Type chooser. As additional properties the root type has to be defined, as well as if the root type has to be made visible in the hierarchical tree.

**Properties:**

typeClass=typechooser  
rootType=[AdminType] e.g. “Part”  
rootTypeVisible=[true|false]

## policychooser

The policychooser element will bring up a policy chooser list. Can only be used in combination with a typechooser and is automatically linked to the selected types in the type chooser.

**Properties:**

typeClass=policychooser

## statechooser

The statechooser element will bring up a policy chooser list. Can only be used in combination with a policychooser and is automatically linked to the selected policies in the policy chooser.

**Properties:**

typeClass=statechooser

## vaultchooser

The vaultchooser element will bring up a vault chooser list.

**Properties:**

typeClass=vaultchooser

## revision

The revision element is a simple textfield which is mainly used in combination with the “latestrevision” element. It has to be linked to the revision property of the BusinessObject.

**Properties:**

typeClass=revision

## latestrevision

The latestrevision element checkbox is directly linked to the revision element. If activated, it makes the revision element non-editable. The meaning is to always search for the latest revision of the Business Object.

**Properties:**

typeClass=revision

## textfield

The textfield element will bring up a standard editable textfield.

**Properties:**

typeClass=textfield

## checkbox

The checkbox element will bring up a standard editable checkbox. An additional property defines if the checkbox is preselected or not.

**Properties:**

typeClass=checkbox  
typeClassSelected=[true|false]



## Table Columns

The Connector uses the *ECADBOMTableDetails* table to define the columns that display in the bill of materials. You can use this table as is, modify the columns that display, or create alternative tables that users can select.

**Important note:** Please note that you should not use the direct attribute name as the name of a column. E.g. use ‘\_name’ instead of ‘name’, ‘\_description’ instead of ‘description’

## UI Element cache refresh

After doing changes to webforms or tables used by the BOM Management, it is required to restart the server.

## Bill of Material Creation

For creating the Bill of Material Structure and for missing parts in 3DEXPERIENCE, webforms are used. There are webforms used for creating missing parts as well as webforms used to create relationships between parts or between parts and their specification document.

For connections between parts, always the usage 'ebomrelationship' is used. For connections between parts and their specification document always the usage 'ebompartspecrelationship' is used.

Please refer to the already used webforms for more technical details on the various settings.

## Material Existence Check

The material existence check feature is a function which checks if the material exists in PDM.

Based on specific rules a query is made to the database.

The command triggers a document search based on specific criteria defined by a webform and returns the component information based on a given table.

As an input to the criteria the command will provide a list of objects (hence here mainly the part number is provided).

In general the existence check mode is using configurable webforms which can build PDM queries in a generic way (e.g. pattern matching, stacked searches, etc.).

With this implementation also more complex scenarios like get the latest released object, get objects based on configured states can be handled.

The Connector will send the to-be-used webform which defines specific search criteria's for exactly one object.

---

## Bill of Material Client Configuration

This feature provides ability to configure the BOM settings in a local file of the Connector installation directory. The user can only see the configured BOM settings in the preferences and cannot modify them. The file is named bom-settings.ini and is stored under “<ROOT>\config” directory. In the bom-setings.ini the administrator can configure the complete BOM settings.

The bom-settings.ini file is structured as Groups and Attributes as shown in below example:

```
[GROUP_NAME]  
ATTRIBUTE_NAME=ATTRIBUTE_VALUE
```

Following chapters describes the configuration of the bom-settings.ini.

## [General] Section

### Design BOM creation

Display Name	Attribute Name	Default Value	Allowed Values
Design BOM Creation	DesignBOMStrategy	NEVER	<ul style="list-style-type: none"> <li>• NON-VARIANT-DESIGN</li> <li>• ALWAYS</li> <li>• NEVER</li> </ul>

#### Allow Design BOM

This setting enables the Design in the BOM Editor dialog based on the selected strategy.

- **Never**  
Design BOM will never be created and is disabled for selection under BOM dialog.
- **Always**  
Design BOM can be always created.
- **Non-Variant-Designs**  
Design BOM can be created if the design does not include variants.

### BOM Settings

Display Name	Attribute Name	Default Value	Allowed Values
Create collapsed BOM	CreateCollapsedBOM	true	<ul style="list-style-type: none"> <li>• true</li> <li>• false</li> </ul>
Allow Disabling of EDA Parts	AllowDisableEDAPart	true	<ul style="list-style-type: none"> <li>• true</li> <li>• false</li> </ul>
Allow Adding Parts from 3DEXPERIENCE or external Part lists	AllowAddENOVIAPart	true	<ul style="list-style-type: none"> <li>• true</li> <li>• false</li> </ul>
Store Parts Added by User	StoreUserPart	true	<ul style="list-style-type: none"> <li>• true</li> <li>• false</li> </ul>
Allow Manual assignment of EDA Parts with parts from 3DEXPERIENCE in the BOM structure	AllowManualAssignment	false	<ul style="list-style-type: none"> <li>• true</li> <li>• false</li> </ul>
Use 3DEXPERIENCE Background Job for BOM creation	UseBackgroundJob	false	<ul style="list-style-type: none"> <li>• true</li> <li>• false</li> </ul>
Threshold	BackgroundJobThreshold	1000	Integer numbers

#### Create Collapsed BOM

- If activated then a collapsed BOM is transferred to 3DEXPERIENCE, otherwise an expanded BOM is transferred. In either case the BOM Editor will display an expanded BOM.

#### Allow Disabling of EDA Parts

This setting defines if the user is allowed to disable components coming in automatically from the EDA side through the BOM extraction process. If this flag is set, all the EDA Components are made optional. If this flag is unset, all EDA components are made mandatory. The default setting is true. This enables the “assign/unassign” context menu for EDA Components in the BOM editor dialog.

#### Allow Manual assignment of EDA Parts with parts from 3DEXPERIENCE in the BOM structure

Enabling this setting allows the user to add Components from 3DEXPERIENCE or from External part list to the BOM structure. The Components in the 3DEXPERIENCE can be found using a part search dialog. Added Components are flagged as USER components and only these Components can be removed. It will not be allowed to remove EDA components.

**Store Parts Added by User**

This setting allows the user to store the parts added by the user. When the user clicks on the “Store” button in the BOM management dialog, the user created parts are stored along with the other EDA parts in the local system.

**Allow Manual assignment of EDA Parts with parts from 3DEXPERIENCE in the BOM structure**

This setting enables the user to change (re-assign) the default part of an EDA component with a new 3DEXPERIENCE Part.

**Use 3DEXPERIENCE Background Job for BOM creation**

Enabling this setting runs the BOM creation as a background job in 3DEXPERIENCE so the user can proceed with the other operations during this process.

- **Threshold**

The user can specify the threshold value which states that the background job will be used only when BOM contains parts more than the specified value. For example, if the specified threshold is 500, then the background job is carried out if the BOM contains more than 500 parts.

## Missing Part Strategy

This setting defines if the BOM creation can be executed or not if the BOM contains EDA parts which do not exist in 3DEXPERIENCE. If the BOM extracted from the EDA side contains parts which have not been created in 3DEXPERIENCE / are missing in 3DEXPERIENCE, various strategies can be applied.

Display Name	Attribute Name	Default Value	Allowed Values
Error   Warning   Ignore	MissingPartStrategy	ERROR	<ul style="list-style-type: none"> <li>• ERROR</li> <li>• WARNING</li> <li>• IGNORE</li> </ul>
Create non existing parts in 3DEXPERIENCE	AllowPartCreation	true	<ul style="list-style-type: none"> <li>• true</li> <li>• false</li> </ul>

- **Error**  
Enabling this setting disables the “Create non existing parts in 3DEXPERIENCE” checkbox. When the user checks this radio button, an error message comes up when the BOM contains EDA parts which do not exist in 3DEXPERIENCE. The user will not be able to proceed with the BOM creation process until he has solved the issues.
- **Warning**  
Enabling this setting enables the “Create non existing parts in 3DEXPERIENCE” checkbox. When the user checks this radio button, a warning message is shown to the user that the BOM contains EDA parts which do not exist in 3DEXPERIENCE. The user is able to proceed with the BOM creation. Depending on the setting of the “Create non existing parts in 3DEXPERIENCE” checkbox the missing parts are either skipped or created
- **Ignore**  
Enabling this setting enables the “Create non existing parts in 3DEXPERIENCE” checkbox. When the user checks this radio button, the BOM is created in 3DEXPERIENCE and no message is shown to the user regarding the missing parts.

Depending on the setting of the “Create non existing parts in 3DEXPERIENCE” checkbox the missing parts are either skipped or created.

### Create non existing parts in 3DEXPERIENCE

This setting creates all the EDA parts which are missing in 3DEXPERIENCE during BOM creation. If this flag is set to false, then the missing parts are removed from the BOM creation process and are not considered for the 3DEXPERIENCE BOM. The created part is defined by the scope specific Part Creation Webform – here e.g. Type, Policy etc. are defined.

Please note that this feature should only be used for testing and prototyping but not in a production environment. It can result in an error message presented by the user if the part is not found in 3DEXPERIENCE because of e.g. not having the correct state or not visible to the user, and the Connector is trying to create the part.

Example:

- The part ‘1234’ exists in state ‘Preliminary’ in 3DEXPERIENCE
- The Connector is looking for a part ‘1234’ in state ‘Release’ and therefore flags the part as non-existent in the BOM Editor.
- If the user now wants to create the non-existing part in 3DEXPERIENCE, an error message will be returned to the user that the part cannot be created. This behavior in fact is correct, as the part already existed, but is not ‘visible’ to the user because of search criteria or access permissions.

## Checkin Behavior after Save to 3DEXPERIENCE

Display Name	Attribute Name	Default Value	Allowed Values
Execute BOM creation after Save	ExecuteBOMCreationAfterSave	false	<ul style="list-style-type: none"><li>• true</li><li>• false</li></ul>
Execute BOM creation after Quick Save	ExecuteBOMCreationAfterQuickSave	true	<ul style="list-style-type: none"><li>• true</li><li>• false</li></ul>

### Execute BOM creation after Save

If this setting is activated, the User has the possibility to directly execute a BOM creation after save. Performing a Save of the CAD object to 3DEXPERIENCE checks in the CAD object and later brings up the BOM editor. The BOM of the selected CAD object is automatically extracted. The BOM selection checkbox is disabled for user selection.

### Execute BOM creation after Quick Save

If this setting is activated, the BOM creation is automatically executed after checkin. The BOM is created without any user interaction.

## [UI] Section

### BOM Table Settings

Display Name	Attribute Name	Default Value
Table BOM Structure View	BOMTableEDAView	ECAD-MEE-BasicPropertyDetails
Table EDA Part Details	BOMTablePartView	ECAD-MEE-BOMTableDetails

#### Table BOM Structure View

This setting defines which table view is used to display the columns in the BOM view.

#### Table EDA Part Details

This setting defines which ECAD properties to be displayed. This setting is basically the view definition which will be used for displaying the parts in the part search dialog.

## [CAD] Section

### CAD Model Webform Settings

Display Name	Attribute Name	Default Value	Allowed Values
CAD Model Existence Check	CADModelExistenceCheck	ECAD-ExistenceCheckDocumentCriteria	
Connect BOM to CAD Model Revision	CADModelConnectToRevision	true	<ul style="list-style-type: none"><li>true</li><li>false</li></ul>

#### Webform - CAD Model Existence Check

It specifies which webform will be used to check the existence of the CAD models in 3DEXPERIENCE. When the user brings up the BOM management UI, the existence of the previously added document is checked and queried in 3DEXPERIENCE for its existence using the document criteria defined in the webform.

#### Connect BOM to CAD Model Revision

This attribute defines if the BOM is connected to the CAD Model Revision or the CAD Model Version. This is useful, if the user does not work with versioned objects or wants to see the BOM always on the CAD Model Revision.

## [ASM] Section

### Assembly Form Settings

Display Name	Attribute Name	Default Value	Allowed Value
Assembly Part Existence Check	AssemblyExistenceCheck	ECAD-MEE-PartExistenceCheck	-

#### Webform - Assembly Part Existence Check

Defines the webform used to check if the assembly component exists in 3DEXPERIENCE. The Connector queries 3DEXPERIENCE for the existence of the assembly part based on selected webform.



## [PCB] Section

### Bareboard Reference Designator Setting

Display Name	Attribute Name	Default Value	Allowed Values
Reference Designator Bareboard	PCBReferenceDesignator	PCB	-

#### Reference Designator Bareboard:

Defines the Reference Designator used for the bareboard

### Bareboard Form Settings

Display Name	Attribute Name	Default Value	Allowed Values
Bareboard Part Existence Check	PCBExistenceCheck	ECAD-MEE-PartExistenceCheck	-

#### Webform - Bareboard Part Existence Check:

Defines the webform used to check the existence of the bareboard component in 3DEXPERIENCE. The Connector queries 3DEXPERIENCE for the existence of the bareboard part based on selected webform.

## [Part] Section

### Part Filter Settings

Display Name	Attribute Name	Default Value	Allowed Values
Enable Ignore EDA Parts	PartIgnoreEDAPart	true	<ul style="list-style-type: none"><li>true</li><li>false</li></ul>
Regular Expression	PartIgnoreEDAPartRegExp	partNo=^TP.*\$;partNo=^TESTP.*\$;partNo=^IGNORE.*\$	Attributes: <ul style="list-style-type: none"><li>partNo</li><li>refDes</li><li>quantity</li></ul> Values: regular expressions

#### Enable Ignore Components/Regular Expression

The user has the possibility to ignore parts coming in from the EDA BOM extraction process by defining regular expressions. This might be very useful to ignore part entries like Testpoints, DNI Parts etc. which are required in the CAD Design itself but have no meaningful representation in the PLM Side.

This setting defines the semicolon separated regular expressions which are used to ignore parts coming from the EDA side.

#### Example:

```
partNo=^OA.*$;partNo=^TESTP.*$;partNo=^IGNORE.*$
```

This expression indicates that, all parts starting with OA, TESTP and IGNORE will be ignored for BOM creation.

## Part Form settings

Display Name	Attribute Name	Default Value	Allowed Values
Part Creation	PartCreationForm	ECAD-MEE-Part-AutoCreateForm	-
Part Existence check	PartExistenceCheck	ECAD-Part-FindLatestReleasedOrLatestRev	<ul style="list-style-type: none"> <li>• ECAD-Part-FindLatestReleasedOrLatestRev</li> <li>• ECAD-Part-FindLatestRev</li> <li>• ECAD-Part-FindLatestReleased</li> </ul>
-	PartExistenceCheckKnownRev	ECAD-MEE-PartExistenceCheck	
EBOM Relationship	PartSpecRelationship	ECAD-BOM-RelationshipForm	-

### Part Creation

Defines the webform used to create missing parts in the 3DEXPERIENCE.

### Part Existence check

Defines the webform used to check the existence of the parts in the 3DEXPERIENCE. The Connector queries 3DEXPERIENCE for the existence of the part based on selected webform which specifies the state of the part for example, when the user selects Latest Released Revision, then it checks for all the assembly parts with the latest revision in the released state.

### Part Existence check for Known Revisions

In case that the revision of the Part is known then existence check is carried out for the exact Type, Name and Revision.

### EBOM Relationship

Defines the form used to create a relationship between parts i.e.; create an EBOM connection between the component and its parent component

## [Search] Section

Display Name	Attribute Name	Default Value
Part Search Criteria	SearchCriteriaForm	ECAD-MEE-ComponentSearchForm
Part Search Details Table	SearchDetailsTable	ECAD-MEE-ENGPARTSearchDetails
Search Limit	SearchLimit	500

### Webform - Part Search Criteria

Specifies which webform will be used for displaying the part search criteria. When bringing up the Search Screen for searching Parts in 3DEXPERIENCE, the used webform defines what kind of search criteria is available to the user.

### Table Part Search Details

This defines the table column definition used in the component search details table. When bringing up the part Search Screen for searching Parts in 3DEXPERIENCE, the used webform defines what kind of table column details are available to the user.

### Search limit

The user can specify the default search limit which limits the number of objects retrieved during search.

---

# Component Data Management

This chapter describes how to configure 3DEXPERIENCE Web Services and how to use the data mapping layer of the Component Data Management. It also provides an overview of how the mapping layer works and how it can be configured. *Regular Expressions* chapter gives a brief overview of the regular expressions used with the data mapping layer.

## 3DEXPERIENCE Web Services

Before Component Data Management can be used it is necessary to configure Web Services in 3DEXPERIENCE. This is described in *Publishing REST Web Service* chapter. After publishing of the Web Services in 3DEXPERIENCE, the connection settings for 3DEXPERIENCE Web Services can be configured.

### Publishing REST Web Service

The REST Web Service used by the Component Data Management needs to be published.

#### Steps:

1. Stop *3DEXPERIENCE 3Dspace TomEE Service*
2. Copy the file *EcadComponentServices.jar* from the directory `<ConnectorforMentorGraphicsExpeditionEnterpriseServer>/ComponentServices` to the directory `<3Dspace>\win_b64\code\tomcat\current\webapps\3dspace\WEB-INF\lib`
3. Restart *3DEXPERIENCE 3Dspace TomEE Service*
4. To verify if the REST Web Service is published use the following URL (case sensitive):  
<https://<SERVERNAME>:<PORTNUMBER>/3dspace/resources/EcadComponentServices/JpoWrapper?action=info>

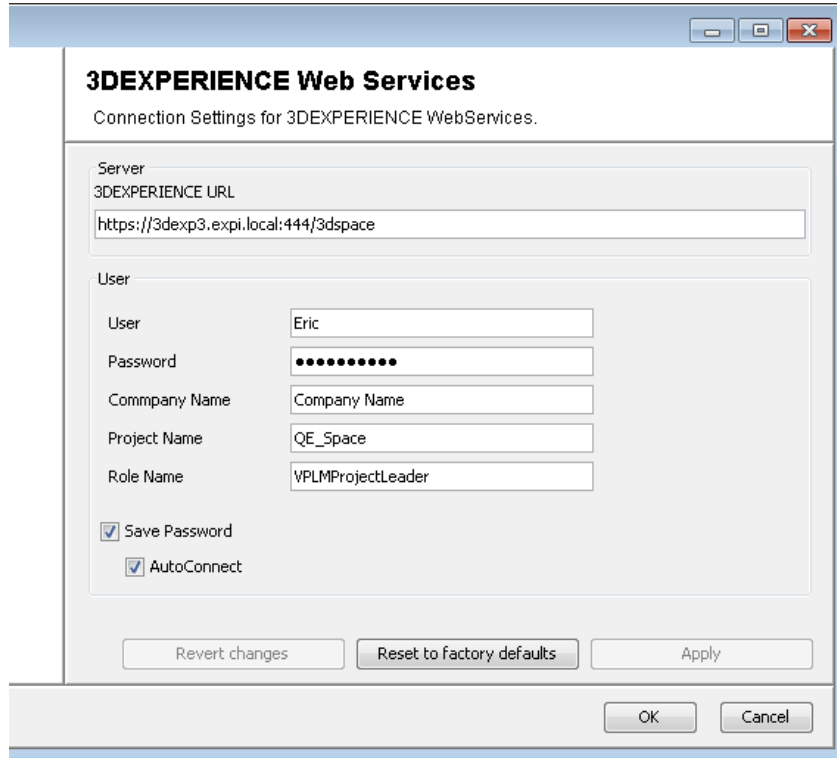
Example:

<https://3dexp4:444/3dspace/resources/EcadComponentServices/JpoWrapper?action=info>

If the Web Service is correctly published *JpoWrapper Service* is displayed in browser.

# Connection Settings for 3DEXPERIENCE Web Services

For the General Settings in the Preferences please refer to the User Manual for a detailed explanation.



**Figure 8 3DEXPERIENCE Web Services Setup**

## Connection Settings

- |                              |   |
|------------------------------|---|
| 3DEXPERIENCE Webservices URL | - The host name and the port number of the 3DEXPERIENCE Webservices Server  |
| User                         | - The username to connect to 3DEXPERIENCE   |
| Password                     | - The password associated with the specified username   |
| Company Name                 | - The name of the company that the user belong to   |
| Project Name                 | - This is the project (collaborative space) to which the user is assigned.  |
| Role Name                    | - The role assigned to the user on the specified project  |
| Save Password                | - This saves the specified password   |
| AutoConnect                  | - Selecting this checkbox enables automatic login process with the saved login credentials. In background synchronization, this checkbox needs to be enabled. |

**Important note:** Enabling this AutoConnect checkbox is mandatory when the Background Synchronization mode needs to be used. In this case, a valid user and password should be provided.

## DxDatabook Configuration

An example to configure DxDatabook configuration for "INNER JOIN", "LEFT JOIN", "RIGHT JOIN", "NONE" (single table), "UNION" is stated below. The DxDatabook configuration can be configured under <ConnectorRootFolder>/data/ as global.dxDdbConfig or <LibraryName>.dxDdbConfig. In the Connector preferences you can configure which dxDdbConfig file should be used (*ClientPreferences* → *Expedition Enterprise* → *DxDatabook* → *Configuration* → *Library Configuration*).

Every value is surrounded by double quotes ("). Backslash is an escape character that works like in Java. To use a double quote in a value, use the backslash as escape character (\").

Everything in this file that is not a comment or a value (between double quotes) is a key word. Spaces and tabulators are ignored. Newlines as well (except for the newlines at the end of a comment).

Configuration	Description
ModifyLibrary "<LIBRARYNAME>"	This is a library modification for the configured library.
JoinType "JOINTYPE"	The join type: "INNER JOIN", "LEFT JOIN", "RIGHT JOIN", "NONE" (single table), "UNION".
TableNames "< LIBRARYNAME>" "<TABLENAME1>"	Modify the names of the tables to use. The original is overwritten.
JoinOn "<TABLENAME2>" = "<TABLENAME3>"	Join the tables on these column names. In this case [LIBRARYNAME].[TABLENAME2] = [TABLENAME1].[TABLENAME3]

Example:

```
ModifyLibrary "als"
JoinType "INNER JOIN"
TableNames "als" "Symbols"
JoinOn "Part Number" = "Part Number"
```

## JDBC Configuration

The user needs to configure JDBC. The configuration must be manually created by the user. The configuration file should have the same name as the ".dbc" file with an extension of ".jdbc.ini" and under the same directory as the library.

**Example:** In case the database name is DxLib.dbc, create a file under the same directory as DxLib.jdbc.ini

Once the file is created, please use a text editor to open It and specify the following URL. The structure of the INI file has 2 global Keys namely [URL] and [DRIVER].

URL- This is the central part of the configuration. The connection URL is specified as shown

**Example: Syntax - UcanAccess for Microsoft Access database**

```
url=jdbc:ucanaccess://<Path to the .mdb>
url=jdbc:ucanaccess://C:\temp\DxLib\DxLib.mdb
```

Driver- This contains the driver class name to establish a connection to the data source

**Example: Syntax**

```
driver=net.ucanaccess.jdbc.UcanaccessDriver
```

The driver configuration is only necessary if there are two drivers in the system that feel responsible for the same URL schema. If no driver is specified (the key driver does not exist) then the first driver that feels responsible will take effect.

An additional Key named [Properties] contains driver specific settings.

**Example: Syntax**

```
url= jdbc:ucanaccess://C:\temp\DxLib\DxLib.mdb
[Properties]
user=Test
password=test
CharacterSet=UTF-8
```

Since DxDatabook has the possibility to configure for each EdaPartCategory a different data source, two additional sections namely [Name] and [Name properties], where Name is replaced with the DxDatabook Library. In this case, DxDatabook Library first loads the Global setting and later Library specific settings.

**Example: Syntax**

```
[Properties]
user=Test
password=test

[Capacitor]
url=jdbc:ucanaccess://C:\temp\DxLib\Capacitor.mdb

[Capacitor Properties]
CharacterSet=UTF-8

[Resistor]
url=jdbc:ucanaccess://C:\temp\DxLib\Resistor.mdb

[Resistor Properties]
CharacterSet=Windows-1252
```

## Data Mapping Layer

The Data Mapping Layer provides all necessary functionality to convert 3DEXPERIENCE parts information to ECAD parts information and vice versa.

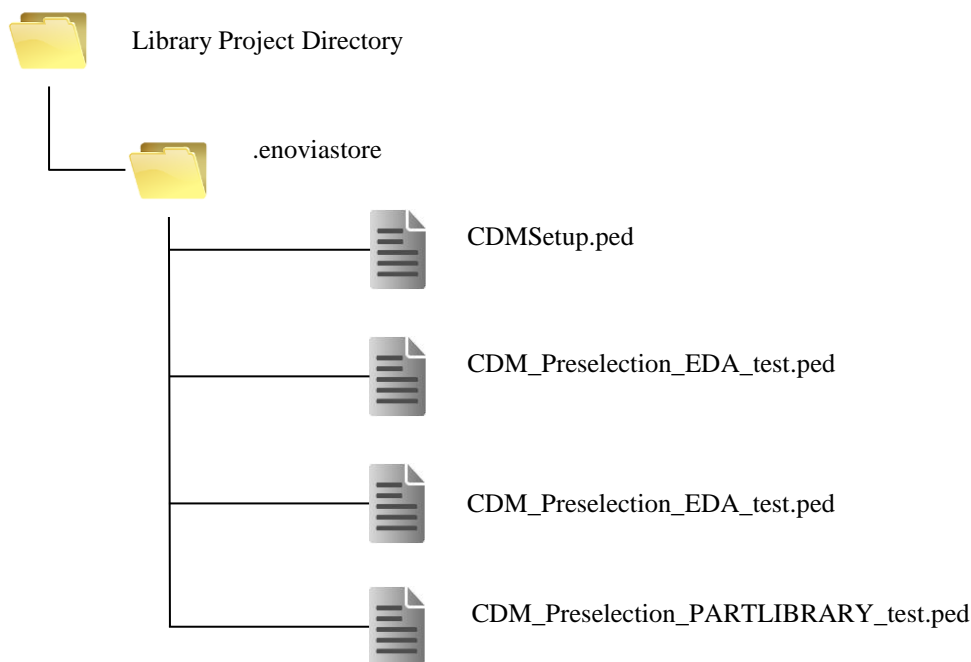
The Data Mapping Layer is configured by a mapping XML file which contains instructions (rules/tests and definitions/operations) for the part manipulation. The instructions will be applied on the part which is passed from the source into the mapping layer. After applying all instructions, the modified part is then passed over to the target for further processing. It is important to mention that the output part is by default the same than the input part. Only manipulations of part data will occur.

The tests and operations are described in the subsequent sections of this chapter.

The data mapping layer is configured with the use of configuration files. A configuration is structured in a XML file which consists of instructions and which can import further files with instructions. The format of the configuration is defined in a XSD file (XML-Schema).

## CDM Setup Wizard Settings Files

Once the user finished the CDM Setup Wizard task, the following files are created under the Library directory. There are dedicated files for all the pre-selections made and another which stores the path of the mapping file and other CDM setup. The settings made here are used when the user runs the CDM task. The Connector loads all the settings automatically from these files into the CDM synchronization dialog. The user has to run the CDM Setup Wizard task to run the CDM Task and the background synchronization process.



**Figure 9 Library Project Directory**

## General Structure of the XML Configuration

Following is small example of an XML mapping configuration.

```
<?xml version="1.0" encoding="UTF-8"?>
<pemapping version="1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation="../mapping-schema.xsd">
  <test>
    <operation></operation>
  </test>
  <operation></operation>
</pemapping>
```

**The XML schema “mapping-schema.xsd” for CDM Data Mapping Layer is found in ../data of the installation.**

## Regular Expressions

*Regular expressions* are a way to describe a set of strings based on common characteristics shared by each string in the set. They can be used to search, edit, or manipulate text and data. The Regular Expressions syntax supported is the JAVA Regular expression syntax.

The Connector will also support the concept of Capturing groups; in fact this is one of the essential elements of the regular expression support. Capturing groups are a way to treat multiple characters as a single unit. They are created by placing the characters to be grouped inside a set of parentheses. For example, the regular expression (dog) creates a single group containing the letters "d" "o" and "g". The portion of the input string that matches



# Processing Instructions

This instructions control the workflow of the Data Preparation Layer.

## pemappings

general Syntax	<pre>&lt;pemapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"            xsi:noNamespaceSchemaLocation="path to the XSD file "            version="mapping engine version number"&gt;             child instructions  &lt;/pemapping&gt;</pre>		
description	The root node of the configuration. This is the start node. It contains the version of the mapping file. The current version of the mapping files is 1.0.		
attributes	Name	Use	Annotation
	version	required	The version of the configuration
examples	<pre>&lt;!-- the root structure of the configuration the mapping-schema.xsd can be found       in &lt;3DEXPERIENCE_home&gt;/data/mapping-schema.xsd --&gt; &lt;pemapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"            xsi:noNamespaceSchemaLocation="mapping-schema.xsd"            version="1.0"&gt;   ...child instructions...  &lt;/pemapping&gt;  &lt;!-- the root structure of the configuration the mapping-schema.xsd can be found       on local system in file C:/mapping-schema.xsd --&gt; &lt;pemapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"            xsi:noNamespaceSchemaLocation="file:///C:/mapping-schema.xsd"            version="1.0"&gt;   ...child instructions...  &lt;/pemapping&gt;</pre>		

## import

general Syntax	<pre>&lt;pemapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"            xsi:noNamespaceSchemaLocation="mapping-schema.xsd "            version="1.0"&gt;             &lt;import name="unique name of the import to reference it as a namespace"                  path="the path of the imported file" /&gt;  &lt;/pemapping&gt;</pre>		
description	Defines an import of a configuration XML file. The imported and importing file must be of the same version.		
attributes	Name	Use	Annotation
	name	required	The name of the import used to address the regular expression references. The references in the imported file can then be addressed with the syntax: importName::referenceName e.g.

			Allowed character set: [p{Alnum}\_\-] = [a-z A-Z 0-9 _ -]
	path	required	The path of the file to import. The path can be absolute or relative to the path of the importing file.
examples	<pre> &lt;!-- imports the file import1.xml from the same directory as the current file, (relative path) --&gt; &lt;pemapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="mapping-schema.xsd " version="1.0"&gt;    &lt;import name="import1" path="import1.xml" /&gt;  &lt;/pemapping&gt;  &lt;!-- imports the file import1.xml from given path (absolute path) --&gt; &lt;pemapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="mapping-schema.xsd " version="1.0"&gt;    &lt;import name="import1" path="C:/configurations/import1.xml"/&gt;  &lt;/pemapping&gt; </pre>		

## namespace

general syntax	<pre> &lt;pemapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="mapping-schema.xsd " version="1.0"&gt;    &lt;namespace/&gt;  &lt;/pemapping&gt; </pre>		
description	Defines a namespace of the configuration.		
examples	<pre> &lt;pemapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="mapping-schema.xsd " version="1.0"&gt;    &lt;!-- default namespace --&gt;   &lt;namespace&gt;      child instructions    &lt;/namespace&gt;  &lt;/pemapping&gt; </pre>		

## define

general syntax	<pre> &lt;define id="a unique define id within the namespace"&gt;   ..child instructions.. &lt;/define&gt; </pre>		
description	Defines a block which can be reused. The define block contains a set of instructions which can be reused. The instruction <a href="#">userref</a> references a define block.		
attributes	Name	Use	Annotation
	id	required	The id of the define block to use to reference the block. Allowed character set: [p{Alnum}\_\-] = [a-z A-Z 0-9 _ -]

examples	<pre> &lt;pemapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"            xsi:noNamespaceSchemaLocation="mapping-schema.xsd "            version="1.0"&gt;   &lt;import name="import1" path="import1.xml" /&gt;    &lt;namespace&gt;     &lt;!-- define block can be referenced within its namespace --&gt;     &lt;define id="internal_define_block"&gt;       ...child instructions...     &lt;/define&gt;      &lt;!-- references a define block within this mapping --&gt;     &lt;userref refid="internal_define_block" /&gt;      &lt;!-- references a define block from an external mapping which is          imported in the current mapping(import1.xml) --&gt;     &lt;userref refid="import1::external_define_block" /&gt;    &lt;/namespace&gt; &lt;/pemapping&gt; </pre>
----------	--

## userref

general syntax	<code>&lt;userref refid="referenced id" /&gt;</code>		
description	References a define block.		
attributes	Name	Use	Annotation
	refid	required	The id of the define block to use. To reference blocks inside an imported configuration, use the following syntax:  importedName::defineBlockName
example	<pre> &lt;!-- references a reference inside the current mapping --&gt; &lt;userref refid="define1" /&gt;  &lt;!-- references a define block from an external mapping which is      imported in the current mapping(imported1) --&gt; &lt;userref refid="imported1::define1" /&gt;  &lt;!-- here the current configuration imports imported1 which imports      imported2. The reference references define1 inside imported2 --&gt; &lt;userref refid="imported1::imported2::define1" /&gt; </pre>		

## alternatives

general syntax	<pre> &lt;alternatives&gt;   child instructions &lt;/alternatives&gt; </pre>
description	<p>The alternatives statement is like an if-else operation. When the first child instruction in the alternatives statement returns true, than the next child instruction is not executed and the alternatives statement is exited. If the first child instruction returns false, than the next child instruction will be executed.</p> <p>Please note that only alternatives and test statements can return false. Other instructions or operations will always return true.</p>
examples	<pre> &lt;!-- the execution here is as follows:   1. if attribute SYMBOL exist, add attribute SUB SYMBOL with value 1 and      return true and exit alternatives   2. if attribute SYMBOL does not exist, it returns false and proceed to the      next child instruction where an attribute SYMBOL with value 1 is added      and exit alternatives --&gt; </pre>

	<pre> &lt;alternatives&gt;   &lt;test type="attribute" name="SYMBOL"&gt;     &lt;add target="attribute" name="SUB_SYMBOL value="1" /&gt;   &lt;/test&gt;   &lt;add target="attribute" name="SYMBOL" value="1" /&gt; &lt;/alternatives&gt; </pre>
--	--

## break

general syntax	<code>&lt;break /&gt;</code>
description	Breaks the execution. If used within <a href="#">duplicate</a> continues with the siblings of the parent duplicate. Else breaking the execution of the component and returning the component with the current modifications.
examples	<pre> &lt;pemapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"   xsi:noNamespaceSchemaLocation="mapping-schema.xsd "   version="1.0"&gt;    &lt;namespace&gt;     &lt;duplicate&gt;       &lt;userref refid="define1" /&gt;       &lt;userref refid="define2" /&gt;       &lt;test type="attribute" name="SYMBOL"&gt;         &lt;test type="attribute" name="SUB_SYMBOL"&gt;           &lt;!-- breaks execution of duplicate, writes the duplicate and             exits duplicate (add of SUB_SYMBOL will not be executed) --&gt;           &lt;break /&gt;         &lt;/test&gt;         &lt;add name="SUB_SYMBOL"&gt;       &lt;/test&gt;       &lt;add name="SYMBOL"&gt;     &lt;/duplicate&gt;     &lt;!-- breaks execution of the current component --&gt;     &lt;test type="attribute" name="SYMBOL"&gt;       &lt;break /&gt;     &lt;/test&gt;   &lt;/namespace&gt; &lt;/pemapping&gt; </pre>

## ignore

general syntax	<code>&lt;ignore /&gt;</code>
description	Ignore, does not add the current component and breaks the execution.
examples	<pre> &lt;!-- ignores the operand (breaks the complete execution and removes the current   operand from the result if the attribute SYMBOL exists --&gt; &lt;test type="attribute" name="SYMBOL"&gt;   &lt;ignore /&gt; &lt;/test&gt; </pre>

## commit

general syntax	<code>&lt;commit /&gt;</code>
description	Commits the copy of the component with the current state to the list of the duplicated components and continues execution of the descendant instructions.
examples	<pre> &lt;!-- commits a copy of the operand to the duplicate list and continues with the   operand if the attribute SYMBOL exists --&gt; &lt;test type="attribute" name="SYMBOL"&gt;   &lt;commit /&gt; &lt;/test&gt; </pre>

## Test Instructions

The Test instruction is a special statement which is used to execute tests on a specific component. If the test matches the condition, the return value is set to true, otherwise to false. If the test returns true, all child/nested instructions of it will be executed.

All following tests are aliases (shortcuts) for the test instruction.

Type	Full Instructions	Alias Instructions
<a href="#">attribute</a>	<code>&lt;test type="attribute" name="SYMBOL" /&gt;</code>	<code>&lt;attribute name="SYMBOL" /&gt;</code>
<a href="#">specialAttribute</a>	<code>&lt;test type="specialAttribute" name="part.number" value="123" /&gt;</code>	<code>&lt;specialAttribute name="part.number" value="123" /&gt;</code>
<a href="#">class</a>	<code>&lt;test type="class" name="MY_CLASS" /&gt;</code>	<code>&lt;class name="MY_CLASS" /&gt;</code>
<a href="#">type</a>	<code>&lt;test type="type" name="MY_TYPE" /&gt;</code>	<code>&lt;type name="MY_TYPE" /&gt;</code>
<a href="#">group</a>	<code>&lt;test type="group" name="MY_GROUP" /&gt;</code>	<code>&lt;group name="MY_GROUP" /&gt;</code>

**Table 1: Aliase instructions for test instruction**

## test

general syntax	<code>&lt;test type="typename" name="regexp" [value="regexp" regexpid="id_name"] /&gt;</code>		
description	Is used to execute tests on a specific component. If the test matches the condition, the return value is set to true, otherwise to false.		
attributes	Name	Use	Annotation
	type	required	Defines the type on which the test is executed. Valid values are: <ul style="list-style-type: none"> <li>• <a href="#">attribute</a></li> <li>• <a href="#">specialAttribute</a></li> <li>• <a href="#">class</a></li> <li>• <a href="#">group</a></li> <li>• <a href="#">type</a></li> </ul> Alternatively the test aliases mentioned above can be used.
	name	required	Defines the name of 'type'. The value can be either a string or any Regular Expression.
	value	optional	Defines the value of 'name'. The value can be either a string or any Regular Expression reference. If the optional attribute value is omitted, then the test will basically just execute an existence check. If the value attribute is defined, then the test result is true if the value matches the pattern.
	regexpid	optional	Defines the unique id within the namespace to reference the test using regular expressions. Using the regular expression, the reference of the regexpid can be divided into matching groups e.g. "(g_1)(g_2)(g_n)". If the regular expression of an attribute name or value is set to "(test_)[0-9]" the first matching group is "test_" and the second matching group is "[0-9]". If only one matching group exist i.e. "test_[0-9]", then the first matching group is "[0-9]".  To get the value of any matching group it is necessary to know if the value should come from the attribute name or from the attribute value:  <b>MyRegexpid.name[1]</b> returns the value of the first matching group from the attribute name.  <b>MyRegexpid.value[2]</b> returns the value of the second matching group from the attribute value.  The matching group <b>[0]</b> returns the complete value. For detailed information please see examples.

			Allowed character set: [p{Alnum}\_\-] = [a-z A-Z 0-9 _ -].
	invert	optional	Inverts the behavior of the test. If set to true and the criteria do not match the child instructions will be executed.
examples	<pre> &lt;!--tests the given class if the value equals AC, this is equivalent to a simple existence check --&gt; &lt;test type="class" value="AC" /&gt;  &lt;!--tests the given class if the name starts with AC by using a regular expression --&gt; &lt;test type="class" value="AC.*" /&gt;  &lt;!--tests the given attribute if the name starts with SYMBOL by using a regular expression--&gt; &lt;test type="attribute" name="SYMBOL_.*" &gt;   &lt;!-- add instructions here, if the name of the attribute starts with "SYMBOL " then the instructions included here are executed--&gt; &lt;/test&gt;  &lt;!-- tests if an attribute exist where the name starts with "SYMBOL_" by using a regular expression SYMBOL (.*) and the value starts with "4". If this is the case, it executes all child instructions in this test statement. The regexpid for the backreference is also configured which can be used in the child instructions to get the values of the attributes--&gt; &lt;test type="attribute" name="(SYMBOL_)(.*)" value="(4)(.*)"   regexpid="backreference1" &gt;    &lt;!-- tests if an attribute with the name SUB SYMBOL exist and also the value which contains the second matching group of the currently processed attribute SYMBOL_.* by using the backreference and regular expression.  Following attributes exists: SYMBOL 1=4711 and SUB SYMBOL=1. The mapping finds for e.g. the attribute with the name SYMBOL 1 and SUB_SYMBOL=1. Then all child instructions in this test statement will be executed, because the second matching group of the regexpid "backreference1.name[2]" returns the value 1. --&gt; &lt;test type="attribute" name="SUB_SYMBOL" value="{backreference1.name[2]}"&gt;    &lt;!-- value modification of an existing attribute with the name SYM. Using the matching group [0] in this example the new value from the attribute SYMBOL_1 is changed to 4711. " --&gt; &lt;modify target="attribute.value" name="SYM"   value="{backreference1.value[0]}" /&gt;  &lt;/test&gt; &lt;/test&gt; </pre>		

## Alias Instructions

### attribute

general syntax	<code>&lt;attribute name="regexp" [value="regexp" regexpid="id_name" invert="false"] /&gt;</code>		
description	Tests if an attribute exists. The optional value defined matches if the attribute has the given value. For each positive matching attribute all child instructions of the node will be executed. This test is an alias instruction for <a href="#">test</a> .		
attributes	Name	Use	Annotation
	name	required	Defines the name of attribute. It can be either a string or any Regular Expression.

	value	optional	Defines the value of 'name'. The value can be either a string or any Regular Expression reference. If the optional attribute value is omitted, then the test will basically just execute an existence check. If the value attribute is defined, then the test is successful if the value matches the pattern.
	regexpid	optional	Defines the unique id within the namespace to reference the test using regular expressions. For detailed description and examples see <a href="#">test</a>  Allowed character set: [p{Alnum}\_\-] = [a-z A-Z 0-9 _ -]
	invert	optional	The boolean value "true" inverts the behavior of the instruction. If the attribute is not configured the default value "false" is used and the behavior of the instruction is not inverted. If the value is set to true and the criteria do not match, the child instructions will be executed.
examples	<pre>&lt;!-- tests the given attribute if the name starts with SYMBOL_ by using a regular expression--&gt; &lt;attribute name="SYMBOL_(.*)" /&gt;  &lt;!-- tests the given attribute if the name starts with SYMBOL by using a regular expression and if its value starts with SYM --&gt; &lt;attribute name="SYMBOL_(.*)" value="SYM(.*)" /&gt;</pre>		

## specialAttribute

Special attributes are attributes used by the source and target plug-ins and define the processing of the component. Table 3: Available special attributes lists the available special attributes and the impact of them.

All following tests are aliases (shortcuts) for the test instruction.

Type	Full Statement	Alias Statement
<a href="#">class</a>	<code>&lt;test type="specialAttribute" name="class.name" value="MY_CLASS" /&gt;</code>	<code>&lt;class name="MY_CLASS" /&gt;</code>
<a href="#">type</a>	<code>&lt;test type="specialAttribute" name="type.name" value="MY_TYPE" /&gt;</code>	<code>&lt;type name="MY_TYPE" /&gt;</code>
<a href="#">group</a>	<code>&lt;test type="specialAttribute" name="group.name" value="MY_GROUP" /&gt;</code>	<code>&lt;group name="MY_GROUP" /&gt;</code>

**Table 2: Alias instructions for specialAttribute instruction**

Available special attributes.

Name	Description
<b>class.name</b>	Contains the name of the class of the part
<b>type.name</b>	Contains the name of the type of the part (not used currently)
<b>group.name</b>	Contains the name of the group of the part (not used currently)
<b>part.number</b>	Contains the part number of the part
<b>part.delete</b>	This attribute marks the part to be deleted if value="true". (only applicable to delete parts on the EDA side) In the preferences of the Connector the user can set to used it or not. If this setting is not used local parts are updated and parts which do not exist are created. Otherwise local parts are deleted how it is configured in the mapping configuration.

**Table 3: Available special attributes**

general syntax	<code>&lt;specialAttribute name="regex" [value="regex" regexpid="my_id" invert="false"] /&gt;</code>		
description	Tests if a special attribute exists. The optional value defined matches if the attribute has the given value. For each positive matching special attribute all child instructions of the node will be executed. This test is an alias instruction for <a href="#">test</a> .		
attributes	Name	Use	Annotation

	name	required	Defines the name of attribute. It can be either a string or any Regular Expression.
	value	optional	Defines the value of 'name'. The value can be either a string or any Regular Expression reference . If the optional attribute value is omitted, then the test will basically just execute an existence check. If the value attribute is defined, then the test is successful if the value matches the pattern.
	regexpid	optional	Defines the unique id within the namespace to reference the test using regular expressions. For detailed description and examples see <a href="#">test</a>  Allowed character set: [p{Alnum}\_\-] = [a-z A-Z 0-9 _ -]
	invert	optional	The boolean value "true" inverts the behavior of the instruction. If the attribute is not configured the default value "false" is used and the behavior of the instruction is not inverted. If the value is set to true and the criteria do not match, the child instructions will be executed.
examples	<pre> &lt;!-- tests if a special attribute class.name with value AC exists.       If exist execute the client instructions --&gt; &lt;specialAttribute name="class.name" value="AC"&gt;   ...client instructions... &lt;/specialAttribute&gt;  &lt;!-- tests if a special attribute class.name exist and the value       starts of the the class.name with SYM. If exist execute       the client instructions--&gt; &lt;specialAttribute name="class.name" value="SYM(.*)"&gt;   ...client instructions... &lt;/specialAttribute&gt;  &lt;!-- tests if a special attribute part.number ant with value 123 exist.       If exist execute the client instructions --&gt; &lt;specialAttribute name="part.number" value="123"&gt;   ...client instructions... &lt;/specialAttribute&gt;  &lt;!-- tests if a special attribute part.delete with the value true exist.       If exist execute the client instructions --&gt; &lt;specialAttribute name="part.delete" value="true"&gt;   ...client instructions... &lt;/specialAttribute&gt; </pre>		

## attributeExists

general syntax	<code>&lt;attributeExists name="regexp" [value="regexp" regexpid="id_name" invert="false"] /&gt;</code>		
description	Tests if an attribute exists. The optional value defined matches if the attribute has the given value. When found first matching attribute all child instructions of the node will be executed (no further test if more than one matches available and no further execution of the child instruction for further matches). This test is an alias instruction for <a href="#">test</a> .		
attributes	Name	Use	Annotation
	name	required	Defines the name of attribute. It can be either a string or any Regular Expression.



	value	optional	Defines the value of 'name'. The value can be either a string or any Regular Expression reference. If the optional attribute value is omitted, then the test will basically just execute an existence check. If the value attribute is defined, then the test is successful if the value matches the pattern.
	regexpid	optional	Defines the unique id within the namespace to reference the test using regular expressions. For detailed description and examples see <a href="#">test</a>  Allowed character set: [p{Alnum}\_\-] = [a-z A-Z 0-9 _ -]
	invert	optional	The boolean value "true" inverts the behavior of the instruction. If the attribute is not configured the default value "false" is used and the behavior of the instruction is not inverted. If the value is set to true and the criteria do not match, the child instructions will be executed.
examples	<pre> &lt;!-- tests if at least one attributes name starts with SYMBOL_ by using a regular expression--&gt; &lt;attributeExists name="SYMBOL_.*" /&gt;  &lt;!-- tests if at least one attributes name is starting with SYMBOL by using a regular expression and if its value starts with SYM exists --&gt; &lt;attributeExists name="SYMBOL_.*" value="SYM.*"/&gt; </pre>		

## partNumber

general syntax	<code>&lt;partNumber [value="regexp" regexpid="id_name" invert="false"] /&gt;</code>		
description	Tests the existence of the part number (if is not null). If the part number is null(not set) always returning false. If the optional value defined matches returning true.		
attributes	Name	Use	Annotation
	value	optional	Defines the value of the part number The value can be either a string or any Regular Expression reference. If the optional attribute value is omitted, then the test will basically just execute an existence check. If the value attribute is defined, then the test is successful if the value matches the pattern.
	regexpid	optional	Defines the unique id within the namespace to reference the test using regular expressions. For detailed description and examples see <a href="#">test</a>  Allowed character set: [p{Alnum}\_\-] = [a-z A-Z 0-9 _ -]  Note: When regexpid is used outside the tag and the value is not match an illegal state exception is thrown. (See correct example)
	invert	optional	The boolean value "true" inverts the behavior of the instruction. If the attribute is not configured the default value "false" is used and the behavior of the instruction is not inverted. If the value is set to true and the criteria do not match, the child instructions will be executed.
examples	<pre> &lt;!-- tests if the part number is starting with CAP- regular expression and if its value starts with CAP exists --&gt; &lt; partNumber value="CAP-(.*)" invert="false" /&gt;  &lt;!-- If the partNumber value starts with "Test" a regexpid --&gt; &lt;partNumber value="Test-(.*)" regexpid="PART_NUMBER_WITH_PREFIX"&gt;     &lt;!--Set part number value from regexpid exclude "Test"--&gt;     &lt;setPartNumber value="\\${PART_NUMBER_WITH_PREFIX.value[1]}/&gt; </pre>		

</partNumber>

## class

general syntax	<code>&lt;class [value="regexp" regexpid="my_id" invert="false"] /&gt;</code>		
description	Test the class name of the component. This test is an alias instruction for <a href="#">test</a> which is an alias instruction for <a href="#">specialAttribute</a> .		
attributes	Name	Use	Annotation
	value	optional	It can be either a string or any Regular Expression reference. If the optional attribute value is omitted, then the test will basically just execute an existence check. If the value attribute is defined, then the test is successful if the value matches the pattern.
	regexpid	optional	Defines the unique id within of the namespace to reference the test using regular expressions. For detailed description and examples see also <a href="#">test</a> Allowed character set: <code>[p{Alnum}\_\-] = [a-z A-Z 0-9 _ -]</code>
	invert	optional	The boolean value "true" inverts the behavior of the instruction. If the attribute is not configured the default value "false" is used and the behavior of the instruction is not inverted. If the value is set to true and the criteria do not match, the child instructions will be executed.
examples	<pre>&lt;!-- tests if a special attribute class.name exists / is defined --&gt; &lt;class /&gt;  &lt;!-- tests if class name equals AC --&gt; &lt;class value=" AC" /&gt;  &lt;!-- tests if class name starts with AC --&gt; &lt;class value="AC(.*)" /&gt;</pre>		

## group

General syntax	<code>&lt;group[ value="regexp" regexpid="my_id" invert="false"] /&gt;</code>		
description	Test on the group name of the component. This test is an alias instruction for <a href="#">test</a> and This test is an alias instruction for <a href="#">specialAttribute</a> .		
attributes	Name	Use	Annotation
	value	optional	Defines the value of 'name'. It can be either a string or any Regular Expression reference . If the optional attribute value is omitted, then the test will basically just execute an existence check. If the value attribute is defined, then the test is successful if the value matches the pattern.
	regexpid	optional	Defines the unique id within of the namespace to reference the test using regular expressions. For detailed description and examples see also <a href="#">test</a> Allowed character set: <code>[p{Alnum}\_\-] = [a-z A-Z 0-9 _ -]</code>
	invert	optional	The boolean value "true" inverts the behavior of the instruction. If the attribute is not configured the default value "false" is used and the behavior of the instruction

			is not inverted. If the value is set to true and the criteria do not match, the child instructions will be executed.
examples	<pre> &lt;!-- tests if a special attribute group.name exists / is defined--&gt; &lt;group /&gt;  &lt;!-- tests if group name equals AC --&gt; &lt;group value=" AC" /&gt;  &lt;!-- tests if class name starts with AC--&gt; &lt;group value="AC(*)" /&gt; </pre>		

## type

general syntax	<code>&lt;type [value="regexp" regexpid="my_id" invert="false"] /&gt;</code>		
description	Test on the type name of the component. This test is an alias instruction for <a href="#">test</a> and This test is an alias instruction for <a href="#">specialAttribute</a> .		
attributes	Name	Use	Annotation
	value	optional	Defines the value of 'name'. The value can be either a string or any Regular Expression reference . If the optional attribute value is omitted, then the test will basically just execute an existence check. If the value attribute is defined, then the test is successful if the value matches the pattern.
	regexpid	optional	Defines the unique id within of the namespace to reference the test using regular expressions. For detailed description and examples see also <a href="#">test</a> Allowed character set: [p{Alnum}\_\-] = [a-z A-Z 0-9 _-]
	invert	optional	The boolean value "true" inverts the behavior of the instruction. If the attribute is not configured the default value "false" is used and the behavior of the instruction is not inverted. If the value is set to true and the criteria do not match, the child instructions will be executed.
examples	<pre> &lt;!-- tests if a special attribute type.name exists / is defined --&gt; &lt;type /&gt;  &lt;!-- tests if type name equals AC --&gt; &lt;type value=" AC" /&gt;  &lt;!-- tests if type name starts with AC --&gt; &lt;type value="AC(*)" /&gt; </pre>		

## Operations

### duplicate

general syntax	<pre>&lt;duplicate&gt;   ...child instructions... &lt;/duplicate&gt;</pre>
description	Duplicates the component and executes the instructions on it. Adds the duplicate to the list of the duplicated components before exiting the element.
examples	<pre>&lt;!-- duplicates the component and add attribute A1 with a value V1--&gt; &lt;duplicate&gt;   &lt;add target="attribute" name="A1" value="V1" /&gt; &lt;/duplicate&gt;</pre>

### add

general syntax	<pre>&lt;add target="attribute" name="regexp" [value="regexp"] /&gt;</pre>		
description	Adds a new attribute with given value to the component. The value is optional, if not defined it will be set to "null". The attribute is only added if it not exists. Else add will be silently omitted.		
attributes	Name	Use	Annotation
	target	required	The target defining the type of the operation to execute. Valid values are: <ul style="list-style-type: none"> <li><b>attribute</b>: defines to add an attribute.</li> </ul>
	name	required	Defines the new name of the target. The value can be either a string or any Regular Expression reference.
	value	optional	Defines the new value of the target. The value can be either a string or any Regular Expression reference. Defaults to null if omitted.
examples	<pre>&lt;!-- adds a new attribute named SYMBOL and set its value to 1234 --&gt; &lt;add target="attribute" name="SYMBOL" value="1234" /&gt;</pre>		

### modify

general syntax	<pre>&lt;modify target="targetname" [name="regexp"] value="regexp" /&gt;</pre>		
description	Modifies a name or value of the given target. The target can be a special attribute value, an attribute or an attribute value.		
attributes	Name	Use	Annotation
	target	required	The target on which the type of the operation is executed. Valid values are. Defines an attribute type. <ul style="list-style-type: none"> <li><b>attribute.name</b>: selector representing the name of an attribute,</li> <li><b>attribute.value</b>: selector representing the value of an attribute</li> </ul> class : represents a class <ul style="list-style-type: none"> <li><b>class.name</b>: selector representing the name of a class, meaning the class itself</li> <li><b>part.number</b>: selector representing the part number of the component</li> <li><b>part.delete</b>: selector marking the part to be deleted. (boolean value="true")</li> </ul> type: not used currently group : not used currently

	value	required	Attribute defining the new value of the target selector. The value can be either a string or any Regular Expression reference.
	name	required	Attribute defining the name of the target itself. The value can be either a string or any Regular Expression. <b>Optional for class.</b>
examples	<pre> &lt;!-- modification of an existing attribute named SYMBOL by renaming it to COMP_PROPERTY (the name of the attribute is required)--&gt; &lt;modify target="attribute.name" name=" SYMBOL" value=" COMP_PROPERTY " /&gt;  &lt;!-- modification of an existing attribute named SYMBOL by changing its value to COMP_PROPERTY (the name attribute is here required)--&gt; &lt;modify target="attribute.value" name="SYMBOL" value=" COMP_PROPERTY " /&gt;  &lt;!-- modification of the current class by renaming it to AC --&gt; &lt;modify target="class.name" value=" AC" /&gt;  &lt;!-- modification of the class named RESISTOR by renaming it to AC. If no name is specified any class will be renamed to AC when this instruction will be executed --&gt; &lt;modify target="class.name" name="Resistor" value=" AC" /&gt;  &lt;!-- modification of the class named RESISTOR by renaming it to AC using a class test --&gt; &lt;class name=" Resistor" /&gt; &lt;!--modification of the current class by renaming it to AC ? &lt;modify target= class.name value= AC /&gt; --&gt; &lt;class /&gt;  &lt;!-- modification of special attribute part.number to the value 123--&gt; &lt;modify target="part.number" value="123" /&gt;  &lt;!-- modification of special attribute part.delete to value true --&gt; &lt;modify target="part.delete" value="true" /&gt; </pre>		

## set

general syntax	<code>&lt;set target="targetname" name="regexp" value="regexp" /&gt;</code>		
description	Sets an attribute to a value if exists, if not adds the attribute and sets the value then.		
attributes	Name	Use	Annotation
	target	required	The target on which the type of the operation is executed. Valid values are: <ul style="list-style-type: none"> <li><b>attribute</b>: defines an attribute type</li> </ul>
	name	required	Defines the new name of the target. The value can be either a string or any Regular Expression.
	value	required	Defines the new value of the target. The value can be either a string or any Regular Expression.
examples	<pre> &lt;!-- sets a new attribute named SYMBOL with value 1234 if it does not exist else the value of the attribute is changed to the specified one --&gt; &lt;set target="attribute" name="SYMBOL" value="1234" /&gt; </pre>		

## setSpecialAttribute

general syntax	<code>&lt;set name="regexp" value="regexp" /&gt;</code>		
description	Sets a special attribute to a value if exists, if not adds the special attribute and sets the value then.		

attributes	Name	Use	Annotation
	name	required	Defines the new name of the target. The value can be either a string or any Regular Expression.
	value	required	Defines the new value of the target. The value can be either a string or any Regular Expression.
examples	<pre>&lt;!-- sets a new attribute named SYMBOL with value 1234 if it does not exist else the value of the attribute is changed to the specified one --&gt; &lt;setSpecialAttribute name=" part.interface.name" value="Interface Name" /&gt;</pre>		

## setPartNumber

general syntax	<code>&lt;setPartNumber value="regexp" /&gt;</code>		
description	Sets a the part number to specified value.		
attributes	Name	Use	Annotation
	value	required	Defines the new value of the target. The value can be either a string or any Regular Expression.
examples	<pre>&lt;!-- sets Partnumber with value 12345 --&gt; &lt;setPartNumber value="12345" /&gt;</pre>		

## setEnoviaLink

general syntax	<code>&lt;setEnoviaLink name="&lt;attribute-name&gt;" /&gt;</code>		
description	<p>Sets a link (URL- with an object ID) into an attribute of the mapped part from 3DEXPERIENCE into a local data base. Can be used only in the mapping configuration from 3DEXPERIENCE to ECAD.</p> <p>Note: Configure maximum length of the column in the data base. The length of the URL from a part can be very long.</p>		
attributes	Name	Use	Annotation
	name	required	Defines the attribute/column name of the link to the part in 3DEXPERIENCE.
examples	<pre>&lt;!--set a link to a part in 3DEXPERIENC --&gt; &lt;setEnoviaLink name="3DS_Part" /&gt;</pre>		

## remove

general syntax	<code>&lt;remove target="targetname" name="regexp"/&gt;</code>		
description	Removes an attribute from the attribute list.		
attributes	Name	Use	Annotation
	target	required	<p>The target on which the type of operation is executed.</p> <p>Valid values are:</p> <ul style="list-style-type: none"> <li><b>attribute</b>: defines an attribute type</li> </ul>
	name	required	The name of the 'target' to remove.
examples	<pre>&lt;!-- removes the attribute named SYMBOL --&gt; &lt;remove target="attribute" name="SYMBOL" /&gt;</pre>		

## removeSpecialAttribute

general syntax	<code>&lt;removeSpecialAttribute target="targetname" name="regexp"/&gt;</code>		
description	Removes a special attribute from the special attribute list.		
attributes	Name	Use	Annotation
	target	required	The target on which the type of operation is executed. Valid values are: <ul style="list-style-type: none"> <li><b>attribute</b>: defines an attribute type</li> </ul>
	name	required	The name of the 'target' to remove.
examples	<pre> &lt;!-- removes the special attribute named part.interface.name --&gt; &lt;removeSpecialAttribute target="attribute" name="part.interface.name" /&gt; </pre>		

## concatAttribute

general syntax	<pre> &lt;concatAttribute target="targetname" name="regexp" value="attributelist" separator="separatorstring" [maxLength="maxlength" defaultValueForNotExistent="attributelist"] /&gt; </pre>		
description	Sets an attribute value created from values of the attributes in the <b>attributeList</b> separated by the given separator. Each attribute has to be surrounded by a single quote ('). If maximum length defined, the value will be cropped to the <b>maxLength</b> . If an attribute from the list does not exist the <b>defaultValueForNotExistent</b> will be used. If the <b>defaultValueForNotExistent</b> is not defined, the not found attribute will be ignored (also the separator will be not added).		
attributes	Name	Use	Annotation
	target	required	The target on which the type of operation is executed. Valid values are: <ul style="list-style-type: none"> <li><b>attribute</b>: defines an attribute type</li> </ul>
	name	required	The name of the target to set. The value can be either a string or any Regular Expression.
	value	required	List of attribute names (each in single quotes (') to be concatenated).
	separator	required	The string to use to separate the entries of the value.
	maxLength	optional	The maximum length of the value. If the concatenated (inclusive separators) is longer than the max value it will be cut to maximum length. Vaule needs to be a non negative integer.
	defaultValu eForNotExi stent	optional	This value will be used if an attribute from the attribute list does not exists in the component. If not set those attributes will be ignored (inclusive separator).
examples	<pre> &lt;!-- concatenates the attributes PART NAME and VALUE separated by   to value of the DESCRIPTION attribute. If the concatenated value has more than 20 it will be cropped to 20 characters. If for e.g. no VALUE attribute exists a ? will be used instead. - PART NAME=res, VALUE=100 -&gt; DESCRIPTION=res 100 - PART NAME=myNewPart, &lt;no value attribute&gt; -&gt; DESCRIPTION=myNewPart ? --&gt; &lt;concatAttribute target="attribute" name="DESCRIPTION" value="PART_NAMEVALUE" </pre>		

	<pre>separator=" " maxLength="10" defaultValueForNotExistent="?" /&gt;</pre>
--	--

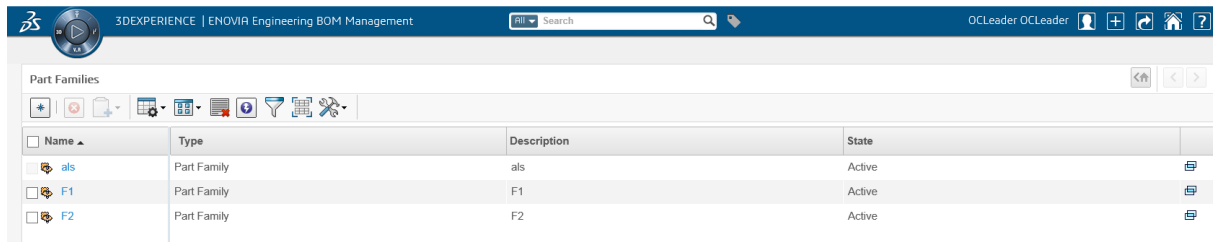
## log

general syntax	<code>&lt;log message="log messages" level="&lt;loglevel&gt;" /&gt;</code>		
description	Logging configuration of the mapping with different log levels. The logs are displayed in CDM.log.		
attributes	Name	Use	Annotation
	message	required	Log messages displayed in CDM.log Special attribute name and attribute names in single quotes (e.g. 'ATTR1') displays the value of the attribute.
	level	required	Log level configuration Values are: <ul style="list-style-type: none"> <li>- info</li> <li>- warn</li> <li>- debug</li> <li>- trace</li> <li>- error (error message is displayed in an error dialog)</li> </ul>
examples	<pre>&lt;log message="info log messages" level="info" /&gt; &lt;log message="warn log messages" level="warn" /&gt; &lt;log message="debug log messages" level="debug" /&gt; &lt;log message="error log messages" level="error" /&gt;  &lt;test type="specialAttribute" name="part.number" value="C0603-100N-25-5" regexpid="partNumber1"&gt;   &lt;test type="attribute" name="PARTVALUE" value=".*" regexpid="test"&gt;     &lt;log message="\${test.name[0]}: \${test.value[0]}" level="trace" /&gt;     &lt;modify target="attribute.value" name="PARTVALUE" value="0,0000001"   /&gt;     &lt;log message="'part.number' - \${test.name[0]}: old: \${test.value[0]} new: 'PARTVALUE'" level="trace" /&gt;   &lt;/test&gt;   &lt;log message="'part.number' 'PARTVALUE'" level="info" /&gt; &lt;/test&gt;</pre>		



# Part and IP Classification Mapping Configuration

The figure below shows the Part Families under 3DEXPERIENCE Library.



Name	Type	Description	State
als	Part Family	als	Active
F1	Part Family	F1	Active
F2	Part Family	F2	Active

Figure 10 Part Families in Part and IP Classification

## Upload of Parts

A part in the ECAD library can be mapped to the Part Family in 3DEXPERIENCE using the following config.

```
<?xml version="1.0" encoding="UTF-8"?>
<pemapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="mapping-schema.xsd"
  version="1.0">
  <namespace>
    <modify value="description" name="DESCR" target="attribute.name"/>

    <class value="ECAD_IC">
      <removeSpecialAttribute target="attribute" name="class.name"/>
      <setSpecialAttribute value="/Library1/IC/ASIC" name="part.interface.name"/>
    </class>
  </namespace>
</pemapping>
```

In the above config, all parts coming from the ECAD library “*ECAD\_IC*” will be upload to part families namely “*ASIC*” in 3DEXPERIENCE.

## Download of Parts

A part in 3DEXPERIENCE which is started in a Part Family can be mapped to ECAD Library using the following config.

```
<?xml version="1.0" encoding="UTF-8"?>
<pemapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="mapping-schema.xsd" version="1.0">

  <namespace>
    <modify name="ECADVALUE" value="VALUE" target="attribute.name" />
    <modify name="description" value="DESCR" target="attribute.name" />

    <specialAttribute value="/Library1/IC/ASIC" name="part.interface.name">
      <modify value=" IC" target="class.name" />
    </specialAttribute>
  </namespace>
</pemapping>
```

In the above config, all parts coming from the 3DEXPERIENCE Part Family “*ASIC*” will be download to ECAD libraries namely “*ECAD\_IC*”.

## Multiclassification

The user can use Part and IP Classification configuration to multiclassify the parts. The multiclassification is supported during upload of parts from ECAD library to 3DEXPERIENCE and vice versa.

## Upload of Parts

A part from ECAD Library can be classified into more than one part family in 3DEXPERIENCE. This can be done using the following example config.

### Example 1

```
<?xml version="1.0" encoding="UTF-8"?>
<pemapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="mapping-schema.xsd"
  version="1.0">
  <namespace>
    <modify value="description" name="DESCR" target="attribute.name"/>

    <class value="ECAD_IC">
      <removeSpecialAttribute target="attribute" name="class.name"/>
      <setSpecialAttribute value="/Library1/IC/ASIC|Library2/IC"
name="part.interface.name"/>
    </class>

  </namespace>
</pemapping>
```

In the above config, all parts coming from the ECAD library “*ECAD\_IC*” will be multiclassified into two part families namely “*ASIC*” from Library1 and “*DIODE*” from Library2 in 3DEXPERIENCE.

### Example 2

```
<?xml version="1.0" encoding="UTF-8"?>
<pemapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="mapping-schema.xsd"
  version="1.0">
  <namespace>
    <class value="als">
      <modify value="Part" target="class.name"/>
      <removeSpecialAttribute target="attribute" name="class.name"/>
      <setSpecialAttribute value="/Library1/IC/ALS|Library1/IC/ALS TEST"
name="part.interface.name"/>
      <attribute name="SymCount" value="1">
        <specialAttribute name="part.interface.name" value="(
Library1/IC/ALS) ([|]) (Library1/IC/ALS.*)" regexpid="v">
          <setSpecialAttribute value="{v.value[0]}| Library1/IC"
name="part.interface.name"/>
        </specialAttribute>
      </attribute>
    </class>
    <modify value="Part Name" name="Part_Name" target="attribute.name"/>
    <modify value="Path" name="Path" target="attribute.name"/>
    <modify value="Tk1" name="Tk1" target="attribute.name"/>
    <modify value="Symbol" name="SymCount" target="attribute.name"/>
  </namespace>
</pemapping>
```

In the above config, all parts coming from the ECAD library “*als*” will be multiclassified into two part families namely “*ALS* and *ALS TEST*” in 3DEXPERIENCE. Later an attribute check is performed. If the attribute “*SymCount*” with value “*1*” exists then parts are classified to the 3DEXPERIENCE part family “*IC*”. Hence the parts are multiclassified into three part families in 3DEXPERIENCE.

## Download of Parts

A part from 3DEXPERIENCE can be classified into more than one ECAD Library. This can be done using the following example config.

```
<?xml version="1.0" encoding="UTF-8"?>
<pemapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="mapping-schema.xsd" version="1.0">
  <namespace>
    <modify name="ECADVALUE" value="VALUE" target="attribute.name" />
    <modify name="description" value="DESCR" target="attribute.name" />
    <specialAttribute value="/Library1/DIODE" name="part.interface.name">
      <modify value="ECAD_DIODE1" target="class.name" />
    </specialAttribute>
    <duplicate>
      <specialAttribute value="/Library1/DIODE" name="part.interface.name">
        <modify value="ECAD_DIODE2" target="class.name" />
      </specialAttribute>
    </duplicate>
  </namespace>
</pemapping>
```

In the above config, all parts coming from the 3DEXPERIENCE Part Family “*DIODE*” will be multiclassified into two ECAD libraries namely “*ECAD\_DIODE1* and *ECAD\_DIODE2*.”

## Background Synchronization

The background synchronization process itself is executed by running the “<ROOT>\bin\ENOVIACDMSyncBackground.exe” and applying additional parameters.

After executing of the background synchronization from 3DEXPERIENCE to the ECAD library a detailed report file (detailed-report.txt) of the synchronization is created in the library project directory. Synchronization from ECAD to 3DEXPERIENCE does not create a detailed report file.

The following parameters need to be passed over to the binary:

-proj \$libraryproject\$	This defined the library project to be used. The full qualified path to the Library Project File has to be provided
-mode enovia eda	This is the direction which is used: <ul style="list-style-type: none"> <li>- enovia →synchronizes the parts from 3DEXPERIENCE to the ECAD Library. Only PDM selections can be used.</li> <li>- eda → synchronizes the parts from the ECAD Library to 3DEXPERIENCE. Only EDA selections can be used.</li> </ul>
-edaselection \$edaselection\$	This is the name of the file which contains the predefined selection of eda classes to be synchronized. Only state the name of the file. If this argument is not passed, an error will be thrown.
-pdmselection \$pdmselection\$	This is the name of the file which contains the predefined selection of 3DEXPERIENCE type classes to be synchronized. Only state the name of the file If this argument is not passed, an error will be thrown.
Prefix of the selection	For 3DEXPERIENCE Engineering BOM Management types: <ul style="list-style-type: none"> <li>- CDM_Preselection_EDA_&lt;NAME&gt;</li> <li>- CDM_Preselection_PDM_&lt;NAME&gt;</li> </ul> For 3DEXPERIENCE Part and IP Classification Libraries: <ul style="list-style-type: none"> <li>- CDM_Preselection_EDA_&lt;NAME&gt;</li> <li>- CDM_Preselection_PARTLIBRARY_&lt;NAME&gt;</li> </ul>

Example 3DEXPERIENCE to EDA:

```
ENOVIACDMSyncBackground.exe -proj C:\designs\library\lib.lmc -mode enovia
-pdmselection CDM_Preselection_PDM_Resistor
```

Example EDA to 3DEXPERIENCE:

```
ENOVIACDMSyncBackground.exe -proj C:\designs\library\lib.lmc -mode eda -
edaselection CDM_Preselection_EDA_Capacitor
```