

Allegro® EDM

Data Exchange Reference Guide

Product Version 23.1
September 2023

© 2023 Cadence Design Systems, Inc. All rights reserved.

Portions © Apache Software Foundation, Sun Microsystems, Free Software Foundation, Inc., Regents of the University of California, Massachusetts Institute of Technology, University of Florida. Used by permission. Printed in the United States of America.

Cadence Design Systems, Inc. (Cadence), 2655 Seely Ave., San Jose, CA 95134, USA.

Allegro EDM contains technology licensed from, and copyrighted by: Apache Software Foundation, 1901 Munsey Drive Forest Hill, MD 21050, USA © 2000-2005, Apache Software Foundation. Sun Microsystems, 4150 Network Circle, Santa Clara, CA 95054 USA © 1994-2007, Sun Microsystems, Inc. Free Software Foundation, 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA © 1989, 1991, Free Software Foundation, Inc. Regents of the University of California, Sun Microsystems, Inc., Scriptics Corporation, © 2001, Regents of the University of California. Daniel Stenberg, © 1996 - 2006, Daniel Stenberg. UMFPACK © 2005, Timothy A. Davis, University of Florida, (davis@cise.ulf.edu). Ken Martin, Will Schroeder, Bill Lorensen © 1993-2002, Ken Martin, Will Schroeder, Bill Lorensen. Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, Massachusetts, USA © 2003, the Board of Trustees of Massachusetts Institute of Technology. vtkQt, © 2000-2005, Matthias Koenig. All rights reserved.

Trademarks: Trademarks and service marks of Cadence Design Systems, Inc. contained in this document are attributed to Cadence with the appropriate symbol. For queries regarding Cadence's trademarks, contact the corporate legal department at the address shown above or call 800.862.4522.

Open SystemC, Open SystemC Initiative, OSCI, SystemC, and SystemC Initiative are trademarks or registered trademarks of Open SystemC Initiative, Inc. in the United States and other countries and are used with permission. All other trademarks are the property of their respective holders.

Restricted Permission: This publication is protected by copyright law and international treaties and contains trade secrets and proprietary information owned by Cadence. Unauthorized reproduction or distribution of this publication, or any portion of it, may result in civil and criminal penalties. Except as specified in this permission statement, this publication may not be copied, reproduced, modified, published, uploaded, posted, transmitted, or distributed in any way, without prior written permission from Cadence. Unless otherwise agreed to by Cadence in writing, this statement grants Cadence customers permission to print one (1) hard copy of this publication subject to the following conditions:

1. The publication may be used only in accordance with a written agreement between Cadence and its customer.
2. The publication may not be modified in any way.
3. Any authorized copy of the publication or portion thereof must include all original copyright, trademark, and other proprietary notices and this permission statement.
4. The information contained in this document cannot be used in the development of like products or software, whether for internal or external use, and shall not be used for the benefit of any other party, whether or not for consideration.

Disclaimer: Information in this publication is subject to change without notice and does not represent a commitment on the part of Cadence. Except as may be explicitly set forth in such agreement, Cadence does not make, and expressly disclaims, any representations or warranties as to the completeness, accuracy or usefulness of the information contained in this document. Cadence does not warrant that use of such information will not infringe any third party rights, nor does Cadence assume any liability for damages or costs of any kind that may result from use of such information. Cadence is committed to using respectful language in our code and communications. We are also active in the removal and/or replacement of inappropriate language from existing content. This product documentation may however contain material that is no longer considered appropriate but still reflects long-standing industry terminology. Such content will be addressed at a time when the related software can be updated without end-user impact.

Restricted Rights: Use, duplication, or disclosure by the Government is subject to restrictions as set forth in FAR52.227-14 and DFAR252.227-7013 et seq. or its successor.

Contents

<u>Preface</u>	7
<u>About This Guide</u>	7
<u>Intended Audience</u>	7
<u>Related Documentation</u>	7
<u>Typographic and Syntax Conventions</u>	7

1

<u>Working with Libraries External to Allegro EDM</u>	9
<u>Basic Synchronization Framework Flow</u>	9
<u>Synchronization Framework Components</u>	13
<u>Input Data Sources</u>	13
<u>Sync Configuration File</u>	14
<u>Sync Engine</u>	14
<u>Sync System Settings File</u>	14
<u>Advantages of Synchronization Framework</u>	15
<u>Two-Way Synchronization between Allegro EDM and External Systems</u>	17
<u>Using a Query File when Exporting Allegro EDM Data</u>	18
<u>Creating a Query File from Database Editor</u>	18
<u>Manually Creating a Query File</u>	20
<u>Query Section</u>	21
<u>Attributes Section</u>	22
<u>Relations Section</u>	22
<u>Interface Section</u>	23
<u>Using Synchronization Framework from the Command Line</u>	24
<u>Basic Command</u>	24
<u>Complete Syntax</u>	25

2

<u>Managing Synchronization Sessions</u>	27
<u>The settings.sync File</u>	28

3

<u>Defining Import Policies</u>	35
<u>Structure of the sync.xml file</u>	35
<u>Rules Section</u>	35
<u>Tags</u>	36
<u>Types Section</u>	36
<u>Interface Section</u>	36
<u>Object Rules</u>	36
<u>Condition - TargetNotExist</u>	37
<u>Condition - TargetExist</u>	37
<u>Condition - TargetBeingModified</u>	38
<u>Condition - TargetRevisionLower</u>	39
<u>Condition - TargetRevisionHigher</u>	40
<u>Condition - TargetRevisionEqual</u>	40
<u>Relation Rules</u>	40
<u>Condition - ObjectUpdated</u>	41
<u>Condition - TargetTypeNotExist</u>	41
<u>Condition - TargetTypeExist</u>	41
<u>Condition - TargetNameNotExist</u>	42
<u>Condition - TargetNameExist</u>	42
<u>Condition - TargetRevisionNotExist</u>	43
<u>Condition - TargetRevisionExist</u>	43
<u>Condition - ObjectDeferred</u>	43
<u>Condition - ObjectCreated</u>	43
<u>Condition - ObjectMajor</u>	44
<u>Condition - ObjectMinor</u>	44
<u>Condition - ObjectRetained</u>	44
<u>Condition - ObjectIgnored</u>	44
<u>Condition - Default</u>	44
<u>Attribute Rules</u>	44
<u>Condition - ObjectUpdated</u>	45
<u>Condition - ObjectDeferred</u>	45
<u>Condition - ObjectCreated</u>	45
<u>Condition - ObjectMajor</u>	45
<u>Condition - ObjectMinor</u>	45

Allegro EDM Data Exchange Reference Guide

<u>Condition - ObjectRetained</u>	45
<u>Condition - ObjectIgnored</u>	46
<u>Condition - Default</u>	46
<u>Lifecycle Rules</u>	47
<u>Conditions for Lifecycle Rules</u>	47
<u>Subconditions for Lifecycle Rules</u>	47
<u>Subcondition - SourcePreliminary</u>	48
<u>Subcondition - SourceCheckedOut</u>	48
<u>Subcondition - SourceCheckedIn</u>	48
<u>Subcondition - SourceReleased</u>	48
<u>Subcondition - SourceDeleted</u>	48
<u>Trigger Rules</u>	49
<u>Attribute Trigger</u>	49
<u>Relation Trigger</u>	50
<u>Sync Type</u>	51
<u>Sync Attributes</u>	52
<u>Sync Relations</u>	52
<u>Syntax of sync.xml</u>	52
<u>A</u>	
<u>Mapping Samples</u>	55
<u>Implementing Pattern Matching</u>	56
<u>Examples</u>	57
<u>Index</u>	59

Allegro EDM Data Exchange Reference Guide

Preface

About This Guide

This *Allegro® EDM Data Exchange Reference Guide* explains:

- How to synchronize data from external sources
- How to export data from Allegro Engineering Data Management (EDM) system
- The components and structure of the synchronization framework

Intended Audience

This guide is for anyone who needs to understand the synchronization framework and set up the logic definition file.

Related Documentation

You can also refer the following documentation to know more about related tools and methodologies:

- For learning how to use Library Import, see *Allegro EDM Library Import User Guide*.
- For learning how to use Library Distribution, see *Allegro EDM Library Distribution User Guide*.

Typographic and Syntax Conventions

This list describes the syntax conventions used for this user guide:

`literal`

Nonitalic words indicate keywords that you must enter literally. These keywords represent command (function, routine) or option names.

Allegro EDM Data Exchange Reference Guide

Preface

<i>argument</i>	Words in italics indicate user-defined arguments for which you must substitute a name or a value.
	Vertical bars (OR-bars) separate possible choices for a single argument. They take precedence over any other character.
[]	Brackets denote optional arguments. When used with OR-bars, they enclose a list of choices. You can choose one argument from the list.
{ }	Braces are used with OR-bars and enclose a list of choices. You must choose one argument from the list.

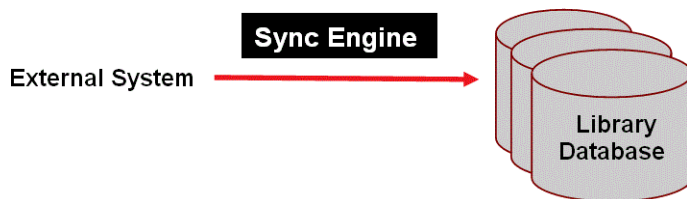
Working with Libraries External to Allegro EDM

Librarians frequently need to merge libraries or transfer data from one library to another. Many Allegro EDM applications are frequently used to:

- import and export data from and into the Allegro EDM Component Database
- check for differences between the data from an external source and the data already in the Allegro EDM database
- import libraries into Allegro EDM

For data to be imported into Allegro EDM, utilities for each of these tasks require the data to be in specific formats. The data must also meet certain conditions.

To create a standard import process that works for all these utilities and to import external libraries that might not always in the same structure or format because they are site-specific and, Allegro EDM offers a synchronization framework that acts as the core logic engine. This logic engine is the interface that specifies how the EDM utilities process the data being imported.



Basic Synchronization Framework Flow

Consider an example: There are four different sources of data called `site1`, `site2`, `site3`, and `site4`, and each one has the ability to make its data available in the XML format. The resulting XML files are called `XML1`, `XML2`, `XML3`, and `XML4`.

Allegro EDM Data Exchange Reference Guide

Working with Libraries External to Allegro EDM

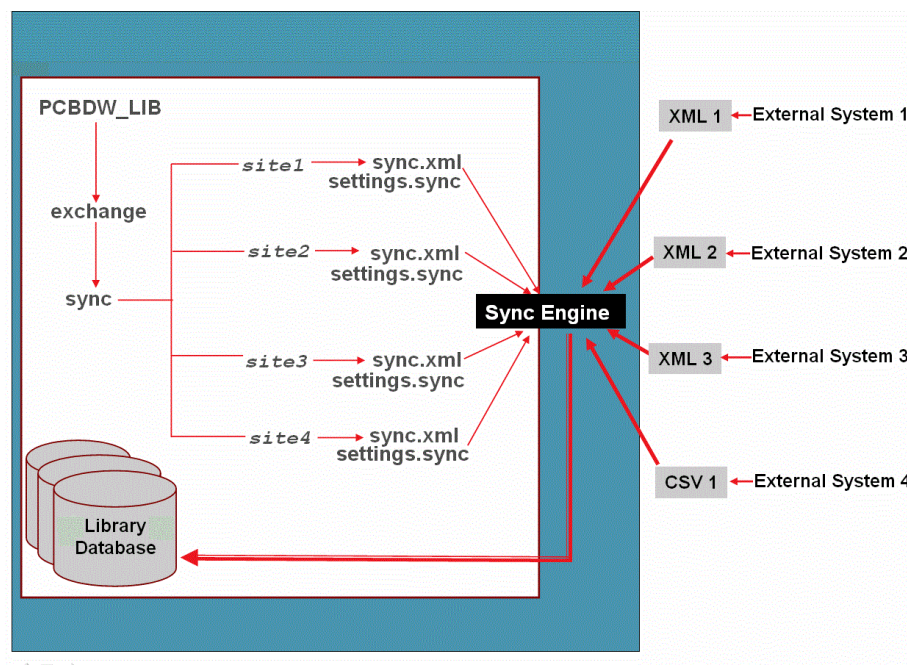
The data from the four files from the four different sources is imported into the Allegro EDM Component Database, and is synchronized with the external data sources.

- The data in each of the four sources is compared with the Allegro EDM Component Database.
- Policies are set to handle all scenarios, such as new, matching, and missing content, for each external source.

Continuing with the example, you would require four different policy sets. You can create one policy set, implement it then return the policy set for the next source and so on, but for the sake of clarity, four different policy sets are used for four different sources.

The following figure shows how the synchronization framework handles XML inputs from different sources:

Figure 1-1 Synchronization Framework Handling Many Sources



Here is a description of the steps in this workflow:

- Using the synchronization framework, you can import data from different sources and process each one differently.
- For every external source that you need to retrieve information from, you need to set up a synchronization system. Each system:

Allegro EDM Data Exchange Reference Guide

Working with Libraries External to Allegro EDM

- ❑ Can have different XML import policies and behavior.

- ❑ Should have the processing rules in a file, `sync.xml`.

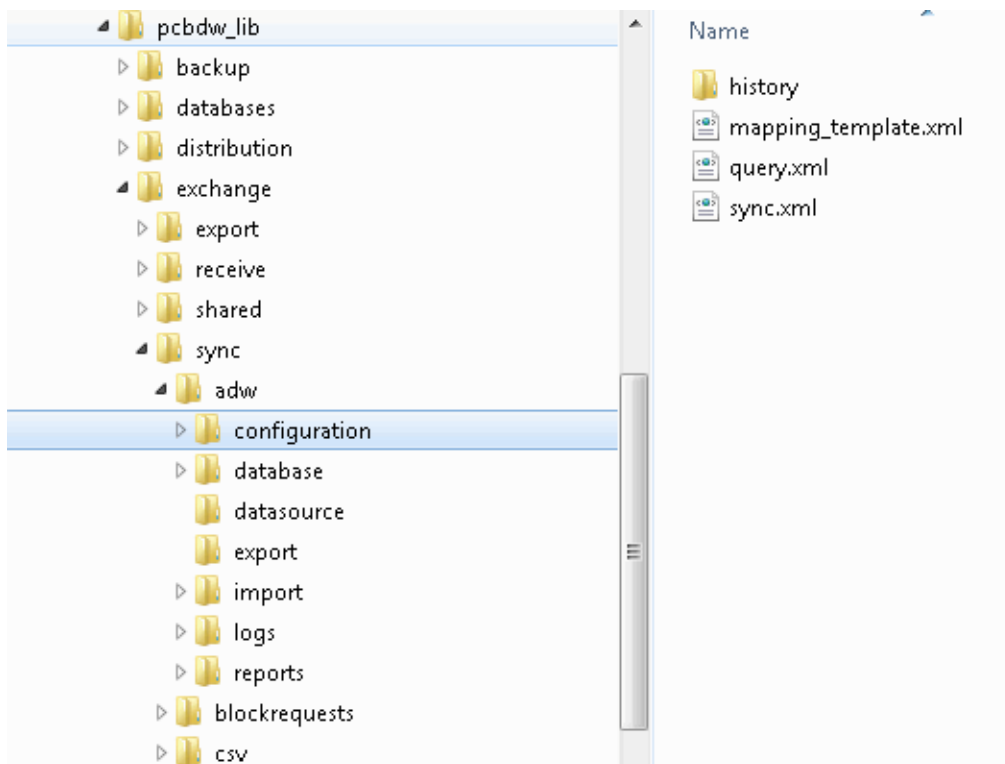
As it stores the import policies or the logic, the `sync.xml` file is also, at times, called the logic or logic definition file.

- ❑ Should have specifications on how the Allegro EDM synchronization system should behave, specified in the `settings.sync` file.

Based on the Synchronization Framework Handling Many Sources figure, the `sync` folder should have the following folders:

- ❑ `<site1>`
- ❑ `<site2>`
- ❑ `<site3>`
- ❑ `<site4>`

And each folder should have a `configuration` folder that holds a `sync.xml` file.



Allegro EDM Data Exchange Reference Guide

Working with Libraries External to Allegro EDM

The `sync`, by default, always has the following folders:

- `adw`
Used to import or merge Allegro EDM component databases
- `csv`
Used to import csv files

Synchronization Framework Components

As you have seen in the [Basic Synchronization Framework Flow](#) section, the information from external systems reaches the Allegro EDM Component Database after the synchronization engine processes the incoming XMLs based on policies in the respective data sources.

So far you have read about the XML files and `sync.xml`. Now, let us understand the main synchronization framework components in detail.

- [Input Data Sources](#)
- [Sync Configuration File](#)
- [Sync Engine](#)
- [Sync System Settings File](#)

Input Data Sources

There are several different types of input data sources. Here is a look at the commonly used ones:

- **Object XML**

Currently, this is the most common import format. As this is a Cadence proprietary XML exchange format, any system that is capable of generating output in an XML format can use the sync engine.

This is used to transfer data from one server to another and also to transfer an outside (customer or external) database into an Allegro EDM component database.

Note: Use the `adwapi` to create this XML.

- **External Systems**

External systems can use the sync engine as a live server. This means that an external system can update the Cadence database as soon as changes occur in their system.

This works as an online synchronous system. You can set either of the following ways to import data from an external system:

- **XML Diff Utility**

The XML Diff utility (in Database Editor) uses the sync engine to generate the comparison and merge results and also show the default merge results.

Using XML Diff, it is possible to modify the default merge sequence. For the most part, the sync engine eliminates the need of merging two objects manually, but there might be rare cases where a manual merge is the only option.

Sync Configuration File

The synchronization configuration file (`sync.xml`) captures all the XML import policies for a certain XML source. The policies cover all data-handling scenarios and the actions for each scenario.

This file and what information it has is described in [Chapter 3, “Defining Import Policies.”](#)

Sync Engine

The sync engine performs the following tasks:

1. Reads the input XML data source and the corresponding `sync.xml` file
2. Applies the configuration to the input data source
3. Tries to resolve conflicts, if any, by querying the server
4. Updates the target database with the resulting data

Sync System Settings File

The `settings.sync` file controls how the framework will behave for each of the synchronization systems. Each folder in `.../sync/` must have a `settings.sync` file.

At the end of a synchronization session, the input from the source is transferred to the target database.

Advantages of Synchronization Framework

As you can see, XML import policies are customizable for each source. Based on the policy for each site or XML source, the synchronization framework engine adds the imported XML data into the Allegro EDM component database.

Once configured, each sync system is self-sufficient and can synchronize systems. The only input to the sync engine will be the location of the corresponding `sync.xml` file.

Based on the sync system name, the sync engine will automatically pick the configuration options for that system and perform the synchronization.

Here is a list of some of the benefits that using a synchronization framework offers:

- Capable of expanding the current data import options

The standard synchronization framework helps capture the logic of what actions to perform for:

- ☐ External System Integration
- ☐ Library Server and Master Server Sync
- ☐ Library Import Upload

- Configurable to adapt the processing for each utility or site

- Reduces the amount of manual checks and coding

If the incoming data is in a specific format, the synchronization framework can be set up to handle most of the incoming conditions.

- Provides a site-independent logic core that acts as a starting point and is then customized for each site or customer

Bundled with the synchronization framework will be included different utilities to configure the functionality. As a start, this is based on the Upload tool, and has the capability to handle most known data import scenarios.

- Implement the sync engine in reverse

The Sync Engine is generic in nature. This implies that it can also be used by external systems to sync their databases with the data from an Allegro EDM Server.

To do this, a small set of APIs is available that enables the Sync Engine to read and update an external system. In this way, the same sync engine is running in a reverse (backwards) fashion, so instead of updating libraries, a database is updated based on Allegro EDM generated XML files.

■ Two-way sync

The sync engine and an external sync engine form the basis of a full two-way synchronization solution. This means that you can send changes from an external data source to an Allegro EDM system and also update the external data system with changes from the Allegro EDM system.

■ Sync book-keeping data

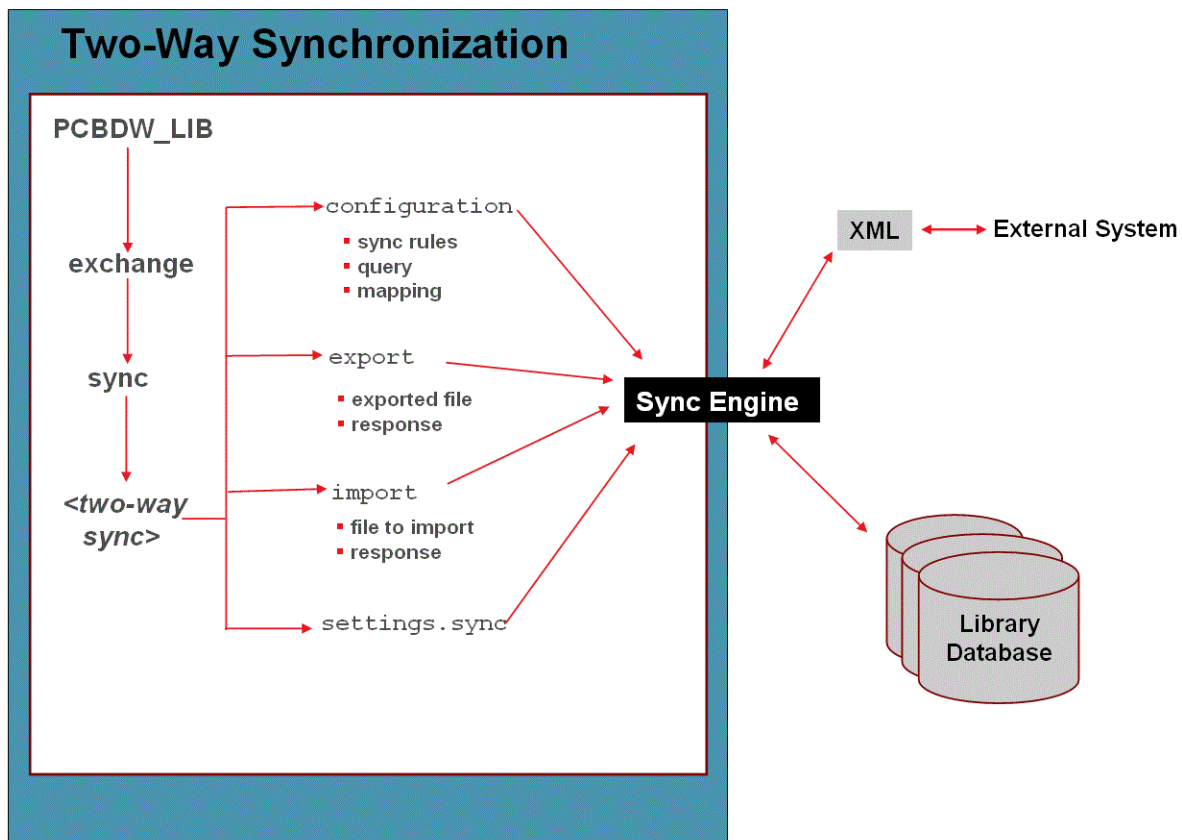
After each XML synchronization run, a record is maintained of the comparisons and mapping. This book-keeping data is used in future synchronization runs. Each time a synchronization run starts, it starts with a copy of the book-keeping data.



Ensure that the synchronization book-keeping data is not deleted.

Two-Way Synchronization between Allegro EDM and External Systems

The sync engine can also be used to keep an Allegro EDM implementation and an external system synchronized. The flow of the two-way synchronization is displayed in the following figure.



To implement the two-way synchronization, the following tasks are performed:

1. XML files are created for both the external system and Allegro EDM.
2. A sync system `<PCBDW_LIB>/exchange/sync/<two-way-sync>` is used.
3. Changes, if needed, are made to the configuration of the site.
4. The XML file to be imported into Allegro EDM is placed in the `import` folder.

5. The XML created based on the Allegro EDM component database that can be used by the external system is placed in the `export` folder.
6. Run the `dataexchange` command.
7. Verify the changes.

Using a Query File when Exporting Allegro EDM Data

To export Allegro EDM data, you need to create a query file. You can either use Database Editor to create one, or manually create one.

Creating a Query File from Database Editor

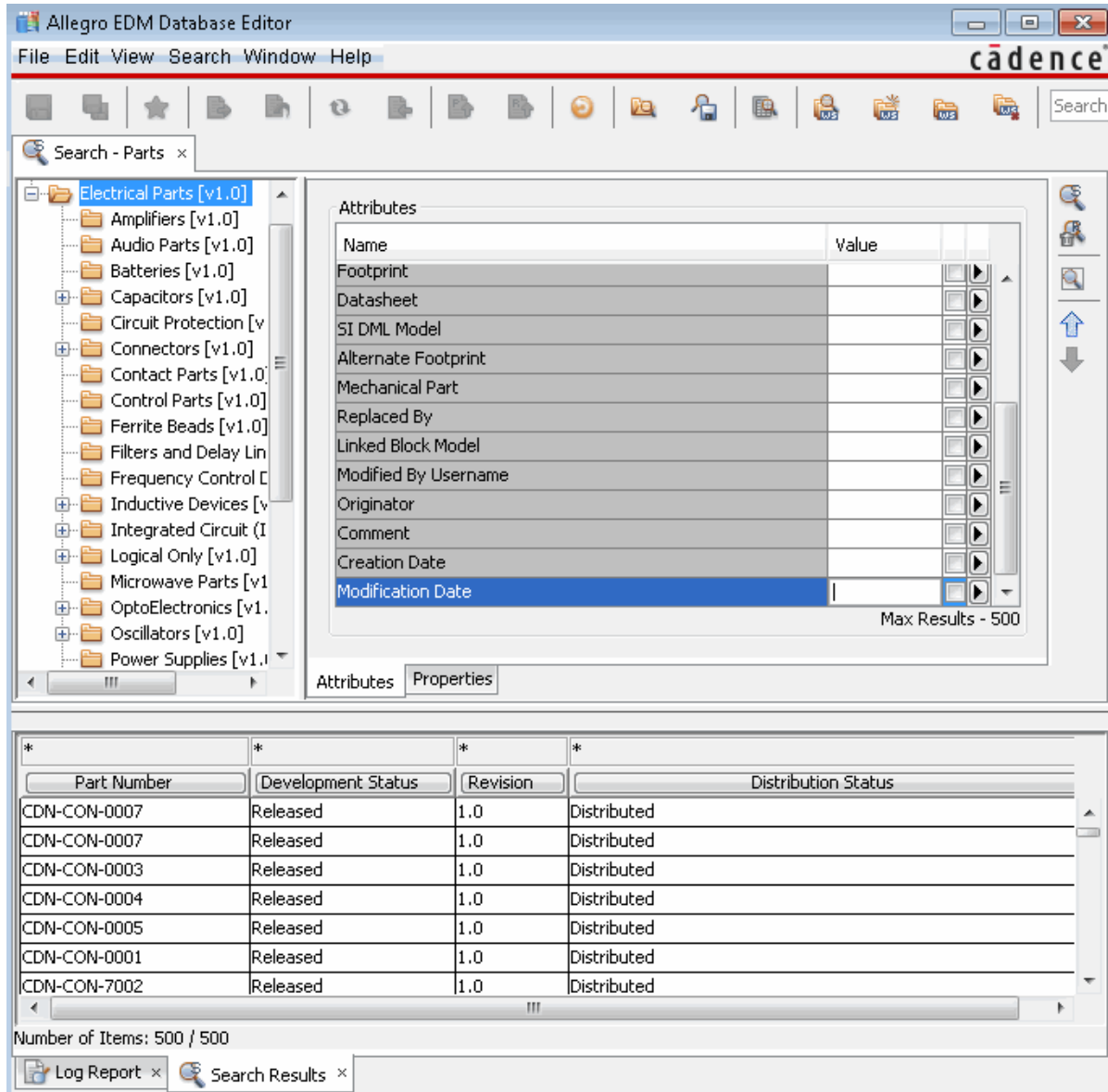
To define the parts of the Allegro EDM component database that need to be selected for the export, you should use the Database Editor save search criteria functionality to generate a query file.

1. Open Database Editor.

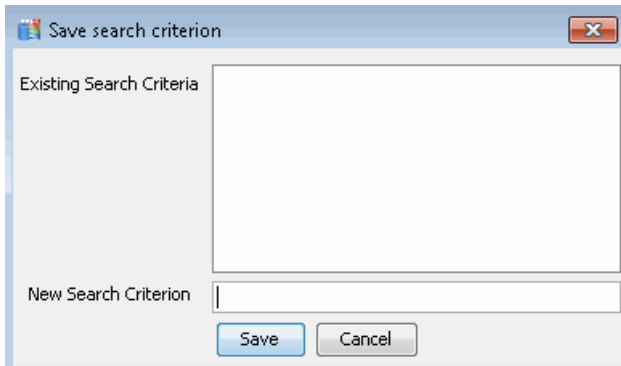
Allegro EDM Data Exchange Reference Guide

Working with Libraries External to Allegro EDM

2. Specify the search parameters.



3. Choose *Search – Save Search Criteria*.



4. Specify a name.

The file is saved in XML format in `<current_project_area>\atdmdir\search`.
For example, `D:\AllegroEDM_projects\lib_mast1\atdmdir\search`.

This file can be used as a query to export a subset of the entire Allegro EDM component database. Restricting the export candidates to only those database entries that match the conditions specified in the saved search.

To use this file to control which Allegro EDM entries are exported, do the following:

1. Rename this file to `query.xml`.

You **cannot** use any other name.

2. Place this file at the following location:

```
.../<pcbdw_lib>/exchange/sync/adw/configuration
```

3. Run the following command:

```
dataexchange -export <sync system name>
```

Manually Creating a Query File

Alternately, you could manually create a query file that lists the conditions that need to be matched. Here is a sample query file:

```
<Queries>
  <Query type="ECAD Component" limit="0" light="true" match="and">
    <Attributes selected="true">
      <Attribute name="name" value==" 6* || == 5*" />
    </Attributes>
  </Query>
</Queries>
```

Allegro EDM Data Exchange Reference Guide

Working with Libraries External to Allegro EDM

```
<Attribute name="Part Number" value="603275-109" />
<Attribute name="revision" value="last" />
<Attribute name="owner" value="*" />
<Attribute name="Comment" value="*" />
<Attribute name="current" value="*" />
</Attributes>
<Relations>
  <Relation name="Sync Tracker" fromtype="ECAD Component"
totype="WCSyncTracker" filter="negate" export="false">
    <Attribute name="Synced" value="true" />
  </Relation>
  <Relation name="Sync Tracker" fromtype="ECAD Component"
totype="WCSyncTracker" filter="none" export="false">
    <Attribute name="Synced" value="false" />
  </Relation>
</Relations>
<Interface type="ECAD Component Classification" name="*">
  <Attribute name="name" value="*" />
</Interface>
</Query>
```

You can include multiple queries in a file. Each query can search just one particular type.

Query Section

Query Parameters

- **Type** - the Schema type on which to perform a search
- **Limit** - Optional - 0 to 32000 - the limit on number of resultset. 0 is for all results satisfying the criteria.
- **Light** - Optional - true, false - this will fill just the necessary attributes in the resultset. These are mainly:
 - ☐ **type**
 - ☐ **name**

- ☐ revision
- ☐ key attributes
- Match - Optional - and, or - by default all criteria are 'and' operations. You can set this to 'or'.

Attributes Section

You can define more than one attribute for which to search. Each attribute must be specified as explained in the Attribute Section.

Parameters

- Selected - Optional - true, false - This will fill just the specified attributes in the resultset. Must be used for better search performance.

Attribute Section

Parameters

- Name - The name of the attribute based on which you want to perform a search. Refer to the Allegro EDM schema for actual attribute names for each type.
- Value - This can be a fixed value or an expression. The following operators are allowed: asterisk (*) and question mark (?) Allegro EDM also supports relational expressions such as '>' or '<'.
For example: "> 5 && < 10" "==" 6* || == 5*"

Relations Section

You can define more than one relation on which to perform a search. Each relation must be defined as explained in Relation Section.

Relation Section

Parameters

- Name - This is the name of the relation that is to be searched. Refer to the Allegro EDM schema for relation names for each type.
- Fromtype - This is the type which is on the from side of the relation.

Allegro EDM Data Exchange Reference Guide

Working with Libraries External to Allegro EDM

- Totype - This is the type which is on the to side of the relation.
- Filter - optional - (negate, none) - 'negate' will perform a NOT on the whole expression. 'None' will not filter on this relation and the criteria will be used for fetching the relation into the resultset. If user does not specify this attribute, a normal filter is applied on this relation.
- Export - optional - (true, false) - the relation will not exported to XML file if this is set to false.

Interface Section

Parameters

- Type - This is the type of the classification. Refer to the Allegro EDM schema for the types of classifications.
- Name - Currently, only asterisk (*) is supported. Searching for classifications by their names is not yet supported.

Using Synchronization Framework from the Command Line

You can use the data exchange command from the command line to use the synchronization framework.

Basic Command

The basic command is:

```
dataexchange -[import | export] <sync system name>
```

You should use this basic command. Provide all relevant information through the sync system.

Sample Commands

```
dataexchange -sync <sync system name>
```

```
dataexchange -import <sync system name>
```

```
dataexchange -export <sync system name>
```

```
dataexchange -response <sync system name>
```


Complete Syntax



This syntax is still supported, but Cadence recommends moving to the basic syntax to run the `dataexchange` command.

Additionally, there is a complete syntax:

```
dataexchange [-mode <type> [-workdir <path> [-ini <file>]] [- sync <system name>]]  
|  
[-<sync_action_name> <sync system name>]
```

where:

<type>	is export or import
-workdir <path>	<p>is optional. This defines the path for the working directory.</p> <p>By default, the current directory is picked as the working directory.</p> <p>In case a <code>workdir</code> is specified, the <code>ini</code> file is searched for in that location. To use another location, use the absolute path for the <code>ini</code> file.</p>
-ini <file>	is optional. Points to the configuration ini file.
- sync	<p>is used only in the <code>import</code> mode to override the entry in the <code>datasource</code> section of the XML file being imported.</p> <p>This is used in two ways. One with <code>-mode import</code> and the other as a full two-way synchronization.</p>

Allegro EDM Data Exchange Reference Guide

Working with Libraries External to Allegro EDM

- <sync_action_name> Is used to synchronize the Allegro EDM system with an external system.
<sync system name>]

The sync_action_name can be: -sync | -import | export | response

■ sync

Does a full two-way synchronization between Allegro EDM and the external system. This command includes all the three commands: import, export, and response.

■ import

Reads in the XML file stored in the import folder to the Allegro EDM system. Only performs the import from the external system.

■ export

Creates an XML file, based on the Allegro EDM system, and stores it in the export folder. The parts which need to be exported can be specified in the query file. Only performs the export from Allegro EDM to the external system.

■ response

Reads the response files created by the last iteration.

Sample Commands

```
dataexchange -mode import -workdir <path> -sync <sync system name> -sync <sync system name>
```

```
dataexchange -mode export -workdir <path where file will be exported>
```

Managing Synchronization Sessions

In addition to rules and conditions for the incoming XML data, you can also control other aspects of the synchronization framework.

These aspects include:

- Controlling the addition of the XML information to the Allegro EDM libraries database

Consider a situation where you want to test the synchronization rules but not update the database. Or, you need to demonstrate the synchronization framework and show all of its functionality but not tamper with the database.

- Verifying the `sync.xml`

The synchronization framework can verify the `sync.xml` file to ensure its correctness and completeness before the XML import is started. You should enable this verification in the initial import sessions and later on, after the `sync.xml` has parsed information and you know it is working as required, you can disable this verification.

- Updating and creating working sets

Working sets are extensively used when working with the Database Editor. The XML synchronization framework offers you the ability to automatically update the working sets information with the newly added library elements. You can:

- ☐ Update the current working set
- ☐ Create a new working set with the XML objects
- ☐ Control the handling of the current working set
- ☐ Specify the naming of the newly created working set

The name of the working set contains:

- ☐ A text prefix
- ☐ The date in a specific format

These settings are specified in the `settings.sync` file.

The settings.sync File

A settings.sync file is located in each sync system folder available at:

```
.../PCBDW_LIB/exchange/sync/<sync_system_name>
```

For example, for the sync system adw, the settings.sync file is at:

```
.../PCBDW_LIB/exchange/sync/adw
```

The settings.sync file usually contains the following information:

```
<settings>

  <parameter name="test.mock_run" value="false" />
  <parameter name="config.validation.enabled" value="true" />
  <parameter name="wset.enabled" value="true" />
  <parameter name="wset.auto_create.enabled" value="false" />
  <parameter name="wset.auto_create.name_prefix" value="sync_adw_" />
  <parameter name="wset.auto_create.name_date" value="yyyy-MM-dd" />
  <parameter name="reports.description.name"
value="DESCRIPTION,COMMENT,description" />
  <parameter name="report.enable" value="true" />
  <parameter name="report.fields" value="Type, TypeOfChange, Name,
Version, LifeCycleStatus" />

  <parameter name="obsolete.enable" value="false" />
  <parameter name="part.mapping.disable" value="false" />
  <parameter name="part.mapping.apply.all" value="false" />
  <parameter name="import.sync.properties" value="syncproperty1,
syncproperty2, syncproperty3, syncproperty4" />

  <parameter name="sync.advanced.enable" value="true" />
  <parameter name="response.type" value="WCSyncTracker" />
  <parameter name="resolve.part.name" value="false" />
  <parameter name="resolve.model.name" value="false" />

  <parameter name="mapping.enable" value="true" />

</settings>
```

Allegro EDM Data Exchange Reference Guide

Managing Synchronization Sessions

The following table explains the entries in the `settings.sync` file. Some setups might use customized entries.

Table 2-1 Information in the setting.sync file

Entry	Explanation
<code>test.mock_run</code>	<p>Specifies whether the XML data should be added to the component database. The possible values for this setting are:</p> <ul style="list-style-type: none">■ True Does not upload the XML file contents to the database.■ False Uploads the XML file contents to the database. The default value is <code>false</code>.
<code>config.validation.enabled</code>	<p>Verifies if the <code>sync.xml</code> file is syntactically valid and all entries are complete and correct. Set to <code>TRUE</code> by default and should not be changed.</p> <p>If you set this to <code>FALSE</code>, the data exchange operation might have potentially incorrect results.</p>
<code>wset.enabled</code>	<p>Specifies whether the newly added content from the XML should be added to the current working sets. The possible values for this setting are:</p> <ul style="list-style-type: none">■ True Objects are added to the working set.■ False Objects are not added to the working set. The default value is <code>false</code>.
<code>wset.auto_create.enabled</code>	<p>Comes into effect if the <code>wset.enabled</code> entry is set to <code>True</code>. This entry controls whether the Data Exchange operation can create a new working set.</p>

Allegro EDM Data Exchange Reference Guide

Managing Synchronization Sessions

Table 2-1 Information in the setting.sync file

Entry	Explanation
<code>wset.auto_create.name_prefix</code>	<p>Comes into effect if the previous two entries (<code>wset.enabled</code> and <code>wset.auto_create.enabled</code>) are set to true.</p> <p>This variable specifies what prefix to use for the working set created by the Data Exchange operation.</p>
<code>wset.auto_create.name_date</code>	<p>Specifies the date format which is to be used for naming the newly created working sets. This is in continuation with the working set creation. You can use any of the common Java date formats.</p>
<code>reports.description.name</code>	<p>Used internally by data exchange. Do not delete or update this entry. This specifies the various possible field names for that describe the part or model.</p>
<code>report.enable</code>	<p>Specifies whether a report of the import operation is to be created. The report is created with a name: <code>dxreport.csv</code></p>
<code>report.fields</code>	<p>Specifies which fields to be included in the <code>dxreport.csv</code> report. The report lists all the added, updated, or changed entries in the Allegro EDM component database.</p>
<code>obsolete.enable</code>	<p>Not used anymore.</p>
<code>part.mapping.disable</code>	<p>Set to <code>False</code> by default. This enables Allegro EDM to handle multiple instances of identical part numbers. Each instance is treated as a unique one and the instances are not mixed up.</p> <p>If set to <code>True</code>, you can set Allegro EDM to handle identical part numbers by using one of the following settings:</p> <ul style="list-style-type: none">■ <code>import.sync.properties</code>■ <code>part.mapping.apply.all</code> (not recommended)

Allegro EDM Data Exchange Reference Guide

Managing Synchronization Sessions



Table 2-1 Information in the setting.sync file

Entry	Explanation
<code>import.sync.properties</code>	<p>This entry comes into effect if <code>part.mapping.disable</code> is set to <code>True</code>.</p> <p>This setting specifies which part properties, known as sync properties, are to be included with the Part Number attribute to form a combination that resolves the multiple instances of identical part numbers in the database.</p> <p>There can be two cases when multiple instances of identical part numbers are not resolved.</p> <ul style="list-style-type: none">■ When the combination of Part Number and sync properties identifies more than one instance of the identical part, an error is displayed. The error shows the duplicate part numbers that were ignored during the data exchange process. <p>In such a case, ensure that you specify the sync properties that can uniquely identify each instance of the identical part.</p> <ul style="list-style-type: none">■ When the combination of Part Number and sync properties does not identify any part, the incoming part is treated as a new part and the rules, as defined in <code>sync.xml</code>, are applied and executed. <p>To use data exchange for updating multiple instances of identical part numbers, see <i>Allegro EDM Frequently Asked Questions</i>.</p>

Allegro EDM Data Exchange Reference Guide

Managing Synchronization Sessions

Table 2-1 Information in the `setting.sync` file

Entry	Explanation
<code>part.mapping.apply.all</code>	<div> Caution <i>Using this setting to handle identical part numbers is not recommended. Instead, use the <code>import.sync.properties</code> setting to specify the sync properties to be used for resolving identical part numbers.</i></div> <div> Important If you want to use this setting, ensure that you remove the <code>import.sync.properties</code> setting from <code>settings.sync</code>.</div> <p>This entry comes into effect if <code>part.mapping.disable</code> is set to <code>True</code>. If set to:</p> <ul style="list-style-type: none">■ <code>True</code>: Applies all changes based on the incoming part number to all matching parts.■ <code>False</code>: Does not apply changes and displays an error when a duplicate part number is found. <p>Note: If there is a 1:1 situation, that is, no duplicates are there in the incoming data, either of the settings will have no impact. In such a case, the incoming part will be processed based on the rules defined in <code>sync.xml</code>.</p>
<code>sync.advanced.enable</code>	<p>Existing Allegro EDM implementations can continue using Data Exchange the way they currently use or use two-way synchronization. The possible values for this setting are:</p> <ul style="list-style-type: none">■ <code>True</code> Implement the two-way synchronization system.■ <code>False</code> Disable the two-way synchronization system. <p>The default value is <code>true</code>.</p>

Allegro EDM Data Exchange Reference Guide

Managing Synchronization Sessions

Table 2-1 Information in the `setting.sync` file

Entry	Explanation
<code>response.type</code>	<p>Specifies whether the Data Exchange system needs to perform a two-way synchronization between two systems. By default, this is set to <code>WCSyncTracker</code>.</p> <p>Note: Do not change this entry if you are using the data exchange synchronization system.</p>
<code>resolve.part.name</code> <code>resolve.model.name</code>	<p>Disables name mangling for parts and models, respectively. These settings come into effect when exporting data from the Allegro EDM component database to an external target. The names of parts and models as shown in the GUI are different from the way they are stored internally.</p> <p>If the resolve entries are set to:</p> <ul style="list-style-type: none">■ <code>True</code>: The part and model names as used in the GUI are displayed/maintained in the export.■ <code>False</code>: The internal name that is used with the Allegro EDM component database is used.
<code>mapping.enable</code>	<p>Specifies whether the mapping information (<code><pcbdw_lib>/exchange/sync/<sync_system_name>/configuration/mapping.xml</code>) should be read by the Data Exchange system.</p> <p>If the mapping file is missing, this entry is skipped.</p> <p>The possible values for this setting are:</p> <ul style="list-style-type: none">■ <code>True</code> Reads the mapping information when data is imported or exported. The default value is <code>true</code>.■ <code>False</code> The mapping information is ignored by the data exchange system.

Allegro EDM Data Exchange Reference Guide

Managing Synchronization Sessions

Table 2-1 Information in the `setting.sync` file

Entry	Explanation
<code>name.mapping</code>	Specifies whether the part/object name should be mapped to the master ID.
<code>export.archive</code>	Includes tar balls of models and packages.
<code>export.package</code>	Creates a zip file containing the tar ball and XML file exported.
<code>delete.relation.enable</code>	<p>Removes parts that are no longer found in the incoming data.</p> <p>If the incoming data does not have an entry, the matching existing Allegro EDM entry can be removed.</p> <p>This entry needs to be manually added to <code>settings.sync</code> in case the following entry is set to <code>override</code> in the <code>sync.xml</code> file for the site.</p> <pre><Condition name="SourceNotExist" action="ignore override"/></pre>

By modifying the settings in the `setting.sync` file, you can control the behavior of the subsequent XML sessions.

Defining Import Policies

The `sync.xml` file contains the logic and import policies for a specific XML source. This chapter explains the conditions that can arise and how to specify what action to take for each.

Structure of the `sync.xml` file

The synchronization definition file (`sync.xml`) file is divided into the following parts:

- [Rules Section](#)
- [Types Section](#)
- [Interface Section](#)

Rules Section

This section stores the rules that describe conditions and the actions to be executed when each of the conditions is met. The rules can apply to any of the following:

- Objects
- Relations
- Attribute
- Lifecycle
- Interface

The Lifecycle rule describes the Development Status of each object after the object rule has been applied.

Each type of rule will be discussed thoroughly in subsequent sections.

Tags

■ Condition tags

Sometimes the action on the relation and attribute depends on the action on the object. So, when an object is modified through a certain action, specific actions need to be performed on the attribute and relation. This is done with special Condition tags.

■ Trigger tags

There are also cases when the action on the object depends on the action taken on the attributes and relations. This is done with trigger tags. You can write an attribute and relation trigger. This trigger maps the action on the attribute or relation to the action on the object. Triggers are explained in detail in the [Trigger Rules](#) section.

■ Special condition tags

Sometimes the action on the Relation and Attribute depends on the action on the Object. When an object is modified by an action, you might need to perform different actions on the attributes and relations. This is done with special condition tags.

These conditions are actions taken on Objects through object rules.

Types Section

This section binds different rules to actual object types, relations and attributes. You can create as many rules as required and can bind different rules with different object types.

The name `ALL` is used in the XML to specify that all object types in the system follow this rule. You can also use specific types along with the `ALL` section.

Note: The specific type overrides the rule described in the `ALL` section.

Interface Section

This section binds different interface rules to the actual interface types. You may want to handle each interface type separately. In such a scenario, you need to create many interface sections, one for each interface type and bind different rules to each type.

Just as for the Types section, the `ALL` type covers all the interface types.

Object Rules

- Every object in the source XML file goes through an analysis phase.

During this phase, synchronization rules are applied to the object.

- As soon as a condition is met for an object, the action related to that condition is applied to the object.
- As a result of applying the rules, an action is determined for that particular object.
- During the sync process, the identified action is performed on the object.

You can create as many object rules as required and assign them unique IDs.

These object rules can later be bound with different type of objects. You can also specify different sync rules for different objects.

Condition - TargetNotExist

This condition holds true when the object from the source does not exist in the target database. This results in an ignore or create action.

- Ignore
Do not do anything. Ignore the source object and proceed with the next object in sequence.
- Create
Insert a new object into the target database with the data of the source object.

Condition - TargetExist

This condition holds true when the object from the source XML exists in the target database.

- This condition only checks for the target object with the same name as that in the source and does not look for the actual revisions.
- The target object may be of a different revision. This condition has subconditions to handle revision-related cases.
A sub-condition is tested only if the parent condition is met.
- All subconditions inherit the actions from the parent condition. The actions from the subconditions override the action from the parent condition.
- If you omit some subconditions, the action for their parents are used.
 - Ignore

Do not do anything. Ignore the source object and proceed with the next object in sequence.

☐ Update

Take the latest version of the object from the database and update the same revision with the data from the source object. After this operation, no new object is created in the database.

☐ Defer

This defers the action on the object. All deferred actions are triggered later through relations and attribute rules.

This is needed when the action on the object depends upon the attributes being modified on the object. For example:

You may want to create a major revision only when certain attributes are modified. For the rest of the attributes, you want to create only a minor revision.

This is also useful when the action also depends on the relations being created for that object.

The actual action on the object is derived from the attributes and relations rules. The details are explained in the following sections:

- ☐ [Attribute Rules](#)
- ☐ [Relation Rules](#)

Condition - TargetBeingModified

This condition is a sub-condition of the TargetExist condition. As a result, this condition is tested only if the TargetExist condition is satisfied, which means that the object from the source exists in the target database. This condition holds true when object in the target database is being modified.

This happens when the object is in the `Checked-out` or `Preliminary` state.

You can ignore the object or choose to update the existing object.

Note: You cannot specify a `Major` or `Minor` action for an object that is in the `Checked-out` or `Preliminary` state.

☒ Ignore

Do not do anything. Ignore the source object and proceed with the next object in sequence.

- **Update**

Take the latest version of the object from the database and update the same revision with the data from the source object. After this operation, no new object is created in the database.

Condition - TargetRevisionLower

The revision of the source object is compared with the revision of the target object. This condition is satisfied if the revision of the target is lower than the revision of the source object. The actions that can be performed on this condition are similar to the actions that can be performed on its parent, the TargetExist condition.

The list of actions include:

- **Ignore**

- **Update**

See the [Condition - TargetExist](#) section for a description.

- **Major**

Take the latest version of the object from the database and make a Major change in the object. After creating the modified object, update the newly created revision with the data from the source object.

- **Minor**

Take the latest version of the object from the database and do a Minor revision of the object. After creating the revision, update the newly created revision with the data from the source object.

- **Retain**

Take the latest version of the object from the database and apply the revision specified in the source Object onto the object in the database. After creating the revision, update the newly created revision with the data from the source object.

- **Defer**

Condition - TargetRevisionHigher

The revision of the source object is compared with the revision of the target object. This condition is satisfied if the revision of the target is higher than the revision of the source object. The actions that can be performed on this condition are similar to the actions that can be performed on its parent condition, the TargetExist condition. The list of actions include:

- Ignore
- Update
- Major
- Minor
- Defer

Condition - TargetRevisionEqual

The revision of the source object is compared with the revision of the target object. This condition is satisfied if the revision of the target is equal to the revision of the source object. The actions that can be performed on this condition are similar to the actions that can be performed on its parent condition, the TargetExist condition.

The list of actions include:

- Ignore
- Update
- Major
- Minor
- Defer

Relation Rules

Just like objects, each relation in an input XML goes through a set of relation rules. During the analysis phase, each relation is tested against a set of conditions. When a condition is met for that relation, the action related to that condition is applied to the relation.

Later in the sync process, the resolved action is performed on the relation.

The first level of conditions on relations depends on the action performed on the object.

You can choose to perform different sets of actions on relations based on different actions for an object. The first-level conditions help achieve that, such as when you want to apply one set of synchronization policies when an object is updated and another set when the object is revised.

Condition - ObjectUpdated

This condition holds true when the object for which the relation is being tested has been marked for updating. This condition has subconditions, which are actually conditions for the relations. The following subconditions are tested only if the parent condition holds true.

Condition - TargetTypeNotExist

The `From` object of the source relation is looked for all relations of the given type in the target database. This condition is set to have been met when the target database does not have the relation of the same type on the object.

The relation is between the same `From` and `To` types. You can choose to:

- **Ignore**
Do nothing. Ignore the source relation and proceed with the next relation in sequence.
- **Create**
Create a new relation in the target database. A relation is created between the two objects in the database.
The `From` and `To` objects are derived from the source relation.

Condition - TargetTypeExist

The `From` object of the source relation is looked for the given relation type in the target database. This condition is set to be true when the `From` object has relations of the given relation type in the target database. The relation can be ignored, created, updated, or overridden.

If no subconditions are specified for this condition, the action listed in this condition is applied. The actions on subconditions override the actions on the parent condition.

- **Ignore**
Do not do anything. Ignore the source relation and proceed with the next relation in sequence.

- **Override**

Take the target relation from the database and delete the relation from the target database.

And create the source relation in the target database.

Condition - TargetNameNotExist

This condition is satisfied when the `From` objects are linked with different `To` objects in the database but with the same type of relationship.

The actions that can be applied when this condition is satisfied are:

- **Ignore**

- **Create**

Take the source relation and create the relation into the target database. After this action a relation is created between two objects in the target database.

- **Override**

The explanations of the other actions are in the [Condition - TargetExist](#) section.

Condition - TargetNameExist

This condition is satisfied when the relation in the database has the same `From` and `To` objects. The revision of objects can be different in the target database as compared to the source relation. This condition also has subconditions that distinguish between the same object names and the same and different revisions. The list of actions includes:

- **Ignore**

- **Update**

Take the target relation from the target database and update the attributes on the target relation with the attributes from the source relation.

- **Override**

The explanations of the other actions are in the [Condition - TargetExist](#) section.

Condition - TargetRevisionNotExist

This condition is set to be true when the target database has a relation between objects but the version of the object is different in the source.

The list of actions include:

- Ignore
- Update
- Create
- Override

Condition - TargetRevisionExist

This condition is satisfied when the target database has the relation between the same objects and even the version of the objects is the same as that of the source relation.

The list of actions is:

- Ignore
- Update
- Override

Condition - ObjectDeferred

This condition is set to be satisfied when the object for which the relation is being tested has been deferred. This condition has the same subconditions as the `ObjectUpdated` condition (see [Condition - ObjectUpdated](#)). All the subconditions can also be tested for this condition.

Condition - ObjectCreated

This condition is satisfied when the object for which the relation is being tested has been marked for Creation. This condition has the same subconditions as the `ObjectUpdated` condition (see [Condition - ObjectUpdated](#)). All the subconditions can also be tested for this condition.

Condition - ObjectMajor

This condition is satisfied when the object for which the relation is being tested has been marked for Creating a Major Revision. This condition has the same subconditions as of the ObjectUpdated condition (see [Condition - ObjectUpdated](#)). All the subconditions can also be tested for this condition.

Condition - ObjectMinor

This condition holds true when the object for which the relation is being tested has been marked for Creating a Minor Revision. This condition has the same subconditions as of the ObjectUpdated condition (see [Condition - ObjectUpdated](#)). All the subconditions can also be tested for this condition.

Condition - ObjectRetained

This condition holds true when the object for which the relation is being tested has been marked for retaining the source revision. This condition has the same subconditions as the ObjectUpdated (see [Condition - ObjectUpdated](#)) condition. All the subconditions can also be tested for this condition.

Condition - ObjectIgnored

This condition holds true when the object for which the relation is being tested has been marked as Ignored. This condition has the same subconditions as of the ObjectUpdated condition (see [Condition - ObjectUpdated](#)). All the subconditions can also be tested for this condition.

Condition - Default

This condition is the default condition. When no specific condition is specified, all conditions match this condition. Compare this to the switch-case statement of most object-oriented languages. You can write conditions for which specific actions are needed and the rest default to the Default condition. Always use the default condition to reduce errors.

Attribute Rules

Attribute rules define synchronizing rules for the attributes of an object. Each attribute can have a different set of attribute rules. You can create many attribute rules and assign them

unique IDs. In the Sync Type section, you can use these IDs to specify the rules that apply to an attribute. The [Sync Type](#) section explains how to set up the rules.

Condition - ObjectUpdated

This condition holds true when the object for which the attribute is being tested has been marked for updating. For the attribute, you can choose to either:

- Ignore
- Update

Condition - ObjectDeferred

This condition holds true when the object for which the attribute is being tested has been deferred. You can either Ignore or Update the attribute.

Condition - ObjectCreated

This condition holds true when the object for which the attribute is being tested has been marked for Creation. You can either Ignore or Update the attribute.

Condition - ObjectMajor

This condition holds true when the object for which the attribute is being tested has been marked for creating a major revision. You can either Ignore or Update the attribute.

Condition - ObjectMinor

This condition holds true when the object for which the attribute is being tested has been marked for creating a minor revision. You can either Ignore or Update the attribute.

Condition - ObjectRetained

This condition holds true when the object for which the attribute is being tested has been marked for retaining the source revision. You can either Ignore or Update the attribute.

Condition - ObjectIgnored

This condition holds true when the object for which the attribute is being tested has been marked as Ignored. You can either Ignore or Update the attribute.

Condition - Default

This condition is a default condition. When no specific condition is specified, all conditions match this condition. Compare this to the switch-case statement of most object-oriented languages. You can write conditions for which specific actions are needed and the rest default to the Default condition. Use a Default condition to reduce errors.

Lifecycle Rules

Lifecycle Rules are used to define the Development Status of an object. Each object goes through the lifecycle rules and the appropriate action is determined for each object. After upload, the specified Development Status is applied to the newly created object.

Note: The *Development Status* attribute was called the *Lifecycle State* until Allegro EDM version 16.2.

Lifecycle rules have the same conditions as those specified in [Relation Rules](#) and [Attribute Rules](#). The actions that can be used for each condition are Ignore, Checkin, Delete, or Release.

Conditions for Lifecycle Rules

The conditions defined for lifecycle rules are:

- Default
- ObjectCreated
- ObjectUpdated
- ObjectMajor
- ObjectMinor
- ObjectRetained

Lifecycle rules are helpful when you need to perform various development status-related operations on the basis of actions defined on the object. For example, when an object is created, you may choose to release it, and when an object is updated, you may choose to ignore it.

For each of these conditions, you can also have a subcondition that needs to be tested for the source object.

Subconditions for Lifecycle Rules

You can extend a condition with additional subconditions. Based on the value of the `current` attribute on the source object, the subcondition to be applied is selected, and the corresponding action is executed. The `current` attribute corresponds to the Development Status of the object.

Allegro EDM Data Exchange Reference Guide

Defining Import Policies

The actions to be executed for subconditions are: Ignore, Checkin, Delete, or Release

Subcondition - SourcePreliminary

This condition holds true when the value of the `current` attribute on the source object is Preliminary.

Subcondition - SourceCheckedOut

This condition holds true when the value of the `current` attribute on the source object is Checkout.

Subcondition - SourceCheckedIn

This condition holds true when the value of the `current` attribute on the source object is Checked-In & Verified.

Subcondition - SourceReleased

This condition holds true when the value of the `current` attribute on the source object is Released.

Subcondition - SourceDeleted

This condition holds true when either the value of the `current` attribute on the source object is Deleted or the value of the `Distribution Status` attribute is Pending Delete.

When you set the value of the `current` attribute on the source object as Deleted in the source file, and then import data into the Allegro EDM Component Database, the Distribution Status changes to Pending Delete. After you run library distribution, the Development Status is changed to Deleted.

Note: Importing data using the CSV format supports Delete only for electrical parts.

Example

The subcondition for the `ObjectMajor` condition can be specified as follows:

```
<LifecycleRules id="lifecycle_rules">
  <Condition name="ObjectMajor" action="ignore">
```



```
<Condition name="SourceCheckedIn" action="checkin"/>

</Condition>
```

```
</LifecycleRules>
```

This condition indicates that if the target object is checked out with a major revision, then check it in only if the value of the `current` attribute on the source object is `Checked-In & Verified`. Else, the `ignore` action is executed.

Note: If you have defined the delete action for a particular lifecycle rule, the rules that are run when you delete an object in Database Editor are also run when importing data from an external source into the Allegro EDM Component Database. This ensures that only the required objects are deleted.

Trigger Rules

Trigger rules are specific rules that are applied when the appropriate action was not determined for a particular object during the Object Rules. This is the case when an object is marked as `Deferred` during the Object Rules.

Sometimes the action on an object depends upon the action on the Attribute or Relation. As an example, you may only want to create a revision if certain attributes are being modified and for the rest of the attributes, you only want to update the existing revision.

There are two types of Trigger rules:

- One generates the trigger from the Attributes
- Another generates the trigger from the Relations

Attribute Trigger

Since there are only two actions that can happen on attributes, there are only two conditions for which you can write a trigger for an object.

- **AttributeUpdated**

This condition is satisfied when an attribute is marked for updating through the Attribute Rules. You may choose to perform different actions on an object. The actions list is the same as that described in the Object Rules section. The list of actions includes:

- ☐ Ignore
- ☐ Update

- ☐ Major
- ☐ Minor
- ☐ Retain

■ **AttributeIgnored**

This condition is satisfied when an attribute is ignored through the Attribute Rules. You may choose to perform different actions on an object. The actions list is the same as that in the Object Rules section. The list of actions includes:

- ☐ Ignore
- ☐ Update
- ☐ Major
- ☐ Minor
- ☐ Retain

Relation Trigger

You can use three different conditions for the Relation trigger, and on each condition, apply different object actions.

■ **RelationUpdated**

This condition is satisfied when a relation is marked for updating through the Relation Rules. User may choose to perform different actions on object. The actions list is the same as the Object Rules section. The list of actions include:

- ☐ Ignore
- ☐ Update
- ☐ Major
- ☐ Minor
- ☐ Retain

■ **RelationCreated**

This condition is satisfied when a relation is created through the relation rules. You may choose to perform different actions on objects. The actions list is the same as that described in the Object Rules section. The list of actions include:

- ☐ Ignore
- ☐ Update
- ☐ Major
- ☐ Minor
- ☐ Retain

■ **RelationIgnored**

This condition is satisfied when a relation is ignored through the Relation Rules. You may choose to perform different actions on object. The actions list is the same as that described in the Object Rules section. The list of actions include:

- ☐ Ignore
- ☐ Update
- ☐ Major
- ☐ Minor
- ☐ Retain

Sync Type

The Sync Type section binds the rules to actual object types. You can bind different rules to different object types. For example, you can create different rules for parts and Schematic Models.

A special name, `ALL`, is provided, which binds the rules to all object types in the database. You can create Sync Types specific to a particular type and bind separate rules. For that type the rules will be overridden with the specific rules.

Sync type supports two types of rules:

- **Object Rule**
- **Lifecycle Rule**

You provide the ID of the object rules described in Object Rules section. Once you specify the ID, the rules corresponding to that ID are applied to this particular sync type.

You can also bind lifecycle rules in the Sync Type section, and the rules specific to that lifecycle rule are applied to the type.

Sync Type has the Sync Attributes and Sync Relations sections, which are for binding Attributes to attributes rules and for binding relations to relation rules. You can specify attribute rules and relations rules in these two sections to bind specific rules to specific entities.

Sync Attributes

You can bind Attribute rules and Trigger rules in this section. All attributes for that particular type of object will be applied with those rules and triggers. You may also want to specify different rules for some attributes. For example, you may want to create a minor revision only if the `TOLERANCE` attribute is modified. Now you bind a different rule to the `TOLERANCE` attribute.

You can set specific rules for some attributes in the SyncAttribute section. These specific rules override the rules specified in the SyncAttributes section. For example, you may want to leave some attributes for update. In this case, create Ignore attribute rules and bind attribute to ignore the attribute rule.

Sync Relations

This section is for binding relation rules to specific relations. You may also bind the same relation rule to all relations. This is done by providing the `rule` attribute in the Sync Relations section.

You can also specify rules for specific relations. This can be done using the SyncRelation tag. In the SyncRelation tag, you can specify the relation name and the rule ID to be used for this relation.

Syntax of sync.xml

```
<AdwSync version="1.0">
  <SyncRules>
    <ObjectRules id="object_id">
      <Condition name="TargetNotExist" action="create" />
      <Condition name="TargetExist" action="defer">
        <Condition name="TargetBeingModified" action="update">
          </Condition>
        </Condition>
      </Condition>
    </ObjectRules>
  </SyncRules>
</AdwSync>
```

Allegro EDM Data Exchange Reference Guide

Defining Import Policies

```
</ObjectRules>

<RelationRules id="relation_id">
  <Condition name="ObjectIgnored" action="ignore"/>
  <Condition name="Default" action="create">
    <Condition name="TargetTypeNotExist" action="create"/>
    <Condition name="TargetTypeExist" action="override">
      <Condition name="TargetNameNotExist"
action="create"/>
      <Condition name="TargetNameExist"
action="override">
        <Condition name="TargetRevisionNotExist"
action="create"/>
        <Condition name="TargetRevisionExist"
action="update"/>
      </Condition>
    </Condition>
  </Condition>
</RelationRules>

<LifecycleRules id="lifecycle_id">
  <Condition name="Default" action="release" />
  <Condition name="ObjectIgnored" action="ignore" />
  <Condition name="ObjectUpdated" action="ignore" />
</LifecycleRules>

<AttributeRules id="attribute_id">
  <Condition name="ObjectIgnored" action="ignore"/>
  <Condition name="Default" action="update"/>
</AttributeRules>

<TriggerRules id="attr_trigger_id_major">
  <Condition name="Default" action="ignore" />
  <Condition name="AttributeUpdated" action="major" />
</TriggerRules>
```

Allegro EDM Data Exchange Reference Guide

Defining Import Policies

```
</TriggerRules>

<TriggerRules id="attr_trigger_id_minor">
    <Condition name="Default" action="ignore" />
    <Condition name="AttributeUpdated" action="minor" />
</TriggerRules>

<TriggerRules id="relation_triggers">
    <Condition name="Default" action="ignore" />
<Condition name="RelationCreated" action="minor" />
</TriggerRules>

<InterfaceRules id="interface_id">
    <Condition name="TargetNotExist" action="create" />
    <Condition name="TargetExist" action="ignore" />
</InterfaceRules>

</SyncRules>

<SyncType name="ALL" rule="object_id" lifecycle="lifecycle_id" >
    <SyncAttributes rule="attribute_id"
trigger="attr_trigger_id_minor">
        <SyncAttribute name="LOW" rule="attribute_id"
trigger="attr_trigger_id_major" />
    </SyncAttributes>
    <SyncRelations rule="relation_id" trigger="relation_triggers">
    </SyncRelations>
</SyncType>

<SyncInterface name="ALL" rule="interface_id" >
</SyncInterface>

</AdwSync>
```

Mapping Samples

This section explains how data exchange uses the mapping file while importing and exporting data.

Note: When you import data from an external system into the Allegro EDM Component Database, the data is mapped to the Allegro EDM system. When exporting, data from the Allegro EDM system is mapped to the external system.

Map part attribute name

To map the attribute name, the syntax is:

```
<Attribute source="foreign Name" target="adw Name"/>
```

Map part attributes value

To map the attribute value, there can be two cases:

- Attribute name map and the value map, as follows:

```
<Attribute source="foreign Name" target="adw Name">  
<Value source="foreign Value" target="adw Value"/>  
</Attribute>
```

- Only attribute value map as follows:

```
<Attribute source="foreign Name">  
<Value source="foreign Value" target="adw Value"/>  
</Attribute>
```

Map classification name

```
Interface source="foreign Name" target="adw Name">
```

Map classification feature name

Allegro EDM Data Exchange Reference Guide

Mapping Samples

To map the feature name, the syntax is:

```
<Feature source="foreign Attribute Name" target="adw Attribute Name"/>
```

Map classification feature value

To map the feature value, there can be two cases

- Map the feature name and its value as follows:

```
<Feature source="foreign Name" target="adw Name">
  <Value source="foreign Value" target="adw Value"/>
</Feature>
```

- Only map the feature value as follows:

```
<Feature source="foreign Name">
  <Value source="foreign Value" target="adw Value"/>
</Feature>
```

Implementing Pattern Matching

Data Exchange allows patterns in attribute names/values, classification names, and feature names/values. Let us see some illustrations on how to map patterns.

In Attribute name mapping

```
<Attribute source="cur*" target="current">
```

When data is imported, if a part attribute has a name that starts with `cur`, it will be mapped to `current`. When you export parts that have the `current` attribute, it will be mapped to `cur*`.

Disabling pattern matching

There might be cases where mapping needs to be disabled for some names that match the pattern. To disable mapping, DataExchange provides an attribute, `reverse`, in the mapping file.

While importing or exporting EDM data, to map the data to values that might have a pattern, set the mapping as follows:

Allegro EDM Data Exchange Reference Guide

Mapping Samples

```
<Attribute source="cur*" target="current" reverse="CURRENT">
```

Now, when data is imported, attributes that start with `cur` will be mapped to `current`. When data is exported, the `current` attribute will be mapped to `CURRENT`.

Examples

Let us look at some examples to better understand import and export mapping.

Example 1

```
<Attribute source="cur*" reverse="CURRENT">
```

```
<Value source="*" target="Released" reverse="RELAVDHESH"/>
```

```
</Attribute>
```

- While importing, all attributes that start with `cur` are mapped to `Released`.
- While exporting, all attributes that start with `cur` are mapped to `CURRENT`. Any attribute that has the value `Released` is mapped to `RELAVDHESH`.

In essence, when the target is not provided, the source is treated as a target.

Example 2

```
<Interface source="Externale.*" target="CAD Component  
Classification._UNCLASSIFIED_13 [v1.0]" reverse="External.PTC.CLASS" >  
  <Feature source="MYRE?" target="REV" revision="YOURREV">  
    <Value source="MYAC" target="AC" reverse="YOURAC"/>  
  </Feature>  
</Interface>  
</Mapping>
```

- While importing:
 - ❑ Any classification which starts with `External` will be mapped to `CAD Component Classification._UNCLASSIFIED_13 [v1.0]`
 - ❑ When matches are found for this criteria, and if any feature name matches `MYRE?`, it is mapped to `REV`
 - ❑ When matches are found for both the conditions listed above, any feature value that matches `MYAC` is mapped to `AC`

Allegro EDM Data Exchange Reference Guide

Mapping Samples

- While exporting:
 - ❑ Any classification which starts with `CAD Component`
`Classification._UNCLASSIFIED_13 [v1.0]` will be mapped to
`External.PTC.CLASS`
 - ❑ When matches are found for this criteria, and if any feature name matches `REV`, it is mapped to `YOURREV`
 - ❑ When matches are found for both, any feature value that matches `AC` is mapped to `YOURAC`

Index

Symbols

[] in syntax [8](#)
 {} in syntax [8](#)
 | in syntax [8](#)

A

action
 create [37](#)
 defer [38](#)
 ignore [37](#)
 major [39](#)
 minor [39](#)
 retain [39](#)
 update [38](#)
 Attribute Rules [44](#)
 attribute trigger [49](#)
 AttributeIgnored [50](#)
 attributes
 sync [52](#)
 AttributeUpdated [49](#)

B

benefits of synchronization framework [15](#)
 braces in syntax [8](#)
 brackets in syntax [8](#)

C

condition
 Default [44, 46](#)
 ObjectCreated [43, 45](#)
 ObjectDeferred [43, 45](#)
 ObjectIgnored [44, 46](#)
 ObjectMajor [44, 45](#)
 ObjectMinor [44, 45](#)
 ObjectRetained [44, 45](#)
 ObjectUpdated [41, 45](#)
 TargetBeingModified [38](#)
 TargetExist [37](#)
 TargetNameExist [42](#)

TargetNameNotExist [42](#)
 TargetNotExist [37](#)
 TargetRevisionEqual [40](#)
 TargetRevisionExist [43](#)
 TargetRevisionHigher [40](#)
 TargetRevisionLower [39](#)
 TargetRevisionNotExist [43](#)
 TargetTypeExist [41](#)
 TargetTypeNotExist [41](#)
 conventions
 user-defined arguments [8](#)
 user-entered text [7](#)
 create [37](#)

D

Default [44, 46](#)
 defer [38](#)

I

ignore [37](#)
 interface section [36](#)
 italics in syntax [8](#)

K

keywords [7](#)

L

lifecycle rules [47](#)
 literal characters [7](#)

M

major [39](#)
 minor [39](#)

O

- object rules [36](#)
- ObjectCreated [43](#), [45](#)
- ObjectDeferred [43](#), [45](#)
- ObjectIgnored [44](#), [46](#)
- ObjectMajor [44](#), [45](#)
- ObjectMinor [44](#), [45](#)
- ObjectRetained [44](#), [45](#)
- ObjectUpdated [41](#), [45](#)
- or-bars in syntax [8](#)

R

- relation
 - rules [40](#)
 - sync [52](#)
- relation rules [40](#)
- relation trigger [50](#)
- RelationCreated [50](#)
- RelationIgnored [51](#)
- RelationUpdated [50](#)
- retain [39](#)
- reverse sync engine [15](#)
- rules
 - attribute [44](#)
 - lifecycle [47](#)
 - object [36](#)
 - relation [40](#)
- rules section [35](#)

S

- section
 - interface [36](#)
 - rules [35](#)
 - types [36](#)
- settings.sync
 - abilities [27](#)
 - entries [29](#)
 - format [28](#)
 - location [28](#)
- sync attributes [52](#)
- Sync Configuration File See [sync.xml](#)
- sync engine
 - about [14](#)
 - in reverse [15](#)
 - two-way sync [16](#)

- sync relations [52](#)
- sync type [51](#)
- sync.xml [35](#)
 - about [14](#)
 - and the workflow [10](#)
 - structure [35](#)
 - syntax [52](#)
- synchronization framework
 - advantages [15](#)
 - benefits [15](#)
 - components [13](#)
 - flow [9](#)
 - input sources [13](#)
 - logic definition [14](#)
 - workflow [10](#)

T

- TargetBeingModified [38](#)
- TargetExist [37](#)
- TargetNameExist [42](#)
- TargetNameNotExist [42](#)
- TargetNotExist [37](#)
- TargetRevisionEqual [40](#)
- TargetRevisionExist [43](#)
- TargetRevisionHigher [40](#)
- TargetRevisionLower [39](#)
- TargetRevisionNotExist [43](#)
- TargetTypeExist [41](#)
- TargetTypeNotExist [41](#)
- trigger rules [49](#)
- two-way sync [16](#)
- types section [36](#)

U

- update [38](#)

V

- vertical bars in syntax [8](#)