



Quality Center Synchronizer
Adapter
Service Provider Interface (SPI)
Developer Guide

Version 1.2

Table of Contents

1	Introduction	4
2	Overview	5
2.1	Constraints in the Use of Quality Center Synchronizer	5
3	Implementing an Adapter	6
3.1	Linking the adapter-spi to its JavaDoc in Eclipse	6
3.2	Adapter Project Structure	6
3.3	Main Interfaces to Implement	7
3.3.1	Schema Builders.....	7
3.3.2	Handling Entity Types	9
3.3.2.1	Record Types Managers.....	10
3.3.2.2	Defect type.....	10
3.3.2.3	Requirement type.....	10
3.3.2.4	Folders Type.....	10
3.3.3	AttachmentHandler.....	10
3.4	Logger.....	11
3.5	Adapter Jar File Location	11
4	Working with the Testing Environment	12
5	The setup.xml file.....	14

©Copyright 2008 Hewlett-Packard Development Company, L.P.

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Quality Center Synchronizer Adapter Service Provider Interface (SPI) Developer Guide

1 Introduction

The Quality Center Synchronizer synchronizes records between Quality Center and other systems. Both systems are referred to as endpoints. To communicate with these endpoints, the synchronizer utilizes adapter modules through a common adapter SPI that is implemented by an adapter developer.

The adapter hides the specific implementation of an endpoint from the Synchronizer. For example, the Quality Center adapter wraps calls to Open Test Architecture (OTA).

A new adapter is deployed on the Quality Center Synchronizer server side in the "adapters" folder under the server installation.

The SPI is a Java SPI, which includes a set of interfaces that the adapter developer implements. In addition, the SPI provides a set of optional interfaces that the adapter can use to provide data.

Important: The usage of HP Quality Center Synchronizer or HP Quality Center Synchronizer Adapter SPI to develop an adapter for Quality Center or to synchronize Quality Center data with Quality Center data is not supported.

2 Overview

The details of the classes and packages are in the JavaDoc.

Main SPI Entry

- AdapterFactory: Entry point, creates an adapter
- Adapter: Main adapter interface, used to create a connection
- AdapterConnection: Provides all connection-dependent actions

Synchronization flow

The Synchronizer server uses adapter services to synchronize defects.

```
Adapter adapter;  
AdapterConnection ac = adapter.Connect(...)  
DefectManager dm = ac.getDefectManager("Defect");  
dm.getRecordIDs(...);  
...  
DefectTypeRecord dtr = dm.create(...)  
...  
dtr.fetchData(...)  
...  
dtr.update(...)  
...  
ac.Disconnect()
```

2.1 Constraints in the Use of the Quality Center Synchronizer

Systems that use the Quality Center Synchronizer must meet the following conditions:

1. ID uniqueness: A record's ID is unique within an endpoint. The ID is constant throughout the lifetime of a record. When an ID is passed by the adapter to the synchronizer, the ID always represents the same record until the record is deleted.
2. Link uniqueness: No two links have the same two endpoints.
3. Records can be modified in the the context of only one link: Only one link is used to synchronize the record between two endpoints. If a record is synchronized across more than one link, the result can be conflicts and false updates. However, a record can be copied without change from a source to more than one target endpoint (one-to-many link).
4. The Synchronizer does not support changing a field to mandatory or optional when a record's state changes. Fields are classified as mandatory only at record creation.

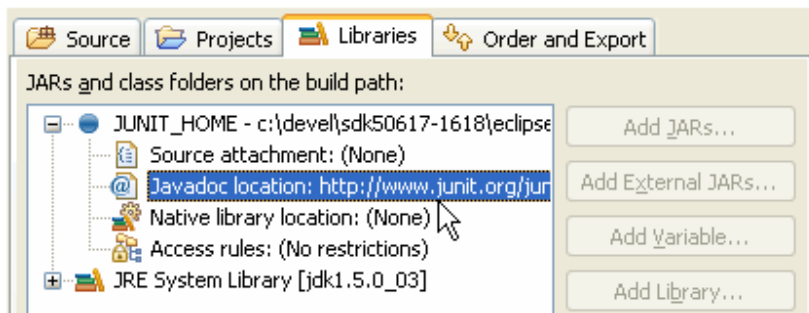
3 Implementing an Adapter

3.1 Linking the adapter-spi to its JavaDoc in Eclipse

If you are implementing your adapter in Eclipse, you can benefit from the Eclipse class generating wizard and JavaDoc support to generate your classes and methods with same parameter names as in the adapter-spi JavaDoc.

Configure the external JavaDoc documentation for SPI library as follows:

Open the build path page (**Project > Properties > Java Build Path**). Select **Libraries** and expand the node of the library so you can edit the 'Javadoc location' node. The documentation can be local on your file system (in a folder or an archive) or on a web server.



Adapters are delivered as a jar file stored under the Quality Center Synchronizer installation in the <Quality Center Synchronizer install folder>\adapters folder.

For class and method details, see the HP Quality Center Synchronizer Adapter SPI Javadoc.

3.2 Adapter Project Structure

Create a project in any Java IDE.

1. The adapter jar name follows the convention: <adapter name>-adapter.jar
2. The name of the package for the source files is:
com.hp.qc.synchronizer.adapters.<adapter name>
3. The adapter jar contains an adapter.xml file in its root folder. This file is in the format:

```
<?xml version="1.0" encoding="UTF-8"?>
<factory name="XML Adapter
Factory">com.hp.qc.synchronizer.adapters.xmladapter.XMLAdapterFactory</factory>
```
4. The adapter jar references only adapter-spi.jar (and not gossip-server.jar).
5. The adapter implements the interfaces in the com.hp.qc.synchronizer.adapters.spi namespace. For details, see below and refer to the JavaDoc provided with the SPI.

3.3 Main Interfaces to Implement

AdapterFactory

The entry point for adapters.

This factory reports the supported adapter types and creates the implemented adapter.

Adapter

Represents an adapter.

Each adapter has its own set of unique connection parameters and connecting method.

AdapterConnection

Represents an adapter after connection.

After connecting, an adapter can check users' permissions, handle entities with the applicable entity manager, check connection status, and so on.

AdapterConnection also holds entity managers instances, that will be created per connection and cached.

The flow is as follows:

Synchronizer server:

```
Adapter ad = ...;
```

After instantiation, the server calls the Adapter method to populate the parameters:

```
ad.getConnectionParams();
```

Adapter:

The adapter method returns the parameters. The declaration is:

```
Map<String, String> getConnectionParams():
```

In Clear Quest, for example, the parameters for connection are the schema repository and the database number. Because they are mandatory parameters, their mapped value is an empty string:

```
map.add("SchemaRepository", "");  
map.add("Database", "");  
return map;
```

If there are any optional parameters, they are added to a map with their default values. Example:

```
map.add("DatabaseType", "Postgres");
```

When the adapters are connected, tasks can be performed and information about the adapter can be queried.

3.3.1 Schema Builders

For each adapter, implement a schema builder class named *<adapter name>*SchemaBuilder that declares the endpoint's schema when requested by the owner link. The response includes the details of:

1. The entity types supported by the adapter (Defects, Requirements, etc.).
2. The names of the filters that exist in the endpoint for the entity type.
3. The fields at the endpoint that are available for mapping (DATE, NUMBER, SINGLE_VALUE_LIST, etc.). For each field, the response includes the attributes, such as name, maxlength, readonly, requiredness, and so on, and the fixed values, if any.

When the AdapterConnection object calls declare EntityTypes, the schema builder is queried for the entity types supported by the endpoint and populates these entity types in the EntityTypesBuilder parameter.

The adapter populates the EntityTypesBuilder with its types and declares the types' subtypes, if any.

Example of declaring entity types:

Synchronizer server:

```
AdapterConnection conn = ad.connect(...);
EndpointEntityTypes eet = new EndpointEntityTypes();
// Call the adapter method to declare the entities
conn.declareEntityTypes(eet);
```

Adapter:

The declaration of the adapter method is declareEntityTypes(EntityTypesBuilder etb).

For example:

```
etb.declareType("Bug","DEFECT"); //name and type
```

Examples of building schemas:

Synchronizer server:

```
EntitySchema epSchema = ...;
// Call the adapter method to declare the build the schema
conn.buildEntitySchema(epSchema, epSchema.getName());
```

Adapter:

The server call: conn.buildEntitySchema(EntitySchemaBuilder esb, ...) causes the following sequence.

```
MySchemaBuilder sb = new MySchemaBuilder(...);
sb.buildEntitySchema(entSchema, entityType);
```

...

```
build<entity name>EntitySchema(esb);
buildFilterSchema(esb);
```

...

```
esb.addFilter(String path);
```

...

If there are sub-types:

```
SubtypeSchemaBuilder subtypeSchemaBuilder = esb.addSubtype(name, identifier);
```

...

If the entity type has sub-types, the schema includes the schema of each sub-type.

EntitySchemaBuilder implements EntityFieldsSupport

```
esb.addField(fieldName, ...) or esb.addListField(fieldName, ...)
```

```
...
```

```
ListFieldValue listFieldSchema = entitySchema.addListField(...);  
listFieldSchema.setFixedList(true/false);
```

```
//addListField return value is the new ListFieldSchema which implements //FieldSchema, so  
setMaxLength can be used.  
listFieldSchema.setMaxLength(fieldMaxLength);
```

You can use filters to reduce traffic between the endpoints by synchronizing only records that match the filter.

A filter can be defined under a specific folder by using its full path: "Folder1\Folder2\filter". Do not use slashes ('\') in a filter name except to indicate its path.

There are limits to the number of fields, values per list, and filters. The maximum values are defined in the PARAMS table in the synchronizer's database. The parameters are:

- MAX_NUMBER_FIELDS (default value: 500)
- MAX_VALUES_LIST_FIELD (default value: 50)
- MAX_NUMBER_FILTERS (default value: 500)

If the maximum number of any of these entities is reached, the next item is not added. A warning is printed to the log.

3.3.2 Handling Entity Types

The adapter handles the various record types, the Defect type, the Requirement type, and the various folder types.

3.3.2.1 Record Types Managers

The following classes are implemented if the adapter supports the record type.

Defect type

- DefectManager
Manages actions related to defect type on a specific endpoint, such as creating a new defect and retrieving all existing defects.
- DefectTypeRecord
Represents a single defect record of the adapter.
Implementation note: Do not fetch the record's full data to memory until explicitly requested.

Requirement type

- RequirementManager
Manages actions related to Requirement type and Folder type on a specific endpoint, such as creating a new requirement or getting the interface to a folder.
- RequirementTypeRecord
Represents a single requirement or folder record of the adapter.
Implementation note: Do not fetch the record's full data to memory until explicitly requested.

3.3.2.2 Folders Type

If the adapter supports folders, the manager for the type of records to be stored in the folder type supports both the data type and the folder type. For example, the Quality Center adapter supports folders for requirements. Therefore, the RequirementManager type handles RequirementTypeRecord and FolderTypeRecord. Defect folders are not supported. Therefore, the DefectManager handles only DefectTypeRecord.

3.3.3 AttachmentHandler

The AttachmentHandler provides operations for maintaining the attachments for a single record. An adapter for an endpoint that does not support attachments does not need to implement this interface.

3.4 Logger

The AdapterLogger logs all important information to synchronizer logs. To ensure that the messages facilitate analysis, provide all information needed to correct the problem, while keeping the messages clear and concise.

Recommendations for writing clear messages for each log category:

- info:
Log the entrance and exit of every method. For example, in the connect() method:
"Connect(...) called with parameters x=.. y =.."Connect() completed"
- warning and error:
Log when something unexpected happened and before throwing any exception.
Write detailed messages. For example:
"CreateRecord(...) failed. User <username> does not have read permissions."
- debug:
Issue whenever a general message is needed.
- fatal:
Use only for errors that prevent further processing, such as a full collapse of the adapter.

Note: Only synchronization info, warning, and error level messages are written to the AdapterLogger.

Note: If your application writes error or fatal level messages to the AdapterLogger during integrity checks to the AdapterLogger, the integrity check fails.

3.5 Adapter Jar File Location

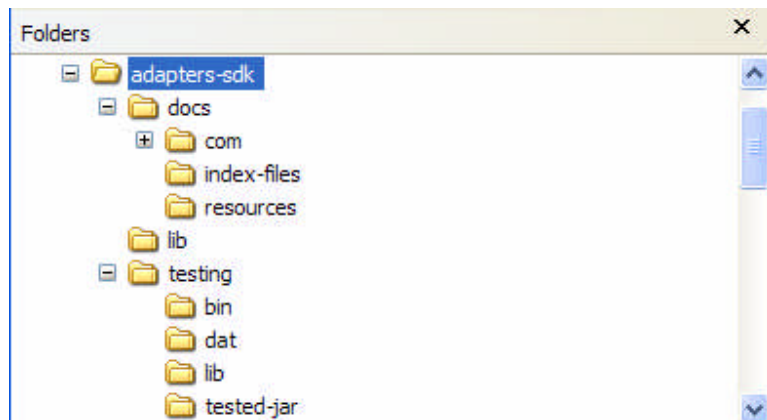
After implementation, place the adapter jars in <Quality Center Synchronizer install folder>\adapters\lib. Place any other jars your adapter requires in this directory.

Some jar files are included in the installation in the <Quality Center Synchronizer install folder>\adapters\lib directory. Do not replace existing jars with different versions.

4 Working with the Testing Environment

A framework for testing and demonstrating most of the SPI methods is provided with the SDK in the adapter-testing.jar archive.

Folder Structure



The adapter adapters-sdk folder contains three folders:

- **docs:** This document and java documentation
- **lib:** The SPI jar file
- **sample:** Example of adapter implementation (xml adapter)
- **testing:** The testing framework

The adapter-spi jar file is also located in the adapters-sdk\testing\lib folder.

Important: The framework tests change data in a live project. Run the tests on projects that do not contain important data and are safe for multiple creation and deletion.

To work with the framework without debugging:

1. Create an adapter.
2. Place the adapter jar file in the testing\tested-jar folder.
3. Create a setup xml file (described below) and place it in the testing\dat folder.
4. Add any other required jars to the lib folder.
5. Launch the test using the RunAutomatedTests.bat batch file in the bin folder, specifying the XML file name and log file name as parameters. There is no need to specify a path for the XML file.

Important: xercesImpl.jar (XML parser for Analyzer/UI) must be in the tested project.

If debugging is required, perform the following in addition to the above:

1. Create a project in any Java IDE using the sources in the sources jar (located at the testing folder).
2. Copy the four subfolders of testing into the project's root folder.
3. Put a jar in the testing\tested-jar folder. The jar file must contain with the adapter.xml file as described above in section Adapter Project Structure. The file does not have to contain classes.
4. Link the project to the adapter-spi jar, the junit jar, and to your adapter project.

5. When launching the project in the IDE for debugging, add these variables to the command line:
-Dxmlname="`<your xml name>`" -Djava.ext.dirs="`<adapter-testing project location>\lib`".
The quotes are a required part of the variable command-line syntax.

4.1 The setup.xml file

The schema for the setup.xml file, AdapterTestingData.xsd, is in the dat folder.

Example of a Setup.xml file:

```
<?xml version="1.0" encoding="UTF-8"?>
<AdapterTestingData>
  <JarFilePath> Full path to your jar file, with file name </JarFilePath> see Note (1)
  <Adapter name=" adapter name "
    entityType=" REQUIREMENT "
    entityName=" Name "
    subTypeName=" 3 "
    supportsFolders="true"> see notes (2) and (3) and (6)
    <ConnectionData>
      <User> login user name</User>
      <Password> login password</>
      <Parameter> (can have many parameters)
        <Name> Parameter_name</Name>
        <Value> Parameter_Value</Value>
      </Parameter>
    </ConnectionData>
    <MandatoryFields> (1st entry is used for record creation)
      <Field modified_in_adapter=" NO">
        see Note (4), can have many fields
        <Name> field name</Name>
        <Value type=" STRING"> New Req</Value> see Note (5)
      </Field>
    </MandatoryFields>
    <MandatoryFields> (2nd entry is used for record updates)
      <Field> (also can have many fields)
        <Name> Name</Name>
        <Value type=" STRING"> Updated Req</Value>
      </Field>
    </MandatoryFields>
  </Adapter>
</AdapterTestingData>
```

Note (1): Ensure that the jar file path reflects the actual location of the file when you work in different locations.

Note (2): subTypeName is either be a sub-type's unique identifier (in Quality Center, this is the Requirement Type Key) or NO_SUBTYPE (for defects).

Note (3): entityType is either DEFECT for defect type records or REQUIREMENT for requirement type records.

Note (4): modified_in_adapter can have the values NO (testing framework can test for equality on values taken from adapter), YES (no test can be performed on this field), or PARTIAL (value in XML is fully contained in value for adapter).

Note (5): value type is a value from the list: USER_LIST, STRING, NUMBER, DATE, MULTI_VAL_LIST, SINGLE_VAL_LIST.

Note (6): supportsFolders attribute is an optional attribute, which indicates to the testing framework whether the system should perform folder-related tests. This attribute is relevant only for requirements. Set this attribute to true for adapters that support folders or false if the adapter does not support folders.