



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

Version 1.3

Table of Contents

1	Introduction	4
2	Overview	5
3	Implementing an Adapter	7
3.1	Linking the adapter-spi to its Javadoc	7
3.2	Adapter Project Structure	7
3.3	Main Interfaces to Implement	8
3.3.1	Schema Builders	9
3.3.2	Handling Entity Types	11
3.3.2.1	Record Types Managers	12
3.3.2.2	Folders Type	12
3.3.3	AttachmentHandler	12
3.4	Logging	13
3.5	Exception handling	13
3.6	Adapter Jar File Location	13
3.7	Adapter DAT Folder	14
4	Working with the Testing Environment	15
5	The setup.xml file	16
6	Sequence Diagrams	17
6.1	Check Connectivity Diagram	17
6.2	Integrity Check Diagram	18
6.3	Synchronization Diagram	19

© Copyright 2009 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 Quality Center and the other system are called endpoints. To communicate with endpoints, the Synchronizer uses 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 Synchronizer server in the "adapters" folder under the server installation.

The SPI is a Java SPI that 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.

For code examples, see the sample XML adapter supplied in the SDK package.

Limitation: Use of the HP Quality Center Synchronizer or the HP Quality Center Synchronizer Adapter SPI to develop an adapter for Quality Center or to synchronize data between Quality Center projects of installations is not supported.

2 Overview

The details of the classes and packages are in the SPI Javadoc.

Definitions

Endpoint	A set of parameters that identify a target system to which the adapter can connect.
Record	A record in the underlying system exposed by the adapter. A record can be either a DEFECT or a REQUIREMENT.
Record ID	An identifier that is unique within the context of the endpoint and the entity type represented by the record.
Modified Record	A record that requires synchronization. A record is modified if its version has changed since the last synchronization and at least one of the fields participating in the synchronization has changed.

Constraints in Using the Quality Center Synchronizer

Systems that use the Synchronizer must meet the following conditions:

- 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. IDs of deleted records are not reused.
- Link uniqueness: No two links that are configured to synchronize the same entity type have the same two endpoints. Endpoints are the same if they have same connection information (server address, project name, and so on).
- Only one link is used to synchronize a record between two endpoints. However, a record can be copied without change from a source to more than one target endpoint (one-to-many link).
- 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.

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 records between Quality Center and the system exposed by the adapter.

For example:

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

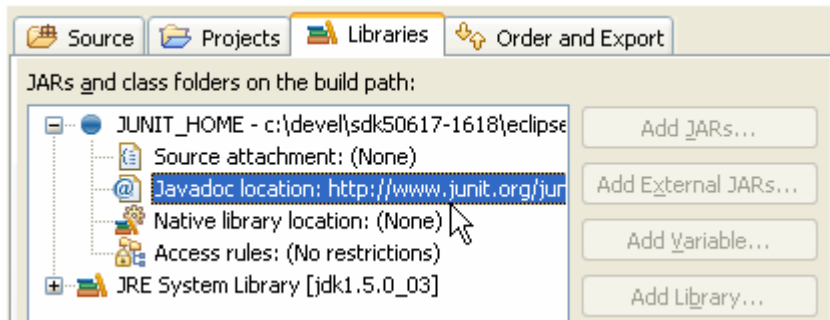
For a detailed description of the flow "Sequence Diagrams" in this document.

3 Implementing an Adapter

3.1 Linking the adapter-spi to its Javadoc

Before you start implementing your adapter, link the SPI Javadoc to the adapter-spi jar. This allows your IDE to generate classes and methods with readable parameter names. For example, if you are implementing your adapter in Eclipse, you can 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 to edit the 'Javadoc location' node. The documentation can be local on your file system (in a folder or archive) or on a web server.



An Adapter is delivered as a jar file stored under the 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

A Synchronizer adapter must have at least one JAR file that satisfies following conditions:

- The adapter jar name follows the convention: `<adapter name>-adapter.jar`
- The adapter jar contains an adapter.xml file in its root folder. This file is in this format:

```
<?xml version="1.0" encoding="UTF-8"?>
<factory name="XML Adapter Factory">
com.hp.qc.synchronizer.adapters.xmladapter.XMLAdapterFactory
</factory>
```

For a full example of an adapter.xml, see the example adapter supplied with the SDK package.

- The adapter code references classes provided in adapter-spi.jar. It does not reference classes in gossip-server.jar.
- 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

The following sections outline the set of interfaces and classes that must be implemented by custom adapter code. For more information on the interfaces described below, see the Javadoc.

AdapterFactory

This class is the first communication point between the Synchronizer and the adapter. AdapterFactory provides the Synchronizer with information about the adapters provided by the adapter library. The Synchronizer calls AdapterFactory to create instances of these adapters.

Adapter

Adapter is the gateway class to the synchronization logic provided by the adapter library. The Adapter class represents the adapter in its stateless condition before the Synchronizer requests that the adapter connect to the system it represents.

The Adapter class exposes the adapter's set of unique parameters and connection method. The parameter list is used by the Synchronizer to construct the connection dialogs with which the user defines a link. The parameters together with the values entered by the user are supplied to the adapter at later stages for creation of a connection object.

If the adapter requires extra configuration information that cannot be expressed as either Connection Parameters or as Endpoint Parameters, the adapter can save this configuration information in a properties file in the adapter "dat" folder. The adapter can use any file format because only the adapter itself reads it. For more information about the "dat" folder, read the "Adapter "dat" Folder" section of this document.

AdapterConnection

AdapterConnection represents the adapter in its connected state.

This class is used by the Synchronizer to check users' permissions, handle entities with the applicable entity manager, check the connection status, and so on.

AdapterConnection also holds references to entity manager instances that are created per connection and cached. It is the responsibility of the AdapterConnection object to cache these instances and return the same instance each time the Synchronizer requests them.

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 optional parameters, they are added to a map with default values. For 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:

- The entity types supported by the adapter. The possible types are:
DEFECT – A non-hierarchical entity with no sub-type entities.
REQUIREMENT – A hierarchical entity that has one or more sub-types.
- The names of the filters that exist in the endpoint for each entity type.
- The fields at the endpoint that are available for mapping, for example: DATE, NUMBER, and SINGLE_VALUE_LIST. 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 declareEntityTypes, 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' sub-types, 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", EntityType.DEFECT); //name and type
or
etb.declareType("ChangeRequest", EntityType.DEFECT);
```

Examples of building schemas:

Synchronizer server:

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

Adapter:

The server call conn.buildEntitySchema(EntitySchemaBuilder esb, ...) initiates 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 that 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, respectively:

- 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 Type 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 implement this interface. For usage examples, see the test package supplied with the SDK.

3.4 Logging

The AdapterLogger writes to synchronizer logs. For information on the logs, refer to the Synchronizer documentation. 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 are:

- 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 an 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.

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

It is possible to change the logging level of the Synchronizer server at runtime without restarting the server. Change the log4j.properties file, which is located in the "dat" folder located in the Synchronizer installation folder.

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 Exception handling.

The two types of errors most commonly thrown from an adapter and caught by the Synchronizer are AdapterException and FatalAdapterException. For other possible exceptions, see the Javadoc.

AdapterException generally indicates to the Synchronizer to stop the current task and continue to the next task. For example, in case of synchronization, if an adapter throws this exception while a record is being synchronized, this record is skipped and the Synchronizer advances to the next record.

FatalAdapterException indicates to the Synchronizer that the entire operation cannot continue, and should be halted. For example, this exception is thrown if the adapter enters an invalid state from which it cannot recover, such as the endpoint becoming not available.

3.6 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 folder.

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

3.7 Adapter “dat” Folder

The adapter “dat” folder is a general purpose folder intended to be used by the adapter for storage of both adapter configuration and resource files.

The path of this folder is:

<Quality Center Synchronizer installation folder> \adapters\dat<AdapterName>**

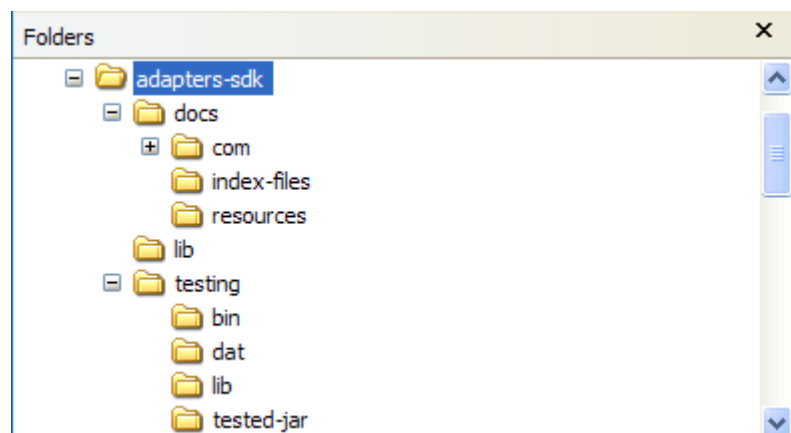
The adapter can resolve the location of this folder by invoking the `AdapterContext.getDatDirPath()` method.

The adapter can use this folder for storage of configuration files or any other resources that cannot be represented using the Connection Parameters or the Endpoint Parameters.

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 the following folders:

- **docs**: This document and the 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.

Warning: 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. Place xercesimpl.jar (the XML parser for the Analyzer UI) in the tested jar folder.
4. Create a setup xml file (described below) and place it in the testing\dat folder.
5. Add any other required jars to the lib folder.
6. 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.

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 in the testing folder).
2. Copy the four subfolders of testing into the project's root folder.
3. Put a jar with the adapter.xml file in the testing\tested-jar folder. The file does not have to contain classes. See section Adapter Project Structure.
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.

5 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 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).

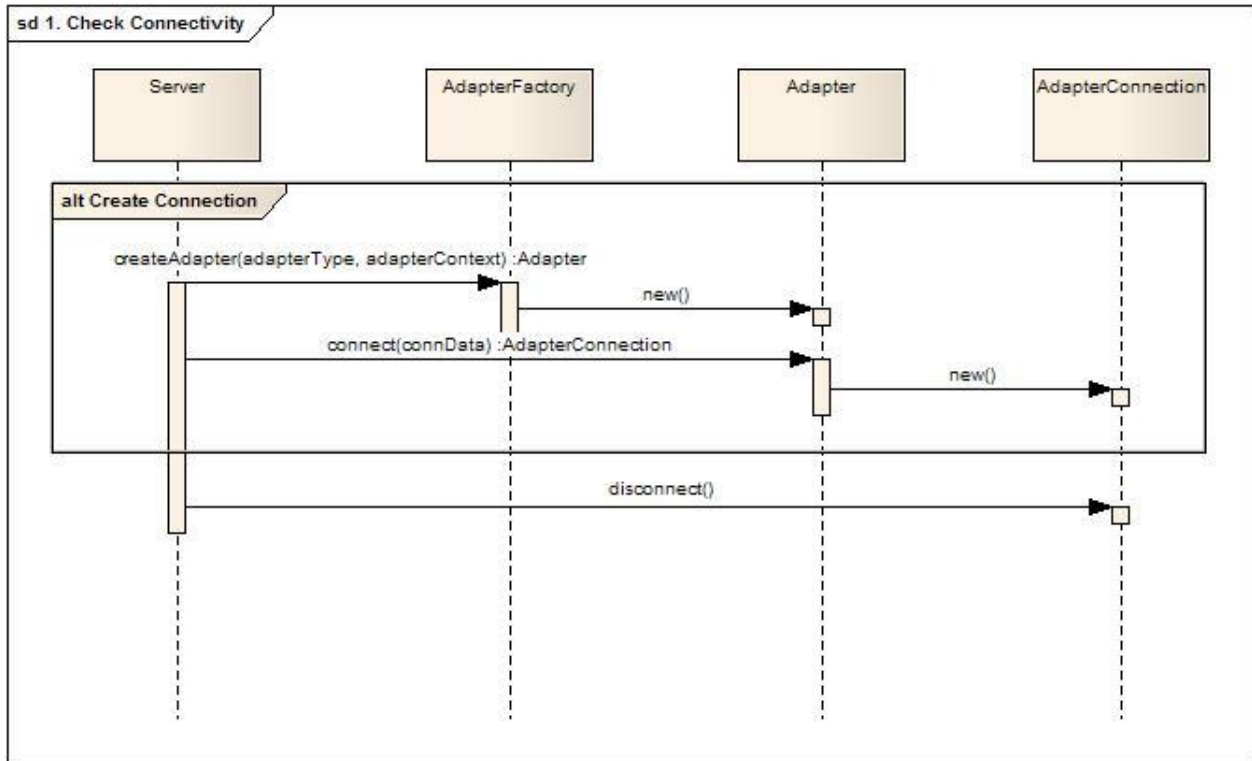
Note (5): value type is a value from the list: USER_LIST, STRING, NUMBER, DATE, MULTI_VAL_LIST, or SINGLE_VAL_LIST.

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.

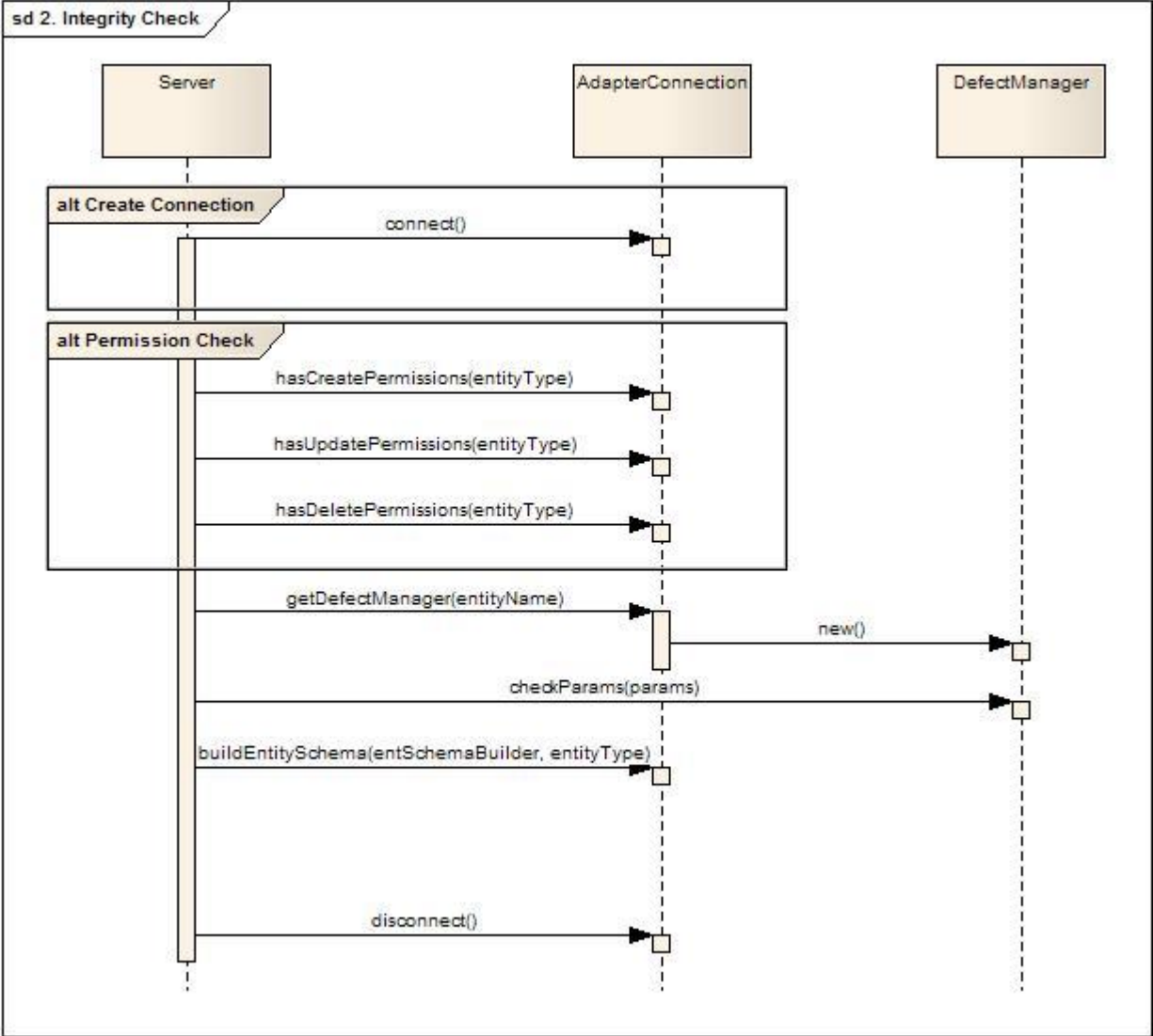
6 Sequence Diagrams

The following diagrams display the main interactions between the Quality Center Synchronizer and an adapter implementation. For detailed explanation of the roles of each class and the method calls, refer to the Javadoc supplied with the SPI.

6.1 Check Connectivity Diagram



6.2 Integrity Check Diagram



6.3 Synchronization Diagram

