HP OpenView Service Quality Manager

Service Adapters Software Development Toolkit

Development Guide

Edition: 2.0

for the HP-UX Operating Systems

March 2007

Warranty

The information contained herein is subject to change without notice. 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.

License Requirement and U.S. Government Legend

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.

Copyright Notices

Trademark Notices

Adobe®, Acrobat® and PostScript® are trademarks of Adobe Systems Incorporated.

HP-UX Release 10.20 and later and HP-UX Release 11.00 and later (in both 32 and 64-bit configurations) on all HP 9000 computers are Open Group UNIX 95 branded products.

Java™ and all Java based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

Microsoft® , Windows® and Windows NT® are U.S. registered trademarks of Microsoft Corporation.

Oracle® is a registered U.S. trademark of Oracle Corporation, Redwood City, California.

UNIX® is a registered trademark of The Open Group.

X/Open® is a registered trademark, and the X device is a trademark of X/Open Company Ltd. in the UK and other countries.

Origin

Printed in France.

Contents

Chapter 1 8

Introduction 8

1.1 Required features implemented by Service Adapters 8
1.2 Architecture 9
1.3 Concepts 9
1.3.1 Registration 9
1.3.2 Discovery 10
1.3.3 Measure 11
1.3.4 Configuration 13
1.3.5 Resource 13

Chapter 2 15

Installation and Configuration 15

2.1 Pre-requisites 15
2.1.1 Installing J2SE Software 15
2.1.2 Setting-up Java environment 15
2.1.3 Installing Apache TomCat 15
2.2 Installing the software 16
2.2.1 Required environment 16
2.2.2 Installing the SA SDK 16
2.3 Uninstalling the software 17
2.4 Product configuration 18
2.4.1 Setting-up Ant environment 18
2.5 Packaged Third Party Products 19
2.5.1 Apache Axis 19

Chapter 3 20

Product Description 20

3.1 Product Content 20
3.1.1 Sample Service Adapter description 20
3.1.2 SQM Simulator description 25
3.1.3 SDK API Documentation 34

Chapter 4 Service Adapter development guidelines 35

4.1 Defining the development environment 35
4.2 Compiling an SQM Service Adapter 36
4.3 Installing a SQM Service Adapter 37
4.4 Deploying a SQM Service Adapter 37
4.5 Testing a SQM Service Adapter 37

Chapter 5 38

Implementing a Service Adapter 38

5.1 Designing a Service Adapter 38
5.1.1 Registration Service 39
5.1.2 Configuration Service 42
5.1.3 Discovery Service 44
5.1.4 Measure Service 48

Chapter 6 60

Debugging, Troubleshooting and Tracing 60

6.1 Debugging a SQM Service Adapter 60
6.1.1 Required environment 60
6.1.2 Setting up the project environment 60
6.1.3 Setting up the Eclipse project 61
6.1.4 Deploying the SampleSA services 63
6.1.5 Debugging the Service Adapter 63

Preface

This document is a guide to the development and implementation of Service Adapters with OV SQM Service Adapters SDK. It allows developers to focus on the application code to be developed independently from SQM concepts and architecture.

This manual contains the recommended procedures for OV SQM Service Adapter development, many of them illustrated by an example. It provides coding examples but does not contain in-depth descriptions of Service Adapter API. This API is explained in detail SDK Java documentation provided as a set of HTML documents in the product kit.

In addition, this document describes how to install and configure the OV SQM Service Adapters SDK.

This document describes how to:

- Install the Service Adapter SDK (and required products)
- Deploy, execute and test the Sample Service Adapter
- Guideline for developing a Service Adapter
- Development Tips

Intended Audience

This document is intended for experienced network managers and system software developers who want to develop an SQM Service Adapter and integrate it into OV SQM using Service Adapter SDK.

Required Knowledge

It is assumed that the reader is familiar with the functionality of Service Quality Manager and has previous experience of the following:

- Java programming
- Web Service Technology
- System administration and operations
- Service Level Management

It is assumed that the reader is familiar with the concepts described in the following books:

- HP OpenView Service Quality Manager Overview.
- HP OpenView Service Quality Manager Service Adapter User's Guide.

Software Versions

The software versions referred to in this document are:

Product Version	Operating System
OpenView Service Quality Manager 1.4 OpenView SA Software Development KToolkit 2.0	HP-UX 11i Windows XP

Typographical Conventions

`Courier` Font:

- Source code and examples of file contents.
- Commands that you enter on the screen.
- Pathnames
- Keyboard key names

Italic Text:

- Filenames, programs and parameters.
- The names of other documents referenced in this manual.

Bold Text:

- To introduce new terms and to emphasize important words.

Associated Documents

The following documents contain useful reference information:

- HP OpenView Service Quality Manager Service Adapter User's Guide

For a full list of Service Quality Manager user documentation, refer to the HP OpenView Service Quality Manager Product Family Introduction.

Support

You can visit the HP OpenView support web site at:

http://support.openview.hp.com/support.jsp

This Web site provides contact information and details about the products, services, and support that HP OpenView offers.

HP OpenView online software support provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued support customer, you can benefit by using the support site to:

- Search for knowledge documents of interest
- Submit enhancement requests online
- Download software patches
- Submit and track progress on support cases
- Manage a support contract
- Look up HP support contacts

- Review information about available services
- Enter discussions with other software customers
- Research and register for software training

Chapter 1

Introduction

hp OpenView SQM provides a complete service quality management solution. It consolidates quality indicators across all domains — telecom, IT networks, servers, and applications — providing end-to-end visibility on service quality. SQM links service quality degradations to potential effects on business, allowing network support operators to address problems and prioritize actions proactively.

SQM monitors the service quality by aggregating performance or quality indicators collected from various data sources, such as the network, the IT infrastructure, and the service provider's business processes. Using this information, service operators can pinpoint infrastructure problems and identify their potential effects on customers, services, and service level agreements (SLAs).

The SQM platform has a southbound composed of various data sources integration modules. These data feeder specific integration modules are called Service Adapters (SA). Service Adapter allows collecting data on various sources and mapped them into performance or quality indicator in a model expected by SQM.

The purpose of the Service Adapter Software Development Kit (SDK) is to provide a generic interface for feeding SQM with the collected performance or quality indicator. Using the SDK the integrator has a development and deployment environment that allows him to quickly produce standalone Service Adapters. The following schema locates the Service Adapters, produced by the integrator, once deployed.

1.1 Required features implemented by Service Adapters

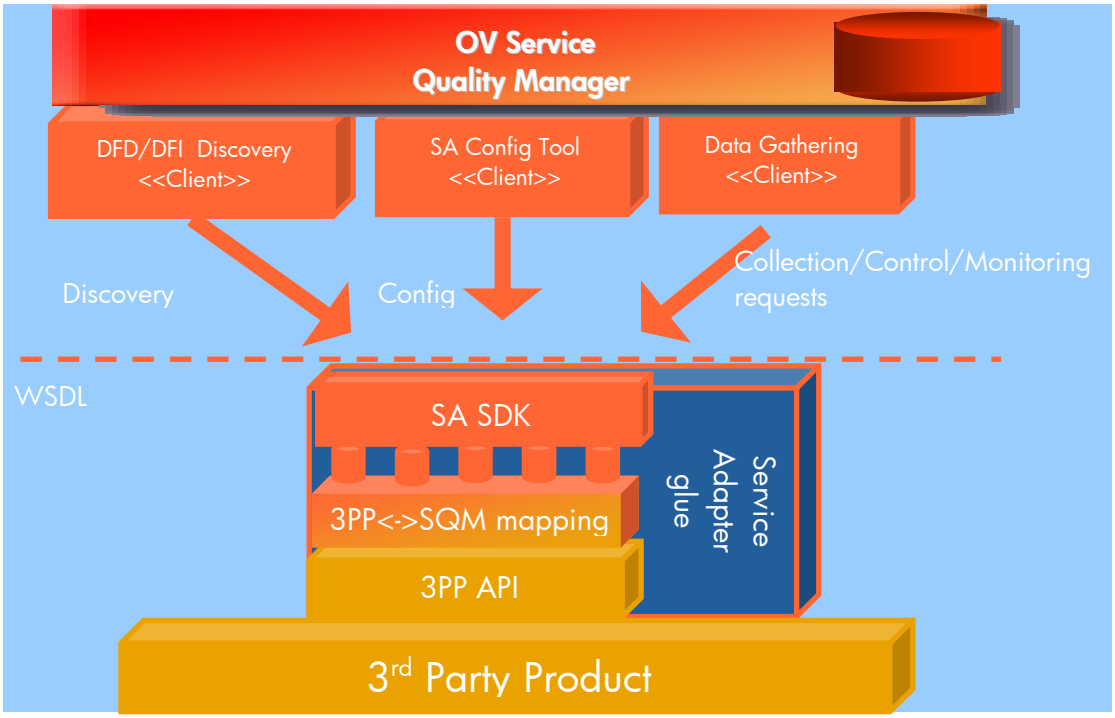
The role of a Service Adapter is mainly:

- **Model Discovery [optional]:** expose to SQM, the model (DFD) of indicators which can be collected from the Third Party Product
- **Instance Discovery [mandatory]**: expose to SQM the DFD instances (DFIs) providing indicators collected from the Third Party Product
- **Controlling the collection [mandatory]:**: to allow SQM to start/stop collecting raw data from the Third Party Product
- **Resynchronization [optional]:** to provide to SQM a way to resynchronize itself (retrieving raw data that it has not completely processed yet)
- **Collection [mandatory]:** to feed SQM with the raw data provided by the Third Party Product:
- **Repair** [**mandatory]:** to auto repair collections' broken resources and to expose to SQM a way to attempt new repairs when the auto repair failed

- **Registration [mandatory]:** to setup (and destroy) a context for each "Service Adapter client" (on the SQM platform), and expose (share) these contexts to the features (services).
- **Configuration [optional]:** to update Service Adapters and Third Party Products configurations.

1.2 Architecture

The following schema represents the Service Adapters, highlighting their role of adapting Third Party Products to an SQM platform. Thus, the Service Adapters communicate on the northbound with the SQM platform (processes) using a standard WSDL interface, crossing a firewall. Note that the calls are always initiated by the SQM platform. Indeed, each call establishes a connection to a HTTP web server, on the Service Adapter platform. Therefore the eventual firewalls have to be configured to authorize outgoing connections (only on one port) from the SQM platform and to allow incoming calls for the Service Adapter platform. Finally, on the southbound the Service Adapters interact with their dedicated Third Party Products.



1.3 Concepts

The previous chapter exposed an overview of the Service Adapter features. The details and their classifications (categories) are provided hereafter. Each following paragraph describes a category of features, its purpose and the functionality.

1.3.1 Registration

The Service Adapter implementation communicates on its southbound with the SQM platform. The SQM platform might have, mainly for scalability reasons, several clients (instances) accessing one single Service Adapter. As most features are state full operations, the Service Adapter has to maintain a context for each client.

Therefore, each client-request references the context that has been allocated to the client. That way the Service Adapter recovers a context for each incoming request. The Service Adapter attributes a new context to each client who is performing the request. As each request is context related, the clients (uniquely identified) have always to request a context before performing its requests. Each context has a unique ID. To avoid any collisions within the context referencing system, the contexts IDs could only be reallocated, as soon as the context associated previously to this ID has explicitly been destroyed (through the entire SA life-cycle: even crashes). A simple Service Adapter implementation of this constraint is to allocate unique identifiers that are unique over time with respect to the host on which they are generated. Note: A context ID is named registrationId at the Service Adapter interface level.

Although the scalability requires client context scoped requests, the context scoped requests are also used for the Service Adapter crashes or clean-up detection. Indeed, all client requests refer a context, which is maintained by the Service Adapter. Thus any client request, which fails because it is referencing a context that is unknown by the Service Adapter, allows the client to deduce that the Service Adapter crashed or cleaned-up since the context creation.

The contexts are destroyed on client request, when the client stops (or in some case, before it crashes). To prevent context leaks (remaining undestroyed contexts), the Service Adapter has to monitor the clients' activities. This monitoring could trigger context garbage collections. Indeed, "inactive" clients are either "disconnected" from the Service Adapter due to a crash or network problems, or they might no more collect Data Feeders. When a client is no more collecting Data Feeders for a time laps, the allocated resources has to be freed. This way, "inactive" clients no longer monopolize the Service Adapter and "dead" clients' allocated resources get wiped out. Finally, the Service Adapter has also to invalidate the contexts that are allocated to clients that are claiming again a context without having requested the destruction of their previous context. This might happen when a client crashed. Therefore the client has to request its context destruction, before he is able to claim a new context in order to perform further requests. This context invalidation is mandatory, as otherwise the Service Adapter and the client might own different Data Feeder states.

The context garbage collecting has of course to de-allocate all related resources. Typically, all the referenced Data Feeder collections have to be stopped (and freed), as well as the referenced external uploaded resources have to be removed (confer Resource paragraph).

1.3.1.1 Versioning visible at registration level

The Service Adapter implementation will evolve over time, independently of the interface (WSDL). Therefore the SQM platform has to be aware of these changes. Indeed, a new revision of an implementation could be available for updated Service Adapter clients, whereas the previous Service Adapter might still be available to none upgraded systems. The origin of a Service Adapter implementation change might be a simple performance improvement or maybe a fix of a functional miss behavior.

This versioning information is visible to the Service Adapter clients (SQM platform) within the context identification. As the context is shared among all features, it is a good place to include versioning controls. Therefore the context ID (registrationId) includes a version number.

1.3.2 Discovery

The SQM platform requires a full description of the Data Feeders in order to be able to perform collections.

The Data Feeder overall description has to be provided by the Discovery in two steps.

First SQM needs the Data Feeder Definition (DFD), that defines what kind of indicators (parameters) the Data Feeder collects and which properties allow to uniquely identify the location where these indicators are collected (measurement reference point schema). Note that the Data Feeder Definition defines a full set of properties, but in general only a subset of these properties is used to uniquely identify the collection location. The DFD are identified through their name and their version. These DFD descriptions could be modeled by the integrator using the SQM Service Designer. Indeed, it is possible that this type of information is not offered by the Third Party Product. In this case the Service Adapter might use alternative mechanism retrieve this data.

The second description, mandatory, provides the Data Feeder description. A Data Feeder is described by what (name and version of the DFD) it collects and where (measurement point). This “what” and “where” information uniquely identifies a DF. The Service Adapter has to provide the descriptions of the Data Feeder it is aware of. The Service Adapter implementation will most likely query the Third Party Product to achieve this task.

As exposed above the DF description might thus exist for a DFD that has been provided to the SQM platform by hand. This type of use cases could introduce several discrepancies. Therefore it is recommended that the Service Adapter also provides the DFD descriptions.

1.3.3 Measure

The features allow SQM to control the collection of Data Feeders and to retrieve the collected measures and statuses changes. The Data Feeder collection control is composed of: a start since a given measure, a stop and a repair operation.

All Service Adapter implementations support Data Feeder collections. The SQM platform whereas require some additional capabilities for its (optional) synchronization mechanism. Therefore, the Service Adapter should support some historical measures querying. Indeed, to allow the SQM platform to recover from its eventual crashes, SQM requests from the Service Adapter the (historical) measures collected in the meantime of its crash.

Thus, we distinguish two kinds of Service Adapter implementations: those who support this historical measures querying, and those who don’t. The implementations declare (provide) their category through the MeasureDeliveryPolicy. Confer to the Java documentation for further details. The measures, provided to the SQM platform, have to be uniquely identified, to allow historical measures querying. Therefore, the Service Adapter implementations supporting this querying feature have of course also to be able to attribute a unique ID (MeasureId) to the measures. The SQM platform could therefore use these IDs for its synchronization mechanism.

The Service Adapter implementation category will mainly depend on the Third Party Products capabilities. Indeed, if the Third Party Product does not provide some query functionalities, the Service Adapter implementation won’t probably support the historical measures querying, too.

Thus the collected measures are either current measures, or historical measures (as those described in the former paragraph). Each measure is time stamped. The current measures might have, like historical measures, timestamps in the past as they are most likely polled from or listened on the Third Party Products. For example, measures polled, every 2 minutes, by the Service Adapter implementation from the Third Party Product are available only up to 2 minutes later.

The Data Feeder collection not only provides the current and historical measures, but also the status changes. Thus measures are declined into value measures or Data Feeder status measures. A status measure is generated by the Service Adapter each time the status, of a Data Feeder collection, changes. This status measure has to be

generated regardless of the action which triggered the change: directive called by SQM, failure or Third Party Product event, etc. Now, how does the SQM platform retrieve these measures? For technical reasons, as previously exposed, the SQM platform permanently fetches the measures accumulated by the Service Adapter. Thus, the SQM platform retrieves a flow (stream) of collected measures. The fetch operation will behave in two ways. It is either blocking for a configurable time, when the Service Adapter currently has not accumulated measures, or it returns all the currently accumulated measures. The fetch operation uses a maximum fetch size providing a way to retrieve "huge" amounts of measures in bunches. The Data Feeder collection controls are implemented by the following directives. The stop directive stops the collection and maybe frees the associated resources. When resources (like sockets) are used for several Data Feeder collections, the Service Adapter has of course to de-allocate the resources only once all the associated Data Feeder collections are stopped.

The collect directive has a different behavior depending on the Service Adapter implementation category (those who support the historical measures querying, and those who don't). Indeed, the Service Adapter implementations which don't support historical measures querying, will only put in place the resources in order to start the Data Feeder collection. The Service Adapter implementations, supporting historical measures querying, whereas put first in place the resources in order to start the Data Feeder collection and (only) then perform some queries to retrieve the historical measures (last available one, or since a given measure) as requested by the SQM platform. Note that a collect directive on already collecting Data Feeders should whereas only perform the eventual historical measures queries, keeping in place the existing collection resources. The collected measures have to be accumulated by the Service Adapter implementation, until the SQM platform fetches them. For details on the historical requests behavior refer to Java documentation (collect directive). The repair directive is called by the SQM platform as soon as a collecting Data Feeder is detected as being "Failed", "Off line", etc… Indeed, when the Service Adapter implementation or the Third Party Product fails to maintain a Data Feeder collection, an availability status change measure has to be generated. Therefore the SQM platform will be aware of this degradation and will be able to perform Data Feeder repair trials. After several repairs tries, the SQM platform will leave unchanged the "un-repairable" collections, whereas, for the "repaired" Data Feeders collections, it will, depending on the policies and the provided status changes, query for the historical measures that the platform eventually missed (confer collect directive). The Service Adapter repair directive implementation should only try to reestablish (repair) the resources (sockets, etc.) of the listed Data Feeder collections (dataFeederIds array). Therefore the Data Feeder collections associated to the repaired resources might collect again. Note that the directives are dedicated to Data Feeders, whereas the measures retrieval is global to all Data Feeders. Thus, the SQM platform will retrieve all the accumulated measures, regardless of the Data Feeder to which these measures belong to. The Data Feeder collection control directives always produce a Data Feeder status. These Data Feeder statuses are provided two times: once as a directive result, allowing an "immediate" feed back to the SQM platform, and a second time within the collected measures flow.

This Data Feeder status duplication is used to cope with the asynchronous aspect of this measure service. Indeed, each directive (Data Feeder collection control) impacts the flow of collected measures, which is consumed independently. This measure flow of values and statuses, produced by the Service Adapter implementation, thus transcribes the Data Feeder collections' chronological "life-cycle" as a context free source. Consider this life-cycle as a simple sequence of Data Feeder statuses and value measures. Therefore the Data Feeder statuses resulting of a collection control directive have also to be provided through the collected measure flow. Remember that these services are moreover built on top of a transport layer. In order to cope with any Data Feeder collection control directive invocation delivery delay problem, due to the transport (or application communication) layer implementations, a transaction

context is provided by the SQM platform. Indeed, notice that no guarantee could be provided on the fact that the SQM platform will be aware of the status-change, first, through the directive result and, only then, through the collected measures flow, and vice and versa. The collected measures (values and statuses) could even reach the SQM platform through the collected measures flow a few seconds after the corresponding collection control directive has been processed. Some extreme situations even show that depending on the volume of the accumulated measures and the rate of the control directives, it might happen that a Data Feeder collection has a given status at SQM level, whereas the collected measures flow still provides values for the previous statuses. Take the example where a Data Feeder collection is collecting, before it is suddenly stopped and started again. At the same time other Data Feeders generate higher amounts of measures. Therefore the SQM platform processing will be lagging behind the value- and status-measures generation. The SQM platform will handle values provided for the former started Data Feeder collection (before the collection has been stopped and started again). This situation would lead on the SQM platform to a de-correlation between the Data Feeder status and the measure flow processing. In other circumstances, the Data Feeder might even be considered as available, whereas in fact it is failed. Indeed, each Data Feeder collection status could be placed in the context of a directive and therefore a transaction. Thus the SQM platform has to attribute a new "transaction id", each time it calls a directive (collected, repair, stop). The Service Adapter has to use this transaction id, not only to discard the out-dated directives, which have been delivered after another more recent control directive, but also to mark all future status-measures resulting of the processing of the various directives. Therefore the SQM platform could discard the status- (and value--) measures that didn't result of the latest requested directive. Note that the SQM platform allocates, per Data Feeder collection, a distinct transaction Id number, incremented on each directive call. A filtering based on discarding measures whose timestamp is previous to the directive processing is not possible, as it is difficult to put in place due to: eventual time jitter, or time differences between the SQM, the Service Adapter and the Third Party Product platforms, etc. Moreover the Service Adapter implementation would otherwise have to eventually realign the Third Party Products measures' timestamps, which is prohibited.

1.3.4 Configuration

As described above the SQM platform accesses a range of Service Adapters adapting different Third Party Products in order to feed the SQM system. Therefore SQM is the ideal place for centralizing the Service Adapters and Third Party Products configuration deployment. The integrator has thus only to store on the SQM platform its different configuration resources. These resources could be any arbitrary file bundles, binary or text files. It is only recommended that the size of this resource is not too huge (< 1MB), unless it is accessible through an URL. Using the configuration tools the integrator could then upload to the Service Adapters and the Third Party Products platforms the required configuration resources, in order to request a global or single configuration deployment. Resource uploads are classified as being resource related features.

1.3.5 Resource

The Service Adapter configurations require resources uploads. These upload features are classified as being resource related features. (Confer the former Configuration paragraph for further details on the resource contents.) As resource uploads are in most cases time consuming, the upload process is split into steps. First a resource ID is allocated by the Service Adapter for a declared resource. The declaration provides the name and the type of the resource. This resource ID is used by the SQM tools to perform a resource upload request. An upload could be performed in two different ways. Either the resource content is streamed (one-way) to the Service Adapter. Or

only the URL for the resource is send to the Service Adapter. In this last case, the Service Adapter will read the resource content from the specified URL. Finally, the SQM tools could independently check that the upload finished and succeeded. Take care, that the resources updated to the Service Adapter are removed as soon as the associated context is destroyed (confer1.3.1 Registration). Also note that a resource could be declared several times. But each declaration generates a new temporary resource on the Service Adapter side.

In order to provide an I18N support for error messages, the Service Adapter offers within this Resource classification an error code translation operation.

Chapter 2

Installation and Configuration

2.1 Pre-requisites

Before installing the OV SQM Service Adapter SDK it is mandatory to install the following third party products:

1. The Java 2 Standard Edition Software Development Kit (SDK)
2. An application server, we recommend Apache TomCat 4.1.31

2.1.1 Installing J2SE Software

To install install J2SE kit:

3. Download the Java 2 Standard Edition (J2SE) SDK, from: http://java.sun.com/j2se/
4. Install the SDK according to the instructions included with the release.
5. Set an environment variable JAVA_HOME to the pathname of the directory into which you installed the SDK release.

2.1.2 Setting-up Java environment

To setup the JAVA environment it is necessary to:

1. Set the JAVA environment variable (it depends where the product is installed)
   ```
   # JAVA_HOME=<java installation directory>/java ; export
   JAVA_HOME
   ```
2. Update the PATH environment variable:
   ```
   # PATH=${JAVA_HOME}/bin:${PATH} ; export PATH
   ```

2.1.3 Installing Apache TomCat

It is recommand to install Apache TomCat 4.1.31. A servlet container is mandatory for executing a SQM Service Adapter. The software kit and documentation can be downloaded from the following location: http://jakarta.apache.org/tomcat/

1. Download the binary distribution of Tomcat
2. Unpack the binary distribution into a convenient location (for instance /opt)
3. Set an environment variable CATALINA_HOME to the path of the directory into which you have installed Tomcat

2.2 Installing the software

This section describes how to install the SQM Service Adapter SDK on a HPUX system.

SA SDK does not require that the OV SQM Kernel is already installed and configured on the primary host. SA SDK is independent from any SQM component.

Anyway it can be installed on an SQM director.

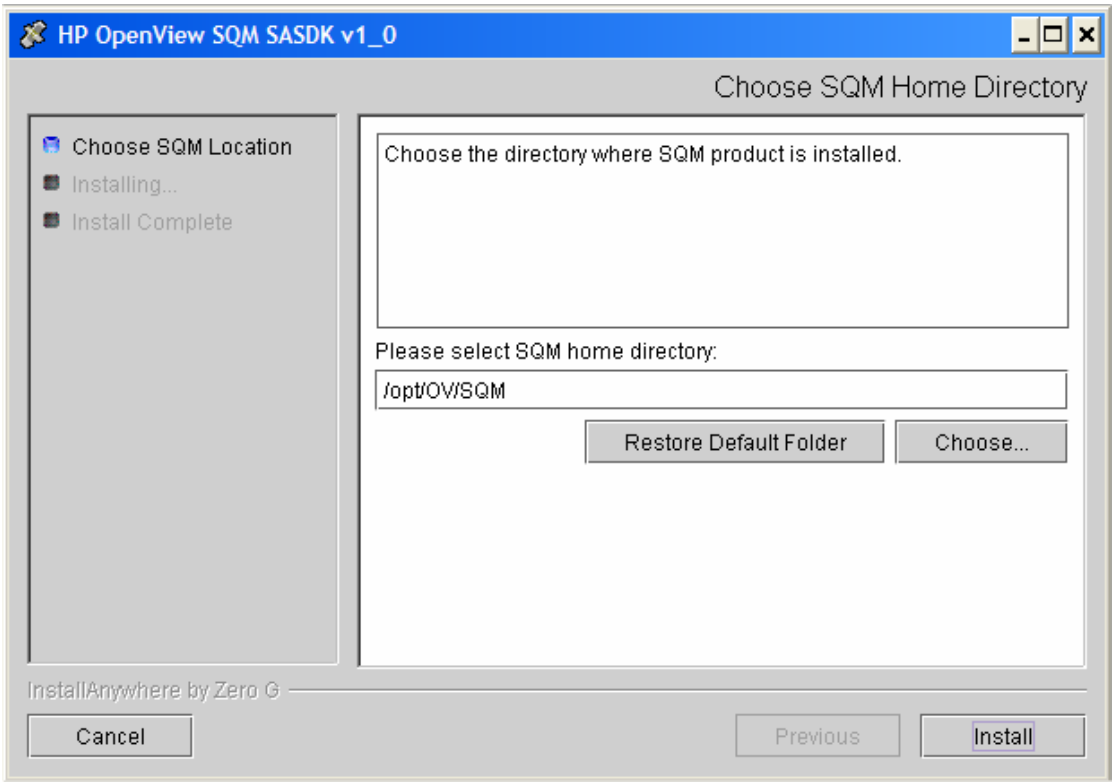
2.2.1 Required environment

To configure the SA SDK, you first have to set the Java and TomCat environment as describes in chapter Chapter 2.

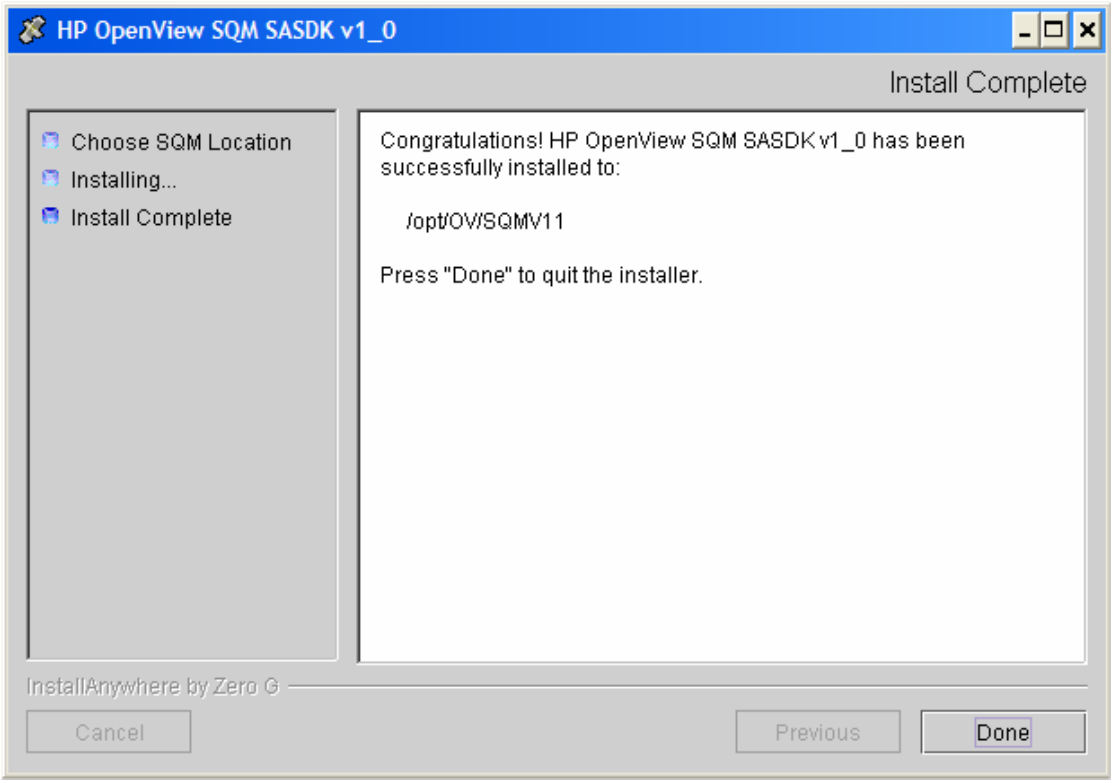
2.2.2 Installing the SA SDK

To install the SA SDK perform the following steps:

1. Log on as root user.
2. Mount the Service Adapters and Gateways CD-ROM on your system
3. Go to the SQM-1.40.00-SAGTW/HPUX directory
4. Run the 'SQMSASDK-2.00.00.bin' installer.



5. Select the installation directory. If SQM is already installed on the target system, we recommend to install the SA SDK in the SQM installation directory (directory refered by the environment variable $TEMIP_SC_HOME)

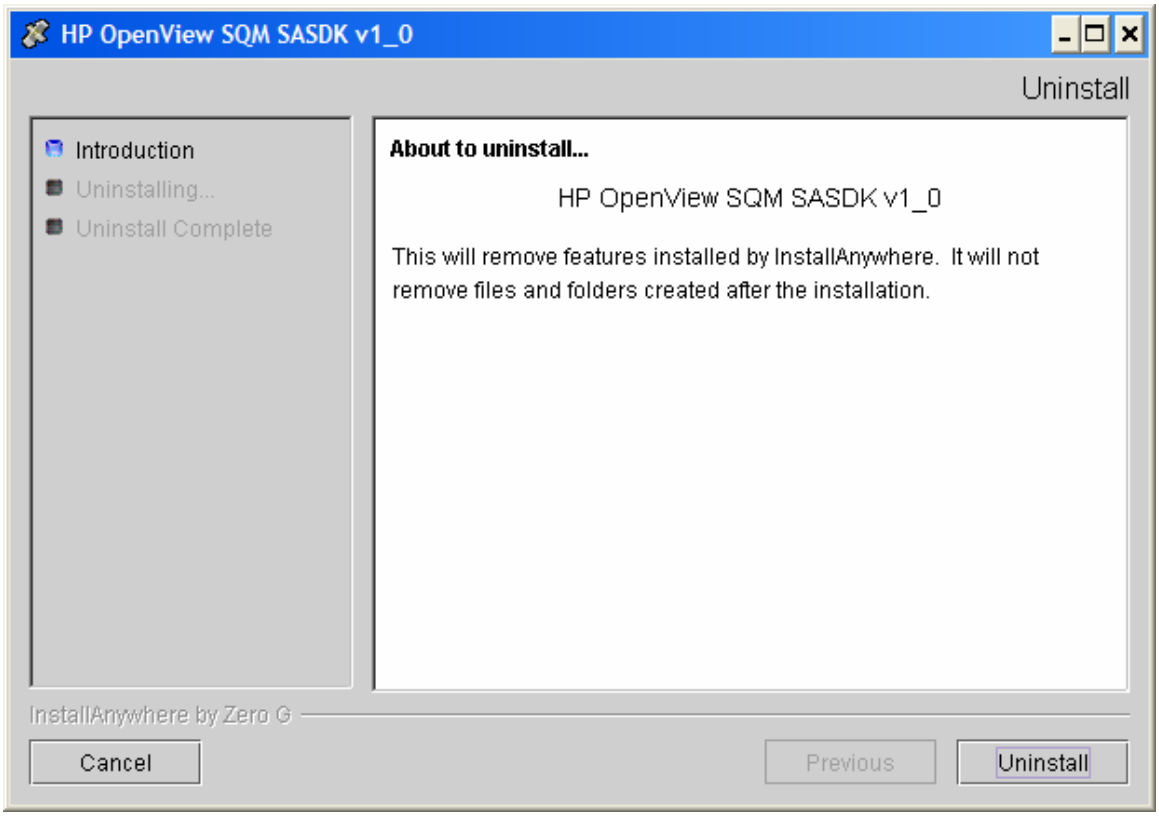


6. To end the installation process, click ‘Done’

2.3 Uninstalling the software

To uninstall the SA SDK, perform the following commands:

1. Log on as root user.
2. Go to the SA SDK installation directory
   `#cd $TEMIP_SC_HOME/ServiceAdapters/SDK/v2_0/Uninstaller_SASDK`
3. Run the Un-installer:
   `#Uninstall_SASDK`
4. The uninstaller window appears, click on Uninstall button.
5. When the un-installation is terminated the following window appears:



6. Click “Done” to accept.

2.4 Product configuration

The SA SDK requires that Ant environment is setup. This setup is mandatory for compiling and executing SA SDK components.

The TEMIP_SC_HOME environment variable is not necessary for the SA SDK. Any way, in this document, this variable refers to the kit installation directory.

2.4.1 Setting-up Ant environment

Apache Ant is a Java-based build tool. It is delivered in the SA SDK at the following location:

${TEMIP_SC_HOME}/ServiceAdapters/SDK/v1_0/ant-1.5.1

To setup the Ant environment it is necessary to:

1. set the ANT_HOME environment variable:
   ```
   # ANT_HOME=${TEMIP_SC_HOME}/ServiceAdapters/SDK/v1_0/ant-1.5.1; export ANT_HOME
   ```
2. Update the PATH environment variable:
   ```
   # PATH=${ANT_HOME}/bin:${PATH} ; export PATH
   ```

2.5 Packaged Third Party Products

2.5.1 Apache Axis

Axis is essentially a SOAP engine -- a framework for constructing SOAP processors such as clients, servers, gateways, etc.

The SA SDK proposes an implementation of Service Adapters based on Apache Axis.

For more information about Apache Axis, please refer to the following Apache Axis Web Site:
http://ws.apache.org/axis/

Chapter 3

Product Description

3.1 Product Content

The SA SDK Software kit is a SQM Service Adapter development environment. It is composed by:

- A set of WSDL files defining Service Adapter interface
- A Service Adapter sample. This SA is delivered as:
    - a web archive (war file). Therefore the Service Adapter sample deployment only requires copying the shipped war file to the user's web application container (Apache Tomcat, etc.)
    - the source files
- A command line tool named SQMSimulator, which can be used to validate Service Adapter services
- A compilation environment based on Apache Ant. The Sample Service Adaptor component is also shipped with an Ant file which can be re-used as a standard Service Adaptor Ant file
- A User documentation:
    - A Development User Guide
    - A Java documentation, describing Service Adapter interface

3.1.1 Sample Service Adapter description

The Service Adapter SDK includes a Service Adapter implementation sample. This sample provides to the SDK's users a canvas for the implementations he has to produce. The sample responds like concrete implementations to Data Feeders discovery, control and collection requests, but it does not interact with a Third Party Product, to execute. Instead, the sample implementation only produces predefined results. Therefore, some Data Feeders could be controlled, in order to collect and provide fake measures, whereas others are dedicated to raise standard errors.

The SDK ships the sample source code, but it also provides a file hierarchy that all Service Adapter implementations should follow. Therefore the automatic implementation packaging producing web applications could be used. These web applications are deployable on web application containers.

3.1.1.1 Data Feeder Model exposed by the Sample Service Adapter

The Sample Service adapter exposes 3 datafeeder definitions:

1. Multimedia DFD: example of video measure

2. OVIS HTTP DFD: same DFD as the HTTP DFD supported by the SQM OVIS Service Adapter
3. TeMIPFaultStatistics DFD: The TeMIP Fault Statistics DFD collects fault statistics parameters on TeMIP managed entities based on the alarms collected in TeMIP Operation Context monitored and integrated with the Service Quality Manager platform.

3.1.1.2 Installating the Sample Service Adapter

The Sample Service Adapter must be installed in a Servlet Container. Installation depends on the selected Servlet Container. In most cases it consists in dropping the Sample SA war file (located in $TEMIP_SC_HOME/ServiceAdapters/SDK/v1_0/SampleSA/war) in the webapps directory of the server and restarting the server, or by using a server-specific mechanism to enable the web application.

For instance, installing the Sample Service Adapter on TomCat 4.1 consists in:

1. Log as root user
2. Copy the war file in the TomCat webapps directory:
   ```
   # cp
   $TEMIP_SC_HOME/ServiceAdapters/SDK/v1_0/SampleSA/war/SampleSA
   _v1_0.war $CATALINA_HOME/webapps
   ```
3. Create the Sample SA data directories. These directories are used for properties, trace and logging files:
   ```
   # mkdir /var/opt/SA_SDK
   ```
4. Start TomCat server (if not already started)
   ```
   # $CATALINA_HOME/bin/startup.sh
   ```

Note

- After copying the war file into the TomCat webapp directory it is not necessary to restart the TomCat server, the web application will be deployed on the fly.
- If the directory `/var/opt/SA_SDK` does not exist, the SampleSA, will copy the default property file in the user directory

3.1.1.3 Validate the Sample Service Adapter installation

To validate the Service Adapter installation you should make sure that the server is running the web application.

To test the installation, using your favorite web browser, look for the available services by entering the following URL:

http://localhost:8080/SampleSA_v1_0/services

You should obtain the following result:

And now... Some Services

- AdminService *(wsdl)*
  - AdminService
- Version *(wsdl)*
  - getVersion

Note

The Service Adapter services are not already deployed (see section 3.1.1.4). The defaults services allow deploying the Service Adapter web services.

3.1.1.4 Deploying the Sample Service Adapter services

The Sample Service Adapter implements the various services. The Sample SA is based on the Axis SOAP Engine (see chapter 2.5.1). It remains to tell Axis how to expose these services. Axis takes a Web Service Deployment Descriptor (WSDD) file that describes in XML what the services are, what methods it exports and other aspects of the SOAP endpoint.

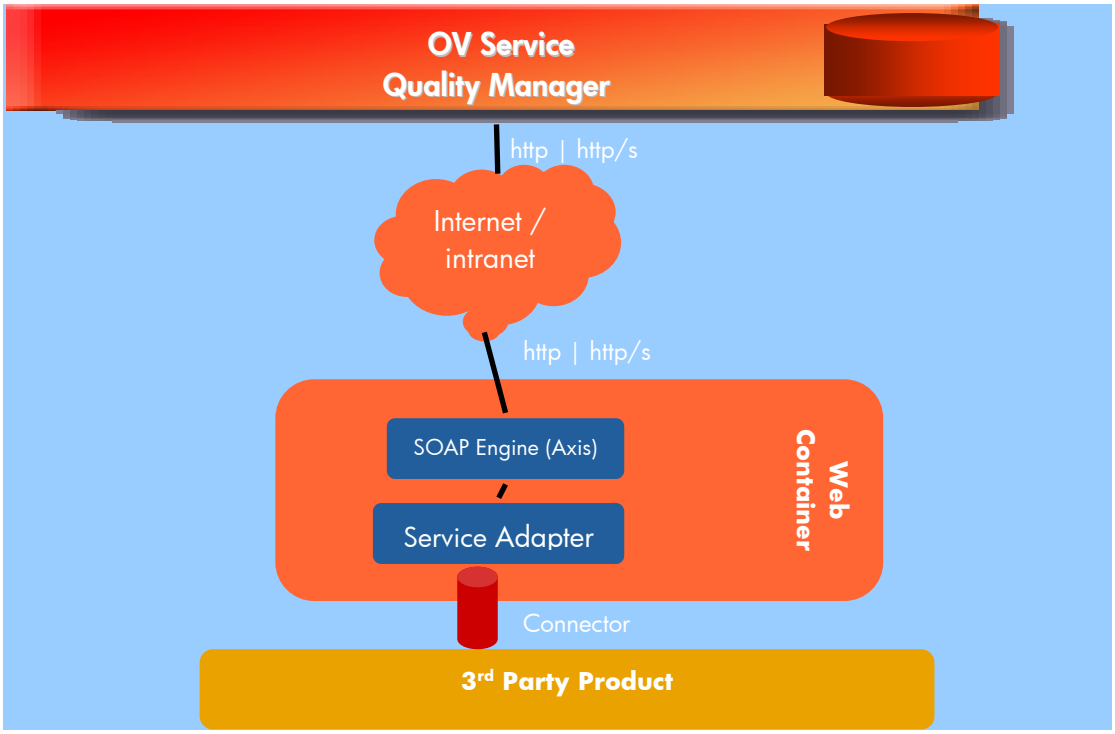
Execute the following command to perform the deployment:

```
# $TEMIP_SC_HOME/ServiceAdapters/SDK/v1_0/SampleSA/bin
/deploy.sh
```

This command deploys the Sample Service Adapter services on a local Web server configure with the http port number 8080. To deploy the Sample Service Adapter services on a remote Web server or on a different http port number, use the option '–s <webapp hostname>' to specify a different hostname, or the option '–p <webapp port number>' to specify a different port number.

Note

This command allows also undeploying the Sample Service Adapter services by add the option '-r'.



3.1.1.5 Testing the Sample Service Adapter services

To test the installation and the deployment of the Sample Service adapter, using your favorite web browser, look for the available services by entering the following URL:

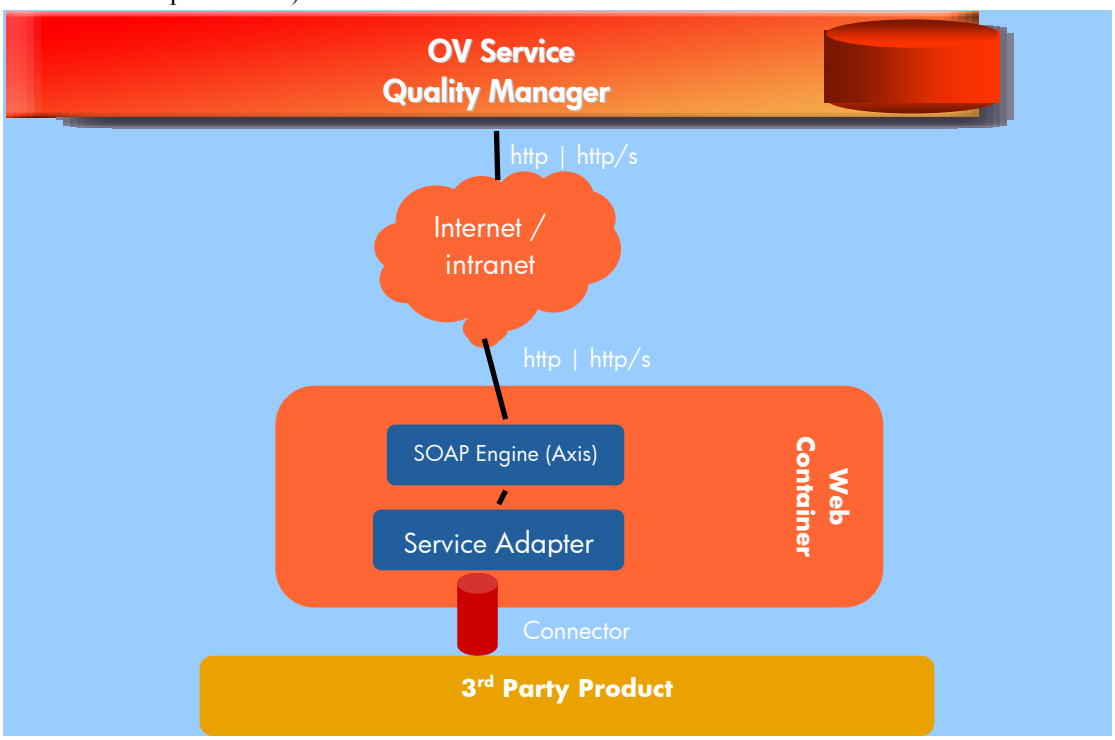
http://localhost:8080/SampleSA_v1_0/services

You should obtain the following result:

And now... Some Services

- ResourceService *(wsdl)*
  - getCodeTranslations
  - declareResource
  - uploadResourceContent
  - uploadResource
  - checkResourceUpload
- DiscoveryService *(wsdl)*
  - getDFs
  - getDFDs
- AdminService *(wsdl)*
  - AdminService
- Version *(wsdl)*
  - getVersion
- ConfigurationService *(wsdl)*
  - acceptConfiguration
  - deployConfiguration
  - getMeasureDeliveryPolicy
- RegistrationService *(wsdl)*
  - register
  - deregister
- MeasureService *(wsdl)*
  - repairDFCollections
  - stopDFCollections
  - getDFsInternals
  - collectSinceDFMeasures
  - getCollectedDFMeasures

To call the Sample Service Adapter's services, uses the SQM Simulator utility (see chapter 3.1.1.6).



3.1.1.6 Service Adapter configuration

The Sample Service Adapter offers many configurable parameters allowing to control for instance the number of DFIs/DFD, messages throughput…

These parameters are located in:

/var/opt/SA_SDK/SampleSA.properties

This file is created at this location in the directory /var/opt/SA_SDK/ exists when starting the Service Adapter.

The Sample Service Adapter provides the following configurable parameters:

Property	Default value	Description
configuration_directory_name	/var/opt/SA_SDK	Directory in which the property file is copied at startup
configuration_file_name	SampleSA.properties	Name of the copy of the property file
configuration_tmp_file_name	/tmp/config_tmp_file	When the new configuration file is provided thank to the Configuration service, this property specify the name of the temporary file in which the provided property file is stored before being applied
logs.level	WARNING	Logging level
trace.level	ALL	Tracing level
trace.files.location	/var/opt/SA_SDK/trace/	Trace file location
log.files.location	/var/opt/SA_SDK/log/	Log file location
trace.file.name	SampleSA_Trace.log	Trace file name
log.file.name	SampleSA_Log.log	Log file name
ARE_EVENTS_CHRONOLOGICAL	True	
RETRIEVE_HISTORICAL_MEASURES_UPON_THEIR_IDS	True	
OvisHTTP_STATUS	AVAILABLE	Status of the OvisHTTP DFs. Possible values: ▪ AVAILABLE: the collection on this DFD is enabled ▪ FAILED: the collection on this DFD is disabled
Sql_MMediaDF_STATUS	AVAILABLE	Status of the Sql_MMediaDF. DF. Possible values: ▪ AVAILABLE: the collection on this DFD is enabled ▪ FAILED: the collection on this DFD is disabled
TeMIPFaultStats_STATUS	AVAILABLE	Status of the Sql_ TeMIPFaultStats DF. Possible values: ▪ AVAILABLE: the collection on this DFD is enabled ▪ FAILED: the collection on this DFD is disabled
OvisHTTP_PERIOD	15000 milliseconds	Acquisition period for the measures of the OvisHTTP DFD (in milliseconds)

Sql_MMediaDF_PERIOD	20000 milliseconds	Acquisition period for the measures of the Sql_MMediaDF DFD (in milliseconds)
TeMIPFaultStats_PERIOD	5000 milliseconds	Acquisition period for the measures of the TeMIPFaultStats DFD (in milliseconds)
TeMIPFaultStats_NUMBER_OF_MEASURE_BEFORE_FAILURE	10	Defines the number of measures done before the collection for the DFD TeMIPFaultStats fails down.
discovery.df.count	1	Number of DFs per DFD when performing a DF discovery
discovery.df.naming.start	1	DF names are prefix by a number. This property defines the first used number for naming

3.1.2 SQM Simulator description

The SQM Simulator tool is command line driven. Each command line allows executing on a service, an operation with a set of required arguments. The configuration of this tool is based on a property file.

The tool is located in:

$TEMIP_SC_HOME/ServiceAdapters/SDK/v1_0/SQMSimulator/bin/SQMSimulator.sh

3.1.2.1 Usage

Command tool usage

```
> SQMSimulator -service <service name> -operation <operation name> -arguments '<operation arguments>'
```

or

```
> SQMSimulator -service <service name> -operation <operation name> -argumentsfile <file>
```

Where:

- **-service <service name>:** The name of the service. The available services are described Table 1 SQM Simulator: available Services and Operations
- **-operation <operation name>:** The operation name. The available operations for each service are described Table 1 SQM Simulator: available Services and Operations
- **-arguments <operation arguments>**: The operation's arguments, encapsulated within a specific XML document. The inputs arguments format is described in the can be Table 2 SQM Simulator input arguments format. Format:
  <Arguments>
  <first argument> ….
  <second argument> ….

…..
</Arguments>

- **-argumentsfile <file>:** file defining the operation arguments, encapsulated within a specific XML document. The inputs arguments format is described in the can be Table 2 SQM Simulator input arguments format.

Description of the available services and operations

The following table sums up the available operations and the required parameters.

Service name	Operation name	Operation arguments	Returned data type
registration	register	▪ ClientId	▪ RegistrationId
	deregister	▪ ClientId	▪ void
discovery	getDFs	▪ RegistrationId	▪ ArrayOfDataFeederId
	getDFDs	▪ RegistrationId	▪ ArrayOfDataFeederDefinition
measure	repairDFCollections	▪ RegistrationId ▪ ArrayOfDataFeederControlId	▪ ArrayOfDataFeederStatusReport
	stopDFCollections	▪ RegistrationId ▪ ArrayOfDataFeederControlId	▪ ArrayOfDataFeederStatusReport
	getDFsInternals	▪ RegistrationId ▪ ArrayOfDataFeederId	▪ ArrayOfDataFeederInternals
	collectSinceDFMeasures	▪ RegistrationId ▪ ArrayOfDataFeederStartControlId	▪ ArrayOfDataFeederStatusReport
	getCollectedDFMeasures	▪ RegistrationId ▪ MaxMeasure ▪ TimeoutInMillis	▪ ArrayOfMeasure
resource	getCodeTranslations	▪ RegistrationId ▪ String ▪ String ▪ String ▪ ArrayOfErrorDesignation	▪ ArrayOfstring
	declareResource	▪ RegistrationId ▪ String ▪ BigInteger	▪ ResourceId
	uploadResourceContent	▪ RegistrationId ▪ ResourceId ▪ byte[]	▪ void
	uploadResource	▪ RegistrationId ▪ ResourceId ▪ String	▪ Void
	checkResourceUpload	▪ RegistrationId ▪ ResourceId	▪ UploadReport
configuration	acceptConfiguration	▪ RegistrationId ▪ ResourceId	▪ Void
	deployConfiguration	▪ RegistrationId ▪ ResourceId	▪ Void

	getMeasureDeliveryPolicy	▪ RegistrationId	▪ MeasureDeliveryPolicy

Table 1 SQM Simulator: available Services and Operations

Description of the argument format per datatype

The following table sums up the operations argument's format.

Argument datatype	Format
ClientId	`<ClientId name="ClientTest" />`
RegistrationId	`<RegistrationId registrationId="1e940b:fffe022d24:-8000:client1" version="v1_0"/>`

ArrayOfDataFeederId	`<ArrayOfDataFeederId>`
	`  <DataFeederId>`
	`  <DataFeederDefinitionId dfdName="Sql_MMediaDF" dfdVersion="v1_0" />`
	`  <ArrayOfPropertyValue>`
	`  <PropertyValue name="DbaseLocation" value="TV4" />`
	`  <PropertyValue name="Id" value="TV4" />`
	`  </ArrayOfPropertyValue>`
	`  </DataFeederId>`
	`</ArrayOfDataFeederId>`

ArrayOfDataFeederStartControlId	`<ArrayOfDataFeederStartControlId>`
	`  <DataFeederStartControlId transactionId="1" measureId="2004-11-09T09:20:18.211">`
	`  <DataFeederId>`
	`  <DataFeederDefinitionId dfdName="OvisHTTP" dfdVersion="v1_0" />`
	`  <ArrayOfPropertyValue>`
	`  <PropertyValue name="HOST" value="honda_1" />`
	`  <PropertyValue name="PROBENAME" value="honda" />`
	`  <PropertyValue name="SYSTEM" value="HP-UX" />`
	`  <PropertyValue name="TARGET" value="TARGET" />`
	`  </ArrayOfPropertyValue>`
	`  </DataFeederId>`
	`  </DataFeederStartControlId>`
	`</ArrayOfDataFeederStartControlId>`

ArrayOfDataFeederControlId	`<ArrayOfDataFeederControlId>`
	`  <DataFeederControlId transactionId="1">`
	`  <DataFeederId>`
	`  <DataFeederDefinitionId dfdName="OvisHTTP" dfdVersion="v1_0" />`
	`  <ArrayOfPropertyValue>`
	`  <PropertyValue name="HOST" value="honda_1" />`
	`  <PropertyValue name="PROBENAME" value="honda" />`
	`  <PropertyValue name="SYSTEM" value="HP-UX" />`
	`  <PropertyValue name="TARGET" value="TARGET" />`
	`  </ArrayOfPropertyValue>`
	`  </DataFeederId>`
	`  </DataFeederControlId>`
	`</ArrayOfDataFeederControlId>`

MaxMeasure	`<MaxMeasure maxMeasureCount="5">`
TimeoutInMillis	`<TimeoutInMillis timeout="10000"/>`

Table 2 SQM Simulator input arguments format

SQM Simulator Output (on the standard output)

Values replied by the service encapsulated within a XML document

SQM Simulator Output (on the error output)

Errors displayed in system's default language (local)

3.1.2.2 Examples

This section provides some examples of SQMSimulator commands.

Registration

For performing a registration, execute the command:

```
# SQMSimulator.sh -service registration -operation register -
arguments '<Arguments><ClientId name="client1"/></Arguments>'
```

Output:

```
<Arguments>
  <RegistrationId registrationId="166c114:1002d1d1ff1:-
7fff:client1" version="v1_0"/>
</Arguments>
```

Note

It is not possible to perform multiple registrations with the same client identifier, an RegistrationException is raised. For performing another registration with the same client identifier, use the deregister operation

Deregistration

For performing an unregistration, execute the command:

```
# SQMSimulator.sh -service registration -operation deregister -
arguments '<Arguments><ClientId name="client1"/></Arguments>'
```

DFD Discovery

For discovering the Datafeefer definitions supported by the Sample SA, execute the command:

```
# SQMSimulator.sh -service discovery -operation getDFDs -
arguments '<Arguments><RegistrationId
registrationId="166c114:1002d1d1ff1:-7fff:client1"
version="v1_0"/> </Arguments>'
```

Output (partial):

```
<Arguments>
  <ArrayOfDataFeederDefinition>
    <DataFeederDefinition>
      <DataFeederDefinitionId dfdName="Sql_MMediaDF"
dfdVersion="v1_0" />
      <ArrayOfPropertyDefinition>
```

```
        <PropertyDefinition name="DbaseLocation"
label="Database Location" dataType="STRING">
          <description>Database Location</description>
        </PropertyDefinition>
        <PropertyDefinition name="Id" label="Identificator"
dataType="INT">
        </PropertyDefinition>
      </ArrayOfPropertyDefinition>
      <ArrayOfParameterDefinition>
        <ParameterDefinition name="EndTime" label="End Movie
Timestamp" isCustomerDependent="true" category="OTHER"
dataType="ABSOLUTE_TIME" partition="OTHER">
          <description>End Movie Timestamp</description>
        </ParameterDefinition>
        <ParameterDefinition name="StartTime" label="Start
Movie Timestamp" isCustomerDependent="true" category="OTHER"
dataType="ABSOLUTE_TIME" partition="OTHER">
          <description>Start Movies Timestamp</description>
        </ParameterDefinition>
        <ParameterDefinition name="NbMovies" label="Number of
downloaded movies" units="movies" isCustomerDependent="true"
category="OTHER" dataType="INT" partition="OTHER">
          <description>Start Movies Timestamp</description>
        </ParameterDefinition>
      </ArrayOfParameterDefinition>
    </DataFeederDefinition>
    . . .
  </ArrayOfDataFeederDefinition>
</Arguments>
```

DFI Discovery

For discovering the Datafeefer instances handled by the Sample SA, execute the command:

```
# SQMSimulator.sh -service discovery -operation getDFs -
arguments '<Arguments><RegistrationId
registrationId="166c114:1002d1d1ff1:-7fff:client1"
version="v1_0"/> </Arguments>'
```

Output (partial):

```
<Arguments>
  <ArrayOfDataFeeder>
    <DataFeeder>
      <DataFeederId>
        <DataFeederDefinitionId dfdName="TeMIPFaultStats"
dfdVersion="v1_0" />
        <ArrayOfPropertyValue>
          <PropertyValue name="EntityName" value="BTS
BTS_Paris_1" />
          <PropertyValue name="OcName" value="OP_IDF" />
          <PropertyValue name="DomainName" value="IDF" />
        </ArrayOfPropertyValue>
      </DataFeederId>
      <ArrayOfPropertyValue>
        <PropertyValue name="EntityScope" value="*" />
      </ArrayOfPropertyValue>
    </DataFeeder>
    <DataFeeder>
      <DataFeederId>
        <DataFeederDefinitionId dfdName="Sql_MMediaDF"
dfdVersion="v1_0" />
        <ArrayOfPropertyValue>
```

```
          <PropertyValue name="DbaseLocation" value="TV_1" />
          <PropertyValue name="Id" value="TV_1" />
        </ArrayOfPropertyValue>
      </DataFeederId>
    </DataFeederId>
    ……..
  </ArrayOfDataFeeder>
</Arguments>
```

Enable Measures collection

For requesting to the SA, collecting Measures of a given set of the Datafeeder instances, execute the commands:

1. Specify the operation arguments in xml file:

```
<Arguments>
  <RegistrationId registrationId="166c114:1002d1d1ff1:-
7fff:client1" version="v1_0"/>
  <ArrayOfDataFeederStartControlId>
    <DataFeederStartControlId transactionId="1">
      <DataFeederId>
        <DataFeederDefinitionId dfdName="OvisHTTP"
dfdVersion="v1_0" />
        <ArrayOfPropertyValue>
          <PropertyValue name="HOST" value="honda_1" />
          <PropertyValue name="PROBENAME" value="honda" />
          <PropertyValue name="SYSTEM" value="HP-UX" />
          <PropertyValue name="TARGET" value="TARGET" />
        </ArrayOfPropertyValue>
      </DataFeederId>
    </DataFeederStartControlId>
    <DataFeederStartControlId transactionId="1">
       <DataFeederId>
        <DataFeederDefinitionId dfdName="TeMIPFaultStats"
dfdVersion="v1_0" />
        <ArrayOfPropertyValue>
          <PropertyValue name="EntityName" value="BTS
BTS_Paris_1" />
          <PropertyValue name="OcName" value="OP_IDF" />
          <PropertyValue name="DomainName" value="IDF" />
        </ArrayOfPropertyValue>
      </DataFeederId>
    </DataFeederStartControlId>
  </ArrayOfDataFeederStartControlId>
</Arguments>
```

2. Run the SQMSimulator passing in argument this xml file

```
# SQMSimulator.sh -service measure -operation
collectSinceDFMeasures –argumentsFile <xml file>
```

Output (status of the collection on the requested DFIs)

```
<Arguments>
  <ArrayOfDataFeederStatusReport>
    <DataFeederStatusReport timestamp="2005-03-
25T16:45:39.125">
      <DataFeederId>
        <DataFeederDefinitionId dfdName="OvisHTTP"
dfdVersion="v1_0" />
        <ArrayOfPropertyValue>
          <PropertyValue name="HOST" value="honda_1" />
          <PropertyValue name="PROBENAME" value="honda" />
          <PropertyValue name="SYSTEM" value="HP-UX" />
          <PropertyValue name="TARGET" value="TARGET" />
```

```
        </ArrayOfPropertyValue>
      </DataFeederId>
      <DataFeederStatus
explanation="Active"availabilityStatus="AVAILABLE"administrativ
eState="UNLOCKED"/>
    </DataFeederStatusReport>
    <DataFeederStatusReport timestamp="2005-03-
25T16:45:39.132">
      <DataFeederId>
        <DataFeederDefinitionId dfdName="TeMIPFaultStats"
dfdVersion="v1_0" />
        <ArrayOfPropertyValue>
          <PropertyValue name="EntityName" value="BTS
BTS_Paris_1" />
          <PropertyValue name="OcName" value="OP_IDF" />
          <PropertyValue name="DomainName" value="IDF" />
        </ArrayOfPropertyValue>
      </DataFeederId>
      <DataFeederStatus
explanation="Active"availabilityStatus="AVAILABLE"administrativ
eState="UNLOCKED"/>
    </DataFeederStatusReport>
  </ArrayOfDataFeederStatusReport>
</Arguments>
```

Collection of Measures

For requesting to the SA the available Measures (here we request a maximum of 2 measures), execute the commands:

1. Specify the following operation arguments in xml file:

```
<Arguments>
       <RegistrationId registrationId="166c114:1002d1d1ff1:-
7fff:client1" version="v1_0"/>
       <MaxMeasure maxMeasureCount="2"/>
       <TimeoutInMillis timeout="10000"/>
</Arguments>
```

2. Run the SQMSimulator passing in argument this xml file

```
# SQMSimulator.sh -service measure -operation
getCollectedDFMeasures –argumentsFile <xml file>
```

Stopping collection

For requesting to the SA stopping Measures on a given set of DFIs, execute the commands:

1. Specify the operation arguments in xml file:

```
<Arguments>
  <RegistrationId registrationId="166c114:1002d1d1ff1:-
7fff:client1" version="v1_0"/>
  <ArrayOfDataFeederControlId>
    <DataFeederControlId transactionId="1">
      <DataFeederId>
        <DataFeederDefinitionId dfdName="OvisHTTP"
dfdVersion="v1_0" />
        <ArrayOfPropertyValue>
          <PropertyValue name="HOST" value="honda_1" />
          <PropertyValue name="PROBENAME" value="honda" />
          <PropertyValue name="SYSTEM" value="HP-UX" />
          <PropertyValue name="TARGET" value="TARGET" />
        </ArrayOfPropertyValue>
      </DataFeederId>
    </DataFeederControlId>
```

```
    <DataFeederControlId transactionId="1">
       <DataFeederId>
        <DataFeederDefinitionId dfdName="TeMIPFaultStats"
dfdVersion="v1_0" />
        <ArrayOfPropertyValue>
          <PropertyValue name="EntityName" value="BTS
BTS_Paris_1" />
          <PropertyValue name="OcName" value="OP_IDF" />
          <PropertyValue name="DomainName" value="IDF" />
        </ArrayOfPropertyValue>
      </DataFeederId>
    </DataFeederControlId>
  </ArrayOfDataFeederControlId>
</Arguments>
```

2. Run the SQMSimulator passing in argument this xml file

```
# SQMSimulator.sh -service measure -operation stopDFCollections
-argumentsFile <xml file>
```

Output (status of the collection on the requested DFI)

```
<Arguments>
  <ArrayOfDataFeederStatusReport>
    <DataFeederStatusReport timestamp="2005-03-
25T17:01:11.615">
      <DataFeederId>
        <DataFeederDefinitionId dfdName="OvisHTTP"
dfdVersion="v1_0" />
        <ArrayOfPropertyValue>
          <PropertyValue name="HOST" value="honda_1" />
          <PropertyValue name="PROBENAME" value="honda" />
          <PropertyValue name="SYSTEM" value="HP-UX" />
          <PropertyValue name="TARGET" value="TARGET" />
        </ArrayOfPropertyValue>
      </DataFeederId>
      <DataFeederStatus
explanation="Stopped"availabilityStatus="AVAILABLE"administrati
veState="LOCKED"/>
    </DataFeederStatusReport>
    <DataFeederStatusReport timestamp="2005-03-
25T17:01:11.615">
      <DataFeederId>
        <DataFeederDefinitionId dfdName="TeMIPFaultStats"
dfdVersion="v1_0" />
        <ArrayOfPropertyValue>
          <PropertyValue name="EntityName" value="BTS
BTS_Paris_1" />
          <PropertyValue name="OcName" value="OP_IDF" />
          <PropertyValue name="DomainName" value="IDF" />
        </ArrayOfPropertyValue>
      </DataFeederId>
      <DataFeederStatus
explanation="Stopped"availabilityStatus="FAILED"administrativeS
tate="LOCKED"/>
    </DataFeederStatusReport>
  </ArrayOfDataFeederStatusReport>
</Arguments>
```

Repair collection

For requesting to the SA repairing Measures on a given set of DFIs, execute the commands:

1. Specify the operation arguments in xml file:

```
<Arguments>
  <RegistrationId registrationId="166c114:1002d1d1ff1:-
7fff:client1" version="v1_0"/>
  <ArrayOfDataFeederControlId>
    <DataFeederControlId transactionId="1">
      <DataFeederId>
        <DataFeederDefinitionId dfdName="OvisHTTP"
dfdVersion="v1_0" />
        <ArrayOfPropertyValue>
          <PropertyValue name="HOST" value="honda_1" />
          <PropertyValue name="PROBENAME" value="honda" />
          <PropertyValue name="SYSTEM" value="HP-UX" />
          <PropertyValue name="TARGET" value="TARGET" />
        </ArrayOfPropertyValue>
      </DataFeederId>
    </DataFeederControlId>
    <DataFeederControlId transactionId="1">
       <DataFeederId>
        <DataFeederDefinitionId dfdName="TeMIPFaultStats"
dfdVersion="v1_0" />
        <ArrayOfPropertyValue>
          <PropertyValue name="EntityName" value="BTS
BTS_Paris_1" />
          <PropertyValue name="OcName" value="OP_IDF" />
          <PropertyValue name="DomainName" value="IDF" />
        </ArrayOfPropertyValue>
      </DataFeederId>
    </DataFeederControlId>
  </ArrayOfDataFeederControlId>
</Arguments>
```

2. Run the SQMSimulator passing in argument this xml file

```
# SQMSimulator.sh -service measure -operation
repairDFCollections –argumentsFile <xml file>
```

Output (status of the collection on the requested DFI)

```
<Arguments>
  <ArrayOfDataFeederStatusReport>
    <DataFeederStatusReport timestamp="2005-03-
25T17:07:18.748">
      <DataFeederId>
        <DataFeederDefinitionId dfdName="OvisHTTP"
dfdVersion="v1_0" />
        <ArrayOfPropertyValue>
          <PropertyValue name="HOST" value="honda_1" />
          <PropertyValue name="PROBENAME" value="honda" />
          <PropertyValue name="SYSTEM" value="HP-UX" />
          <PropertyValue name="TARGET" value="TARGET" />
        </ArrayOfPropertyValue>
      </DataFeederId>
      <DataFeederStatus
explanation="Down"availabilityStatus="FAILED"administrativeStat
e="LOCKED"/>
    </DataFeederStatusReport>
    <DataFeederStatusReport timestamp="2005-03-
25T17:07:18.748">
      <DataFeederId>
        <DataFeederDefinitionId dfdName="TeMIPFaultStats"
dfdVersion="v1_0" />
        <ArrayOfPropertyValue>
```

```
          <PropertyValue name="EntityName" value="BTS
BTS_Paris_1" />
          <PropertyValue name="OcName" value="OP_IDF" />
          <PropertyValue name="DomainName" value="IDF" />
        </ArrayOfPropertyValue>
      </DataFeederId>
      <DataFeederStatus
explanation="Down"availabilityStatus="FAILED"administrativeStat
e="LOCKED"/>
    </DataFeederStatusReport>
  </ArrayOfDataFeederStatusReport>

</Arguments>
```

3.1.2.3 Configuration

The SQMSimulator configuration is defined in the property file located in:

$TEMIP_SC_HOME/ServiceAdapters/SDK/v1_0/SQMSimulator/properties/SQMSimulator.properties

The following table sums up the available configuration options:

Property name	Default value	Description
▪ Services.hostname	▪ localhost	Hostname of the servlet container
▪ Services.portnumber	▪ 8080	portnumber of the servlet container
▪ ServiceAdaptor.name	▪ SampleSA_v1_0	Name of the Service Adapter on which service requests are sent
▪ logfiles.location	▪ /var/opt/SA_SDK/logs	Log and Trace Directory location
▪ logfiles.tracefile.name	▪ SQMSimulator_Trace.log	Name of the trace file
▪ logfiles.logfile.name	▪ SQMSimulator_Log.log	Name of the log file
▪ logs.logs.level	▪ ALL	The logging levels in descending order are: ▪ SEVERE (highest value) ▪ WARNING ▪ INFO ▪ CONFIG (lowest value) In addition there is a level OFF that can be used to turn off logging, and a level ALL that can be used to enable logging of all messages.
▪ logs.traces.level	▪ OFF	The tracing levels in descending order are: ▪ FINE (highest value) ▪ FINER ▪ FINEST (lowest value) In addition there is a level OFF that can be used to turn off logging, and a level ALL that can be used to enable logging of all messages.

Table 3 SQMSimulator configuration properties

3.1.3 SDK API Documentation

The SDK API document is available in HTML format in the SDK kit at the following location:

$TEMIP_SC_HOME/ServiceAdapters/SDK/v1_0/doc/index.html

Chapter 4
Service Adapter development guidelines

This chapter presents some advices for developing an SQM Service Adapter. It introduces the following development points:

- Defining a development environment
- Compiling an SQM Service Adapter
- Installing a SQM Service Adapter in a Servlet engine
- Deploying a SQM Service Adapter in a Servlet engine
- Testing a SQM Service Adapter

This development guidelines are based on the development environment of the Sample Service Adapter provided in the software kit. The most commonly-used development tasks are available in the Sample SA Ant file.

4.1 Defining the development environment

To setup the Service Adapter development environment which will contain your Service Adapter implementation source code:

1. Create you working directory at your favorite location. For instance:
   ```
   # mkdir ~/SQM/MySA
   # cd ~/SQM/MySA
   ```
2. Set the environment variable TEMIP_SA_SDK_HOME to the SA SDK software product directory location:
   ```
   # export TEMIP_SA_SDK_HOME=$TEMIP_SC_HOME/ServiceAdapters/SDK/v1_0
   ```
3. Copy the Sample SA Ant file your working directory
   `# cp $TEMIP_SA_SDK_HOME/SampleSA/build.xml.`
   Create the development environment by call the target ‘create’ of the Ant file. Running this target, the Service Adapter name (<SA name>) will be requested.
   `# ant create`
   This target initialize the environment by creating in the working directory:
   a. A directory named ‘src’. This directory will contain all SA source files
   b. A directory name ‘src/properties’. This directory will contain SA properties files, the <SA name>.properties file defining SA environment at run-time and a property file <SA name>_Version.properties defining the SA version properties
   c. It generates the SA interface classes from the SDK WSDL and moves the implementation template classes in the directory named ‘src’.
   This generation is performed by the Ant target named ‘wsdl’:

 # ant wsdl
This target generates the SA interface classes from the SDK WSDL using the Apache Axis tool WSDL2Java. This target generates:

i. An implementation template class per binding. It is intended that **the Service Adapter writer fill out the implementation from these templates**. These templates classes are located in:
`src/com/compaq/temip/servicecenter/serviceadapter/sdk/soap/service/ConfigurationServiceSOAPImpl.java`
`src/com/compaq/temip/servicecenter/serviceadapter/sdk/soap/service/DiscoveryServiceSOAPImpl.java`
`src/com/compaq/temip/servicecenter/serviceadapter/sdk/soap/service/MeasureServiceSOAPImpl.java`
`src/com/compaq/temip/servicecenter/serviceadapter/sdk/soap/service/RegistrationServiceSOAPImpl.java`
`src/com/compaq/temip/servicecenter/serviceadapter/sdk/soap/service/ResourceServiceSOAPImpl.java`

ii. For all services, one deploy.wsdd file located in:
`soapsrc/com/compaq/temip/servicecenter/serviceadapter/sdk/soap/service/`

iii. For all service, one undeploy.wsdd file, located in:
`soapsrc/com/compaq/temip/servicecenter/serviceadapter/sdk/soap/service/`

4. Edit the build.xml file, to update the value of the property named ‘comp.name’. Set the property value to the SA name specified at the previous step.
`<property name="comp.name" value="<put here the SA name>"/>`
5. The Service Adapter is ready to compile. Even if the template classes do nothing, the templates can be used for test purpose

4.2 Compiling an SQM Service Adapter

Before compiling the Service Adapter:

1. go to you SA working directory. For instance:
`# cd ~/SQM/MySA`
2. Set the environment variable TEMIP_SA_SDK_HOME to the SA SDK software product directory location:
`# export TEMIP_SA_SDK_HOME=$TEMIP_SC_HOME/ServiceAdapters/SDK/v1_0`

To compile the Service Adapter, the Ant file provides the following targets:

- Build [default target]: it generates a SA kit by executing: SA interface classes generation, compilation, war file generation, SA kit
The SA Kit is a zip file containing source files and compilation result. It does not package AXIS libraries.
- Compile: generates the skeleton classes and compile the java source code
- War: generates the SA interface classes and compile the java source code, and generates the War file which can be installed on the Servlet container
- Clean: remove the result of the compilation: deletes directories ‘build’ and ‘soapsrc’

The result of these compilation steps is generated in the directory named ‘build’. The resulting Service Adapter kit content is under ‘build/input’.

4.3 Installing a SQM Service Adapter

The Service Adapter must be installed in a Servlet Container. Installation depends on the selected Servlet Container. In most cases it consists in dropping the SA war file in the webapps directory of the server and restarting the server, or by using a server-specific mechanism to enable the web application.

For instance, installing the Sample Service Adapter on TomCat 4.1 consists in:

1. Log as TomCat admin user
2. Unzip the SA kit at the same location of the SA SDK
   ```
   # unzip build/MySA_v1_0.zip -d $TEMIP_SC_HOME
   ```
3. Copy the war file in the TomCat webapps directory:
   ```
   # cp $TEMIP_SC_HOME/ServiceAdapters/SDK/v1_0/<SA
   Name>/war/<SA Name>_v1_0.war $CATALINA_HOME/webapps
   ```
4. Stop TomCat server:
   ```
   # $CATALINA_HOME/bin/shutdown.sh
   ```
5. Start TomCat server:
   ```
   # $CATALINA_HOME/bin/startup.sh
   ```

4.4 Deploying a SQM Service Adapter

Deploying the SQM Service adapter consists in telling Axis how to expose these services. Axis takes a Web Service Deployment Descriptor (WSDD) file that describes in XML what the services are, what methods it exports and other aspects of the SOAP endpoint.

Execute the following command to perform the deployment:

```
# $TEMIP_SC_HOME/ServiceAdapters/SDK/v1_0/<SA Name>
/bin/deploy.sh
```

This command deploys the Service Adapter services on a local Web server configure with the http port number 8080.

4.5 Testing a SQM Service Adapter

To test the installation and the deployment of the Service adapter, refer to chapter 3.1.1.5, it's the same method as for the Sample SA.

The SQMSimulator tool, described in chapter 3.1.1.6, can be used for testing the SA services.

Chapter 5

Implementing a Service Adapter

This chapter is intended to present:

- How to design a Service Adapter
- How to implement a Service Adaper, the mandatory service operations
- How to implement each service operations

It includes the following topics:

5.1 Designing a Service Adapter

Designing a Service Adapter consists in designing a server implementing the services and operations defined by the SQM Service Adapter Interface. The interface defines the services and operations described in the table thereafter, some operations are mandatory some others are optional.

Service name	Operation name	Mandatory Implementation
Registration	register	♦
	deregister	♦
Discovery	getDFs	
	getDFDs	
Measure	repairDFCollections	♦
	stopDFCollections	♦
	getDFsInternals	
	collectSinceDFMeasures	♦
	getCollectedDFMeasures	♦
Resource	getCodeTranslations	
	declareResource	
	uploadResourceContent	
	uploadResource	
	checkResourceUpload	
Configuration	acceptConfiguration	
	deployConfiguration	
	getMeasureDeliveryPolicy	♦

The starting point is the implementation classes of each service. SQM Service Adapter Interface is defined as a Web Services. Apache Axis allows generating one template implementation class per service. This template doesn't do anything. It is intended that the service writer fill out the implementation from this template.

For instance the template class for the registration service is the following:

```
package com.compaq.temip.servicecenter.serviceadapter.sdk.soap.service;

public class RegistrationServiceSOAPImpl implements
com.compaq.temip.servicecenter.serviceadapter.sdk.soap.service.Registra
tionService{
    public
com.compaq.temip.servicecenter.serviceadapter.sdk.soap.RegistrationId
register(java.lang.String clientId) throws java.rmi.RemoteException,
com.compaq.temip.servicecenter.serviceadapter.sdk.soap.RegistrationExce
ption {
        return null;
    }

    public void deregister(java.lang.String clientId) throws
java.rmi.RemoteException,
com.compaq.temip.servicecenter.serviceadapter.sdk.soap.RegistrationExce
ption {
    }
}
```

The implementation classes are:

- ConfigurationServiceSOAPImpl
- DiscoveryServiceSOAPImpl
- MeasureServiceSOAPImpl
- RegistrationServiceSOAPImpl
- ResourceServiceSOAPImpl

It is not recommended to implement Service Adapter services in these classes but to delegate to other components.

The next sub-headings provides a design proposal for each mandatory Service Adapter Services (Registration, Measure and Configuration)

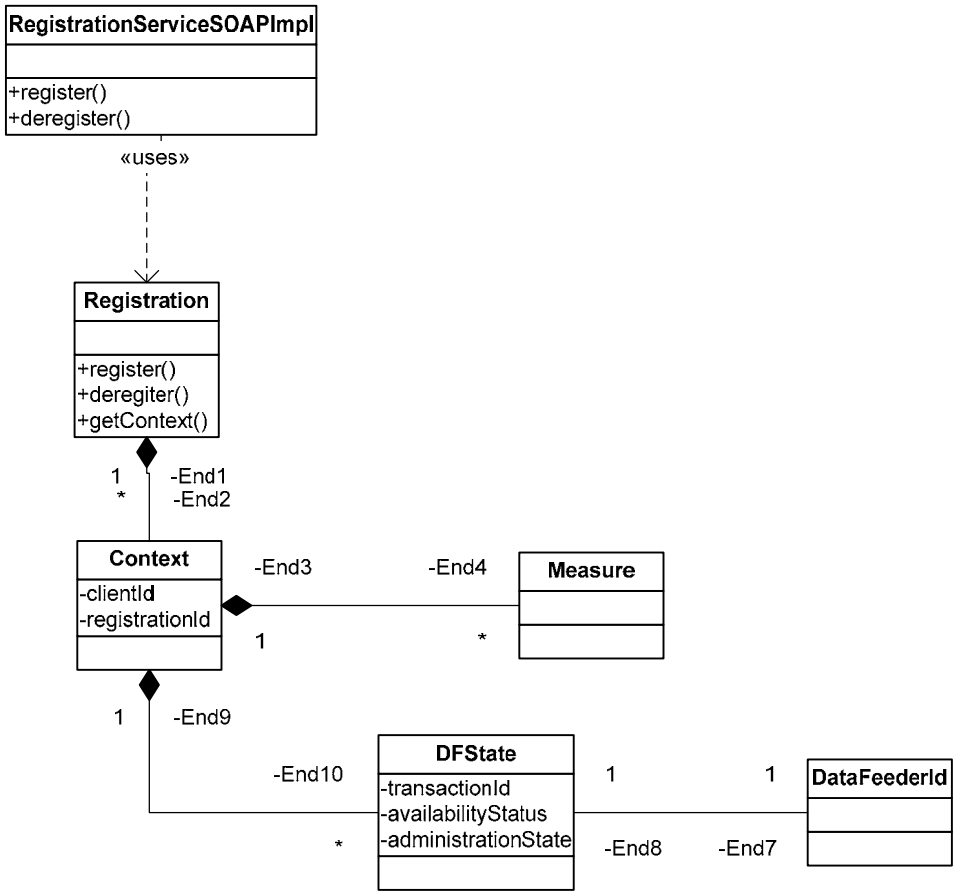
5.1.1 Registration Service

The main purpose of this service is to manage a context per service client. Even if the Service Adapter will be access by a single client (a single SA Proxy Application), the SQM platform might have, mainly for scalability reasons, several clients (instances) accessing one single Service Adapter.

The registration service must manage a single context per client. Each context handles:

- the pair client identifier/registration identifier
- A queue of measures. This queue contains all the measures collected by the Service Adapter which must be published to the client

- Optionally the state of each Data Feeder instance.
- Optionally of Configuration context



5.1.1.1 Operation register

This operation is in charge of creating a client context and associating an identifier (registrationId) to this client context. This identifier must be unique.

It is also recommended to synchronize this operation. Indeed each service call is performed in a dedicated thread and a service can be called by multiple clients.

If the client is already registered the operation must throw a Registration exception passing the reason code " invalidated_previous_registration_id_client_already_registered" . The operation must also invalidate the client context but should not delete it. The client is responsible for context deletion by calling the ' deregister' operation.

Implementation example

```
public RegistrationId register(String clientId) throws
RegistrationException, RemoteException {
synchronized (m_lock) {

   RegistrationId registrationId;
   if (clientId != null) {
          if ( isRegistred(clientId)) {
                 // Client already register
                 // invalidate the context
```

```
                registrationId =
                       (RegistrationId)m_client2regId.get(clientId);
                Context context =
                       (Context) m_reg2context.get(registrationId);
                context.setInvalid();

                // Construct and throw a registration exception
                ErrorDesignation errDesign = new ErrorDesignation(
                       new ArrayOfstring(new String[] {clientId}),
" invalidated_previous_registration_id_client_already_registered" );

                throw new RegistrationException(
                       "Client already registered: "+clientId,
                       errDesign);
          } else {
                // Client not already registered
                // generates a unique registration identifier
          UID uId = new UID();
                String stringUId = new String(uId.toString()+":"+clientId);

                // Allocation of the returned Registration identifier
                registrationId=new RegistrationId();

                // Sets the SA version in the returned registration ID
                registrationId.setVersion("v1_0");

                // Sets the unique identifier as registration ID
                registrationId.setRegistrationId(stringUId);

                // Stores the tuple (clientId,registrationId)
                // and allocates a context for this client
                m_client2regId.put(clientId, registrationId);
                Context context = new Context(registrationId, clientId);
                m_reg2context.put(registrationId, context);
          }
    } else {
          // provided client identifier is null
          // throw a registration exception
          ErrorDesignation errDesign = new ErrorDesignation(
                null,
                "invalid_client_id");

          throw new RegistrationException(
                "Invalid null client identifier",
                errDesign);
     }

     return registrationId;
  }
}
```

5.1.1.2 Operation deregister

This operation deregisters the client identified by it client ID. This operation must also perform a cleanup of all resources associated to the given client, such as:

- Deleting the associated context,
- Stopping the components collecting the measure,
- Destroying the queue of measures ….

It is also recommended to synchronize this operation.

If the given client is not registered, the operation must throw a registration exception passing the error code “ client_not_registered”

Implementation example

```
public void deregister(String clientId)
          throws RegistrationException, RemoteException {
   // Synchronize the operation
   synchronized (m_lock){
          RegistrationId registrationId;

            if ( isRegistred(clientId) ) {
                 // Valid and registered client identifier

                 // Get the client context and destroy it
                registrationId =
                       (RegistrationId)m_client2regId.get(clientId);
                Context context
                           = (Context) m_reg2context.get(registrationId);
                context.destroy();

                m_reg2context.remove(registrationId);
                m_client2regId.remove(clientId);
            } else {
                 // Client is not registered
                 // throw a Registration Exception
                ErrorDesignation errDesign
                    = new ErrorDesignation(
                        new ArrayOfstring(new String[] {clientId}),
                        “client_not_registered”);
                throw new RegistrationException(
                        “Client {0} is not registered”,
                        errDesign);
            }
        }
    }
```

5.1.2 Configuration Service

This service is in charge of managing the Service Adapter configuration. It implements operations to dynamically and remotely deploy the SA configuration.

For a simple implementation of Service Adapter, the configuration can be managed locally and so it is not necessary to implement these services.

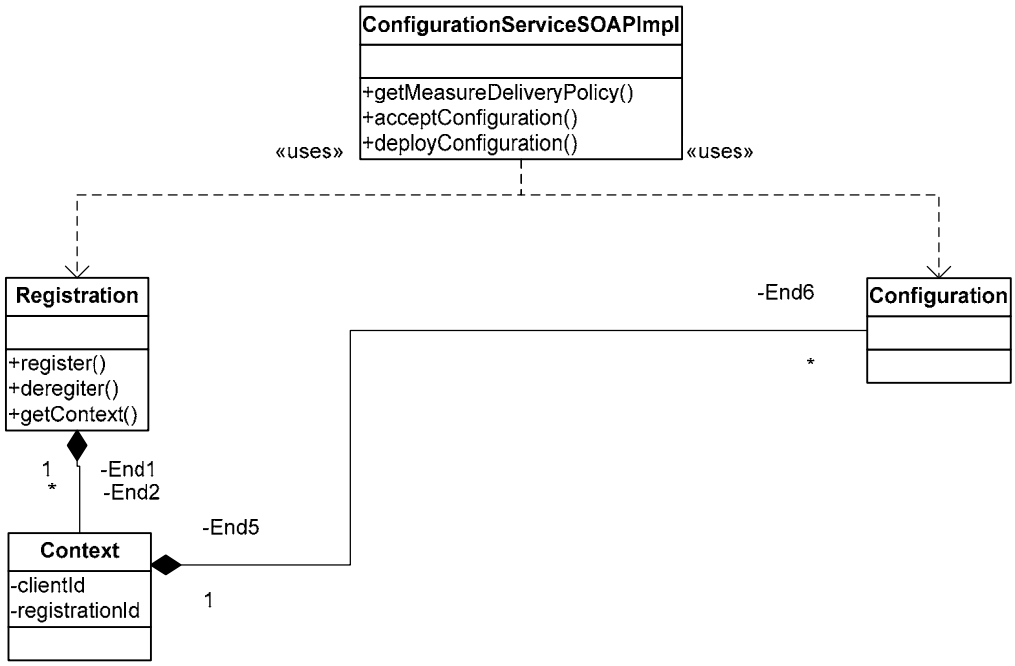
Note

SQM SDK client part (SA Proxy kit) does not yet provide configuration tools for managing remotely Service Adapter configuration.

A more complex Service Adapter implementation will implement all these services and should be able to handle one configuration per client (this is why the registration identifier is passed to the configuration service operations, the Context class can also manage a configuration per client).

The only mandatory operation of this service is the getMeasureDeliveryPolicy operation.

It is recommended to delegate the configuration implementation to a dedicated class and not to the ConfigurationServiceSOAPImpl class.



5.1.2.1 Operation getMeasureDeliveryPolicy

The operation returns the measure delivery policies (capabilities) used by this Service Adapter implementation. These policies indicate to the SA client if the SA supports resynchronization capabilities or not.

Policies are expressed by two flags:

- areEventsChronological: it determines if the service adapter generates measures in a chronological order.
- couldRetrieveHistoricalMeasuresUponTheirIds: it indicates that the Service Adapter implementation is able to retrieve all historical measures since a previously (in the past) collected measure whose MeasureId is provided by the SQM platform.

When one of these flags is set to false the SA client consider that the SA does not implement resynchronization capabilies.

The operation must throw a Registration Exception in the following cases:

- When the provided registration Id is unknown : error code " unkown_registration_id" .
- When the context has been invalidated (it occurs when the client tries to register multiple times): error code " invalidated_registration_id"

Implementation example

In the following example, the getMeasureDeliveryPolicy operation finds the configuration associated to the given client (Configuration instance is allocated at registration time) and returns the policy.

```
public class ConfigurationServiceSOAPImpl implements ConfigurationService{

   public MeasureDeliveryPolicy getMeasureDeliveryPolicy(RegistrationId
regId) throws RemoteException, ConfigurationException {
```

```
            // Get the context associated to the provided client id
            // and delegate call on the configuration instance
            return getContext(regId).
                   getConfiguration().
                         getMeasureDeliveryPolicy();
     }
…..
}

   private Context getContext(RegistrationId regId) throws
ConfigurationException {
        Context context = null;
        try {
            Registration reg = Registration.getInstance();
            // Get the context and validate registration ID
            context = reg.getContext(regId);
        } catch (RegistrationException e) {
            String trace = e.getErrorDefaultTranslation();
            throw new ConfigurationException(trace, e.getError());
        }
        return context;
     }
```

```
public class Configuration {

   public MeasureDeliveryPolicy getMeasureDeliveryPolicy()
                throws ConfigurationException {
          // returns Service Adapter delivery policy
          MeasureDeliveryPolicy policy = new MeasureDeliveryPolicy();
          policy.setCouldRetrieveHistoricalMeasuresUponTheirIds(true);
          policy.setAreEventsChronological(true);
          return m_measureDeliveryPolicy;
     }
…
}
```

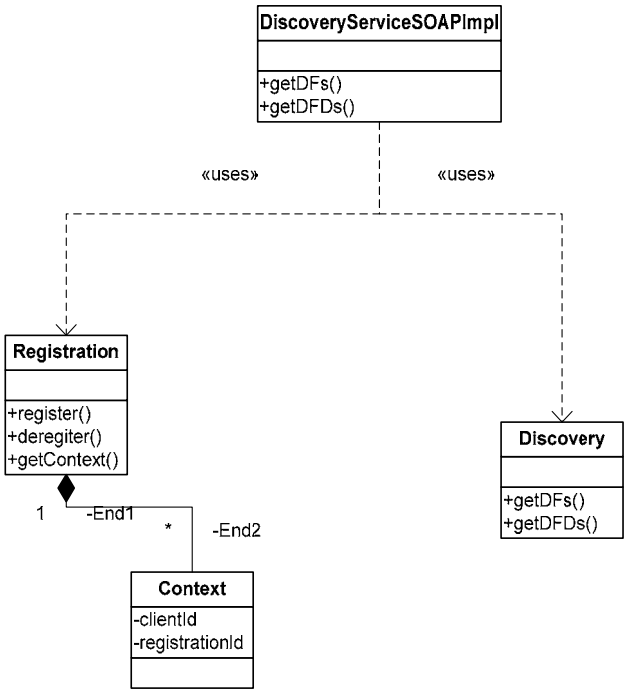
5.1.3 Discovery Service

The Discovery Service provides two operations for exporting into SQM the Data Feeder Model (Data Feeder Definitions) exposed by the Service Adapter and also the instances (Data Feeders) of this model:

- getDFDs: returns the Data Feeder Definitions exposed by the Service Adapter
- getDFs: returns the Data Feeders on which the Service Adapter is able to collect measures

Both operations are optional. Data Feeder Definition can be defined in SQM using the SQM Service Designer and Data Feeder imported using an external tool.

It is recommended to delegate the dicovery implementation to a dedicated class and not to the DiscoveryServiceSOAPImpl class.



5.1.3.1 Operation getDFDs

The operation returns, on each call, the requested number of Data Feeder Definitions that the Service Adapter implementation is aware of. In the current implementation of the SQM DFD Discovery tool, the number of requested definitions is set to the maximum integer value and discovery tool is designed to perform a single operation call.

The operation must throw a Registration Exception in the following cases:

- When the provided registration Id is unknown : error code " unkown_registration_id" .
- When the context has been invalidated (it occurs when the client tries to register multiple times): error code " invalidated_registration_id"

Implementation example

In the following example, the getDFs operation constructs a DataFeederDefinition and returns it. This DFD defines:

- 1 property part of MRP: SYSTEM
- 1 property: LOCATION
- 1 parameter: CPU

```
public class DiscoveryServiceSOAPImpl implements DiscoveryService{

   public ArrayOfDataFeederDefinition getDFDs(RegistrationId regId, int
maxDFDCount) throws java.rmi.RemoteException, DiscoveryException {
        return getContext(regId).getDiscovery().getDFDs(maxDFDCount);
    }

   private Context getContext(RegistrationId regId) throws
ConfigurationException {
        Context context = null;
        try {
```

```
            Registration reg = Registration.getInstance();
            // Get the context and validate registration ID
            context = reg.getContext(regId);
        } catch (RegistrationException e) {
            String trace = e.getErrorDefaultTranslation();
            throw new ConfigurationException(trace, e.getError());
        }
        return context;
    }
```

```
public class Discovery {

   public ArrayOfDataFeederDefinition getDFDs(int maxDFDCount)
          throws DiscoveryException {
          // for the time being the operation's argument 'maxDFDCount'
          // is not handled

          // Allocates and defined the Data Feeder Definition
          DataFeederDefinition dfd = new DataFeederDefinition();

          // Define the MRP properties
          PropertyDefinition system
                = new PropertyDefinition(DataType.STRING,
                                         "Description text",
                                         "SYSTEM",
                                         "SYSTEM");
          PropertyDefinition[] propertyDefsMRP =
                 new PropertyDefinition[] {system};
          dfd.setOrderedMRPPropertyDefinitions(
                 new ArrayOfPropertyDefinition(propertyDefsMRP));

          // Define the additional properties
          PropertyDefinition location
                = new PropertyDefinition(DataType.STRING,
                                         "Description text",
                                         "LOCATION",
                                         "LOCATION");
          PropertyDefinition[] propertyDefsAddition =
                 new PropertyDefinition[] {location};
          dfd.setAdditionalPropertyDefinitions(
                 new ArrayOfPropertyDefinition(propertyDefsAddition));

          // Define the DFD's parameters
          ParameterDefinition cpu
                = new ParameterDefinition(ParameterCategory.PERCENT,
                                          DataType.FLOAT,
                                          "CPU consumption (%)",
                                          false,
                                          "CPU Used",
                                          "CPU",
                                          null,
                                          null);
          ParameterDefinition[] parameterDefs = new ParameterDefinition[]
                      { cpu };
          dfd.setParameterDefinitions(
                 new ArrayOfParameterDefinition(parameterDefs));

          // Define the DFD identifier. A DFD is identified by its name and
          // and its version
            dfd.setDfdId(
                 new DataFeederDefinitionId("MYDFD","v1_0");
```

```
            // Set DFD label and description text
            dfd.setLabel("DFD label");
            dfd.setDescription("DFD Description");

            // Put the defined DFD in a ArrayOfDataFeederDefinition object
            DataFeederDefinition[] dfdArray=new DataFeederDefinition[1];
            dfdArray[0]=dfd;
            ArrayOfDataFeederDefinition dfds
                  = new ArrayOfDataFeederDefinition(dfdArray);

            // return the DFD
            return dfds;
      }
….
}
```

5.1.3.2 Operation getDFs

The operation returns, on each call, the requested number of the Data Feeders that the Service Adapter implementation is aware of. In the current implementation of the SQM DFD Discovery tool, the number of requested DFs is set to the maximum integer value and discovery tool is designed to perform a single operation call.

The operation must throw a Registration Exception in the following cases:

- When the provided registration Id is unknown: error code " unkown_registration_id" .
- When the context has been invalidated (it occurs when the client tries to register multiple times): error code " invalidated_registration_id"

Data Feeder object' s definition must be aligned with the corresponding Data Feeder Definition. And more especially the Data Feeder properties must be defined in the same order as in the DFD.

The getDFs operation provides also a parameter defining the scope of the DFs to return. This scope acts as a filter, to request only Data Feeders of a given set of DFD. In the current implementation of the SQM DFD Discovery tool, this scope is not used

Implementation example

In the following example, the getDFDs operation constructs and returns a single Data Feeder instance

```
public class DiscoveryServiceSOAPImpl implements DiscoveryService{

   public ArrayOfDataFeeder getDFs(RegistrationId regId,
                                   int maxDFCount,
                                   DataFeederDefinitionId optionalDFDScope)
         throws java.rmi.RemoteException, DiscoveryException {
       return getContext(regId).getDiscovery().getDFs(maxDFCount);
    }

   private Context getContext(RegistrationId regId) throws
ConfigurationException {
       Context context = null;
       try {
           Registration reg = Registration.getInstance();
           // Get the context and validate registration ID
           context = reg.getContext(regId);
       } catch (RegistrationException e) {
           String trace = e.getErrorDefaultTranslation();
```

```
                throw new ConfigurationException(trace, e.getError());
            }
            return context;
        }
```

```
    public class Discovery {

    public ArrayOfDataFeeder getDFs(int maxDFCount,
                                    DataFeederDefinitionId optionalDFDScope)
        throws DiscoveryException {
        // for the time being the operation's arguments 'maxDFDCount' and
        // 'optionalDFDScope' are not handled

        DataFeeder df = new DataFeeder();
        // Set the DF's identifier
        // A DF is identified by its DFD and by its MRP properties
        DataFeederId dfId = new DataFeederId();
        df.setDfId(dfId);

        // DataFeederId - fill MRP property values
        PropertyValue system
                    = new PropertyValue(
                            "SYSTEM",
                            "hars.vbe.cpqcorp.net");
        PropertyValue[] mrpPropertyValues = new PropertyValue[] {system};

        dfId.setOrderedMRPPropertyValues(
               new ArrayOfPropertyValue(mrpPropertyValues));

        // DataFeederId - fill DFD Id
        dfId.setDfdId(
               new DataFeederDefinitionId("MYDFD","v1_0"));

        // Set DF additional properties
        PropertyValue location
                    = new PropertyValue(
                            "LOCATION",
                            "Valbonne");
        PropertyValue[] additionalPropertyValues =
               new PropertyValue[] {location};

        df.setAddtionalPropertyValues(
               new ArrayOfPropertyValue(additionalPropertyValues));

        // Put the DF into an ArrayOfDataFeeder object
        DataFeeder [] dfArray=new DataFeeder [1];
        dfArray[0]=df;
        ArrayOfDataFeeder dfs = new ArrayOfDataFeeder (dfArray);

        // return the DF
        return dfs;
    }
```

5.1.4 Measure Service

The Measure service provides the following types of operations:

1. A single operation for collecting measures:
    - getCollectedDFMeasures: collects available measures and collection status

2.Operations to control the measure collection flow:

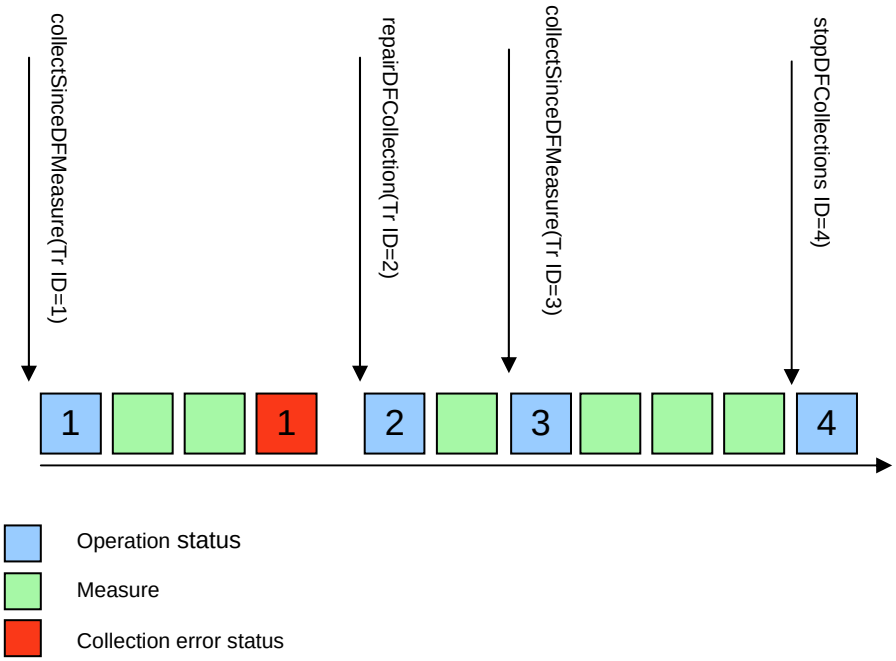
- Starting the collection on a set of Data Feeder => operation collectSinceDFMeasures
- Restoring a collection after failure => operation repairDFCollections
- Stopping a collection on a set of Data Feeder => stopDFCollections

3.Debugging operation

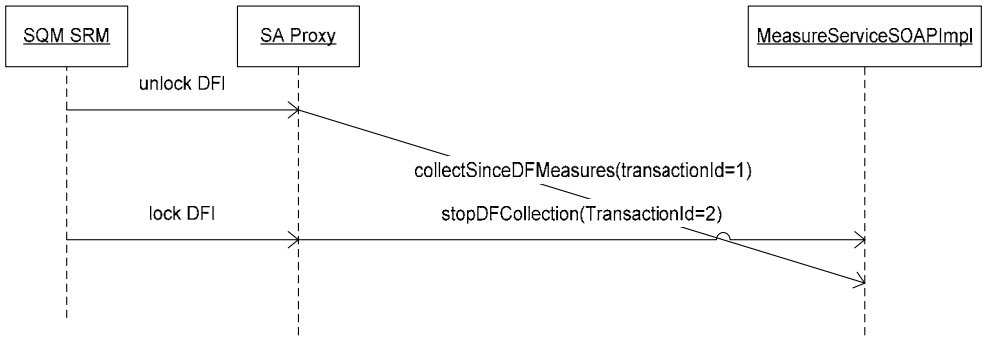
- Dump internal information of on the requested Data Feeders

Each of these control operations provides a Transaction ID. This Transaction ID is passed by the SA client and must be put in a status report published as a measure in the measure stream. These status reports are used by the SA Client (SA Proxy) to control the flow of measures and manage the state of each collection. There is a Transaction ID per Data Feeder, to control each Data Feeder collection independently. It is recommended to store at each control operation the Transaction ID. Indeed when a collection error occurs, the SA needs to publish a error status. This error status must contain the last Transaction Id.

The figure thereafter illustrates the collection flow.



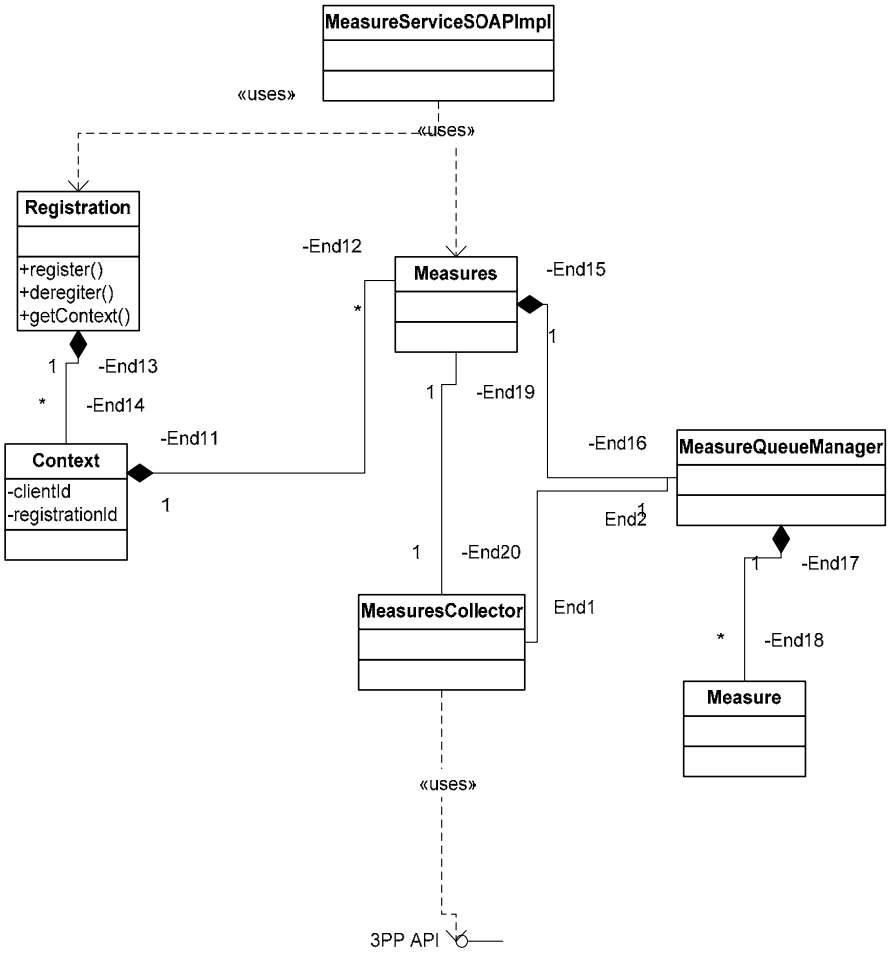
It is recommended to validate the TransactionId of the incoming operations by checking that the operation' s Transaction Id is greater that the previously received Transaction Id (for a same Data Feeder). Indeed Service Adapter implementation cannot guarantee that 2 operations with a successive Transaction Ids are executed in the right order (because of network latency). This case is illustrated in the next figure. In that case ignore is the oldest command.



Each operation is performed in a dedicated thread. It is mandatory to synchronize control operations on the same object.

Concerning the design of the components performing the collection, there are at least 3 main components:

- A component implementing the operations of the Service Measure: Measures class
- A component managing the queue of measures. This component is responsible for providing services to insert, retrieve and destroy measures into/from a queue: MeasureQueueManager
- A component responsible for collecting measure on the third party product: MeasureCollector
  This component is executed in a dedicated thread and control (started/ stopped) by the Measure Service Operations

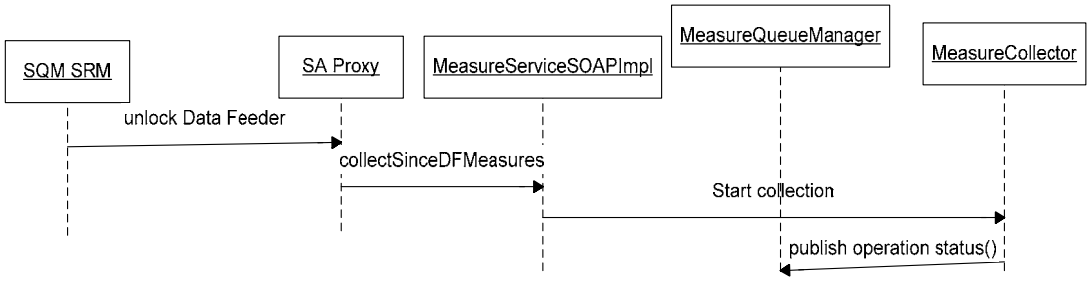


When the Service Adapter is also able to handle measure resynchronization a dedicated component is recommended.

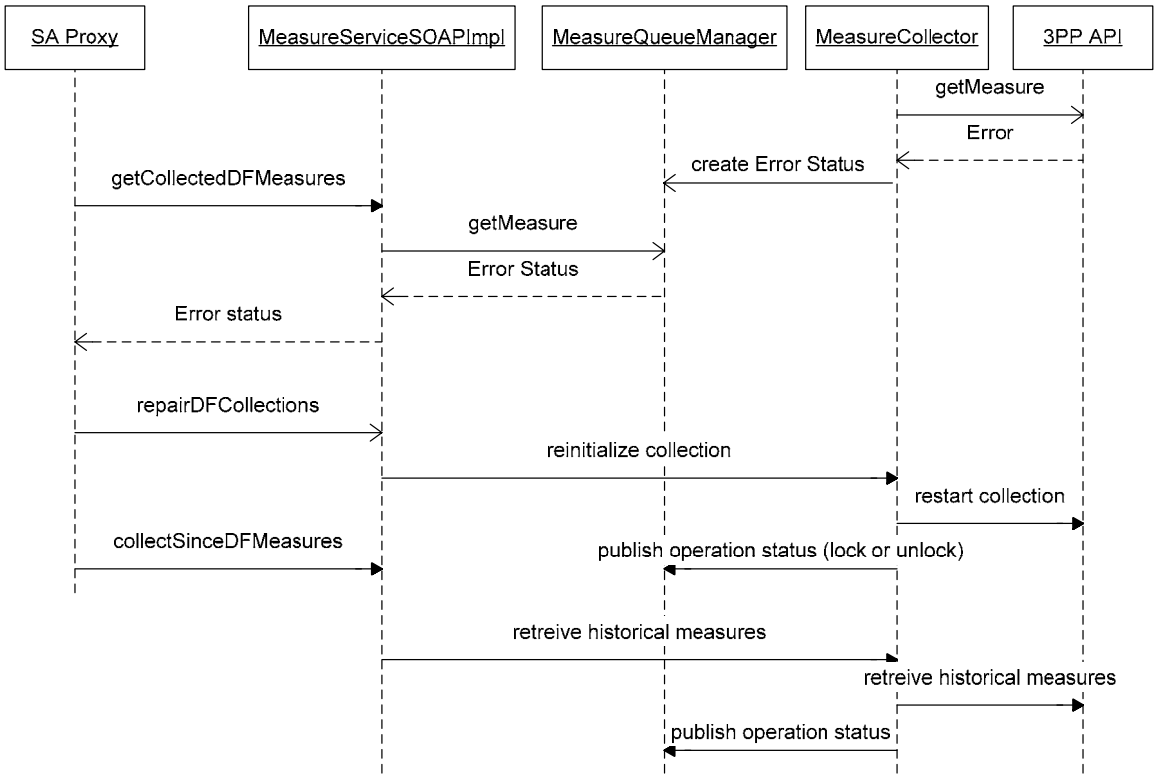
5.1.4.1 Operation collectSinceDFMeasures

This operation starts the collection of measures of a provided set of Data Feeders. The operation is also used for initiating the collection of historical measures for resynchronization purpose (if this feature is supported by the Service Adapter)

1. The operation is called when the SQM Service Repository Manager (SRM) unlocks a Data Feeder

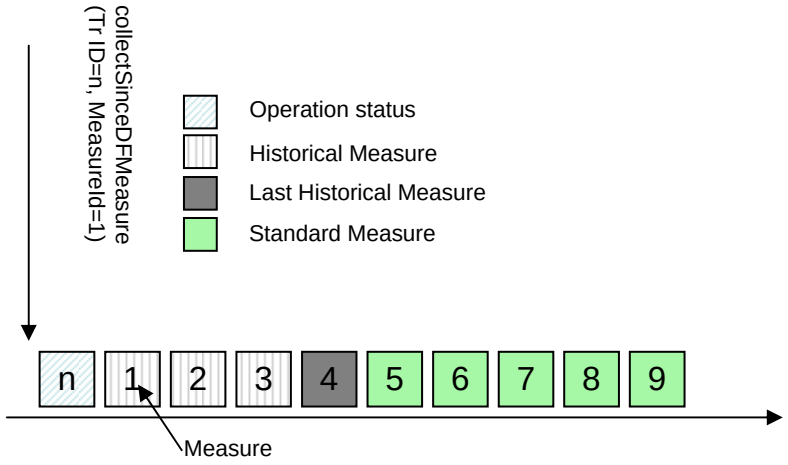


2. The operation is also called by the SA Proxy (Service Adapter client) after a collection failure, to restart the collection and retrieve historical data which have been lost during the failure time.

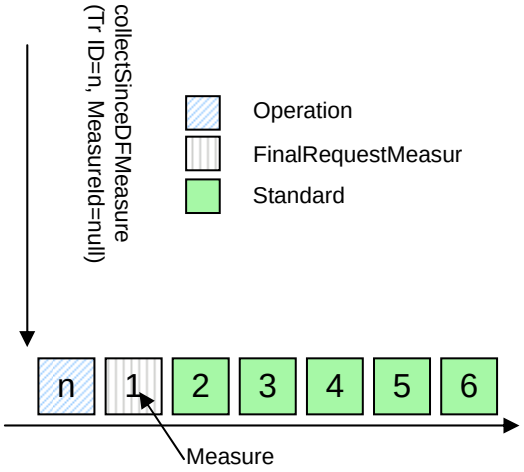


When the Service Adapter supports measures resynchronization, the SA client (SA Proxy) puts in each Data Feeder control structure (DataFeederStartControlId), the ID of the measure from which the resynchronization must be performed. Historical measures must be marked using a dedicated flag and the last measure of resynchronization must be also marked as the last one. It is important to not mix the

historical measures with the current flow of measures. Measure should be emitted in the order of their measure identifier (as illustrated in the next figure).



When the collectSinceDFMeasures does not provide a MeasureId in the control structure (DataFeederStartControlId), it means that no synchronization is requested. Anyway the Service Adapter must return the last available measure. This measure must be marked as a resynchronization measure by setting the flag isFinalRequestedMeasure to true on the measure. Thank to this flag, the Service Adapter Proxy can determine its last point of resynchronization.



Implementation example

In the following example, the collectSinceDFMeasure operation just starts the collector of measures and publishes an operation status for each Data Feeder in the queue of measures.

```
public class MeasureServiceSOAPImpl implements MeasureService{

    public ArrayOfDataFeederStatusReport
          collectSinceDFMeasures(RegistrationId regId,
                 ArrayOfDataFeederStartControlId dataFeederStartControlIds)
                throws java.rmi.RemoteException, DFControlException {
          try {
                return getContext(regId).
                getMeasure().
```

```
                        collectSinceDFMeasures(dataFeederStartControlIds);
            }
            catch (Throwable t) {
                  t.printStackTrace();
                  return null;
            }
      }

      private Context getContext(RegistrationId regId) throws
DFControlException {
          Context context = null;
          try {
              Registration reg = Registration.getInstance();
              context = reg.getContext(regId);
          } catch (RegistrationException e) {
              String trace = e.getErrorDefaultTranslation();
              throw new DFControlException(trace, e.getError(),null);
          }
          return context;
      }
….
}
```

```
public class Measures {

private final Object MEASURE_LOCK               = new Object();

public ArrayOfDataFeederStatusReport collectSinceDFMeasures(
                ArrayOfDataFeederStartControlId
arrayOfDataFeederStartControlIds)
        throws RemoteException, DFControlException {

   // Synchronized the operation with the other control operations
   synchronized (MEASURE_LOCK){
         // Initialize the returned operation status
         DataFeederStartControlId[]   dataFeederStartControlIds
              = arrayOfDataFeederStartControlIds.getItem();
         DataFeederId            dfId;
         MeasureId               measureId;
         DataFeederStatus        dfStatus;

         DataFeederStatusReport[] dfStatusReport
            = new DataFeederStatusReport[dataFeederStartControlIds.length];

         // For each requested DF, build the report status
         // and publish this status into the queue of measures
         for (int i = 0; i < dataFeederStartControlIds.length; i++) {
               dfId    = dataFeederStartControlIds[i].getDataFeederId();

               dfStatusReport[i] = new DataFeederStatusReport();

               // Construct the Data Feeder Status
               dfStatus = new DataFeederStatus(
                      AdministrativeState.UNLOCKED,
                                  AvailabilityStatus.AVAILABLE,
                                  "Collecting",
                       dataFeederStartControlIds[i].getTransactionId());

               // Publish the status in the queue of measures
               publishStatus(dfId,dfStatus);

               // Append the status to the reply
```

```
                    Date currentDateTime = new Date();
                    String timestamp;
                    timestamp =
                      DateTimeFormat.dateToISOStringInGMT(currentDateTime);
                    dfStatusReport[i].setTimestamp(timestamp);
                     dfStatusReport[i].
                     setDataFeederId(
                            dataFeederStartControlIds[i].getDataFeederId());
                    dfStatusReport[i].setDataFeederStatus(dfStatus);
                }

                // Create a new the measure collector if it does not exists
                if ( (m_measureCollector==null) ||
    (m_measureCollector.getState()==MeasureCollector.MeasureConsumer.STOP)) {
                     // Start a new collector
                     m_measureCollector=
                            new MeasureCollector(m_measuresCollection);
                }

                // Start collecting measures
                m_measureCollector.start();

                return new ArrayOfDataFeederStatusReport(dfStatusReport);
            }
        }
    }
```

5.1.4.2 Operation getCollectedDFMeasures

The operation fetches the measures (values and report statuses) accumulated by the Service Adapter. Thus, the SQM SA Proxy retrieves a flow of collected or historical measures.

When there is no available measures, the operation waits until the given timeout parameter. If new measures are accumulated during this period, the operation returns them else it returns null.

When some measures are availables, the operation returns the maximum number of measures as indicated in the given parameter.

Implementation example

In the following example, the getCollectedDFMeasures operation fetches the available measures from the queue of measures and returns them.

```
public class MeasureServiceSOAPImpl implements MeasureService{

    public ArrayOfMeasure getCollectedDFMeasures(RegistrationId regId,
          long timeoutInMillis,int maxMeasureCount)
                throws java.rmi.RemoteException, FlushException {
          Measures measureServiceInstance=getContext_FE(regId).getMeasure();
          ArrayOfMeasure measures=null;
          try {
             measures=measureServiceInstance.
                getCollectedDFMeasures(timeoutInMillis,maxMeasureCount);
          } catch (Throwable t) {
                t.printStackTrace();
          }

        return measures;
    }
```

```
        private Context getContext_FE(RegistrationId regId) throws
    FlushException {
            Context context = null;
            try {
                Registration reg = Registration.getInstance();
                context = reg.getContext(regId);
            } catch (RegistrationException e) {
                String trace =e.getErrorDefaultTranslation();
                throw new FlushException(trace, e.getError());
            }
            return context;
        }
    ….
    }
```

```
    public class Measures {

       private final Object MEASURE_LOCK               = new Object();

       public ArrayOfMeasure getCollectedDFMeasures(long timeoutInMillis,
              int maxMeasureCount) throws RemoteException, FlushException {

            ArrayOfMeasure l_result = null;

            try {
                Measure[] measure;
                if (maxMeasureCount > 0) {
                    // Wait for at least one measure in the queue of measures
                    if (m_measuresCollection.isEmpty()) {
                        try {
                            // This code is provided as an example and it not
                            // the optimal solution. A better solution is to wait
                            // until a measure is available
                             Thread.sleep(timeoutInMillis);
                        } catch (InterruptedException e) {
                             throw new
                                   FlushException(e.getLocalizedMessage(),null);
                        }
                    }
                    if (!m_measuresCollection.isEmpty()) {
                        // Retreive measures from the queue of measures
                        l_result =
                            m_measuresCollection.getMeasure(maxMeasureCount);

                        int retrievedMeasureCount = l_result.getItem().length;
                    }
                }
            } catch (Throwable e) {
                ErrorDesignation errDesign
                    = new ErrorDesignation(null,
                                "Internal Error");
                e.printStackTrace();
                throw new FlushException("Internal Error", errDesign);
            }

            return l_result;
        }
    }
```

5.1.4.3 Operation repairDFCollections

This control operation tries to reestablish the resources (sockets, etc) of the Data Feeder collections after a failure. The operation is called each time an error report is published in the queue of measure during the collection.

As for the other control operations, it publishes per Data Feeder the operation status in the queue of measures and also returns this status.

When the operation succeed in restoring the collection flow for a DF, the SQM SA Proxy will call the collectSinceDFMeasures operation to start the collection and resynchronize measure which have been lost during the failure period.

Implementation example

In the following example, the repairDFCollection operation just returns a success status.

```
public class MeasureServiceSOAPImpl implements MeasureService{

   public ArrayOfDataFeederStatusReport repairDFCollections(
          RegistrationId regId,
          ArrayOfDataFeederControlId dataFeederControlIds)
   throws java.rmi.RemoteException, DFControlException {
        return getContext(regId).
                 getMeasure().
                       repairDFCollections(dataFeederControlIds);
    }

    private Context getContext(RegistrationId regId) throws
DFControlException {
        Context context = null;
        try {
            Registration reg = Registration.getInstance();
            context = reg.getContext(regId);
        } catch (RegistrationException e) {
            String trace = e.getErrorDefaultTranslation();
            throw new DFControlException(trace, e.getError(),null);
        }
        return context;
    }
   ….
}
```

```
public class Measures {

   private final Object MEASURE_LOCK               = new Object();

   public ArrayOfDataFeederStatusReport repairDFCollections(
                ArrayOfDataFeederControlId arrayOfDataFeederControlIds)
                throws RemoteException, DFControlException {
   synchronized (MEASURE_LOCK) {

          DataFeederControlId[] dataFeederControlIds =
                arrayOfDataFeederControlIds.getItem();

          DataFeederStatusReport[] dfStatusReport =
                new DataFeederStatusReport[dataFeederControlIds.length];

          DataFeederStatus dfStatus;

          // Nothing to do - consider that collection cannot fail
```

```
            // Iterate on each DataFeederControlId to build the return
            // status
            // A DataFeederControlId contains the DataFeederID to repair
            for (int i = 0; i < dataFeederControlIds.length; i++) {
                  DataFeederId dfId =
                        dataFeederControlIds[i].getDataFeederId();

                  // Construct a status and publish it
                  dfStatus =
                        new DataFeederStatus(AdministrativeState.UNLOCKED,
                              AvailabilityStatus.AVAILABLE,
                              "Up",
                              dataFeederControlIds[i].getTransactionId());

                  // Publish a status report in the queue of measures
                  publishStatus(dfId,dfStatus);

                  // Contruct the return status
                  String timestamp;
                  Date date = new Date();
                  timestamp = DateTimeFormat.dateToISOStringInGMT(date);
                  dfStatusReport[i] = new DataFeederStatusReport();
                  dfStatusReport[i].setTimestamp(timestamp);
                  dfStatusReport[i].setDataFeederId(dfId);
                  dfStatusReport[i].setDataFeederStatus(dfStatus);
            }

            return new ArrayOfDataFeederStatusReport(dfStatusReport);
      }
      }
….
}
```

5.1.4.4 Operation stopDFCollections

This control operation stops the collection of the specified Data Feeder.

This operation is called when stop in the SQM SA Proxy application of when locking a Data Feeder.

As for the other control operations, it publishes per Data Feeder the operation status in the queue of measures and also returns this status.

Implementation example

In the following example, the repairDFCollection operation just returns a success status.

```
public class MeasureServiceSOAPImpl implements MeasureService{

   public ArrayOfDataFeederStatusReport repairDFCollections(
         RegistrationId regId,
         ArrayOfDataFeederControlId dataFeederControlIds)
   throws java.rmi.RemoteException, DFControlException {
       return getContext(regId).
                  getMeasure().
                        repairDFCollections(dataFeederControlIds);
    }

    private Context getContext(RegistrationId regId) throws
DFControlException {
        Context context = null;
        try {
```

```
                Registration reg = Registration.getInstance();
                context = reg.getContext(regId);
            } catch (RegistrationException e) {
                String trace = e.getErrorDefaultTranslation();
                throw new DFControlException(trace, e.getError(),null);
            }
            return context;
        }
        ….
    }
```

```
    public class Measures {

        private static final Object MEASURE_LOCK               = new Object();

        public ArrayOfDataFeederStatusReport stopDFCollections(
                    ArrayOfDataFeederControlId arrayOfDataFeederControlIds)
                    throws RemoteException, DFControlException {
        synchronized (MEASURE_LOCK) {

               DataFeederControlId[] dataFeederControlIds =
                      arrayOfDataFeederControlIds.getItem();

               DataFeederStatusReport[] dfStatusReport =
                      new DataFeederStatusReport[dataFeederControlIds.length];

               DataFeederStatus dfStatus;

               // For each DF construct and publish it status
               for (int i = 0; i < dataFeederControlIds.length; i++) {
                      DataFeederId dfId =
                             dataFeederControlIds[i].getDataFeederId();
                      dfStatus =
                             new DataFeederStatus(AdministrativeState.LOCKED,
                                          AvailabilityStatus.AVAILABLE,
                                          "Stopped",
                                          dataFeederControlIds[i].
                                                 getTransactionId());

                      // Insert the status in the collection queue
                      publishStatus(dfId, dfStatus);

                      // Construct the status report
                      Date currentDateTime = new Date();
                      String timestamp;
                      timestamp = DateTimeFormat
                             .dateToISOStringInGMT(currentDateTime);

                      dfStatusReport[i] = new DataFeederStatusReport();
                      dfStatusReport[i].setTimestamp(timestamp);
                      dfStatusReport[i].setDataFeederId(dfId);
                      dfStatusReport[i].setDataFeederStatus(dfStatus);
               }

               // Suspend the Collector of measures if not stopped
               if (m_measureCollector.getState()!=
                      MeasureCollector.MeasureConsumer.STOP){
                             m_measureCollector.suspend();
               }


               return new ArrayOfDataFeederStatusReport(dfStatusReport);
```

```
            }
            }
        ….
        }
```

Chapter 6

Debugging, Troubleshooting and Tracing

6.1 Debugging a SQM Service Adapter

This section explains how to debug a Java SQM Service Adapter. It provides an example based on the debugging of the SQM Sample Service Adapter.

6.1.1 Required environment

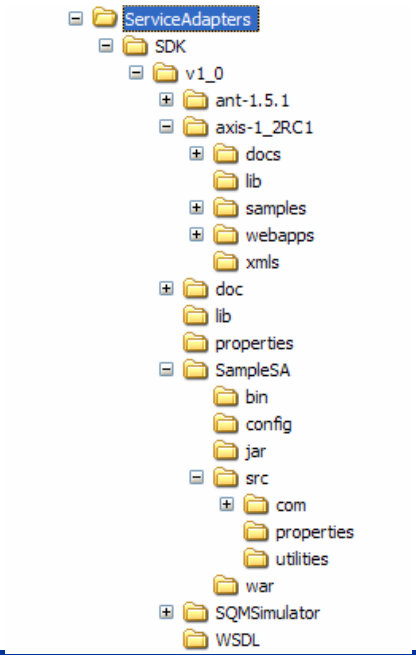
The proposed debugging solution is explained for a Windows platform with the following development environment installed:

- Eclipse release 3.0.1 (can be downloaded from the following location: http://www.eclipse.org/downloads/index.php)
- A TomCat Eclipse plugin (http://www.sysdeo.com/eclipse/tomcatPluginV3.zip)
- TomCat 4.1.31
- Java 2 Standard Edition Software Development Kit (SDK). Recommended release: 1.4.1_05

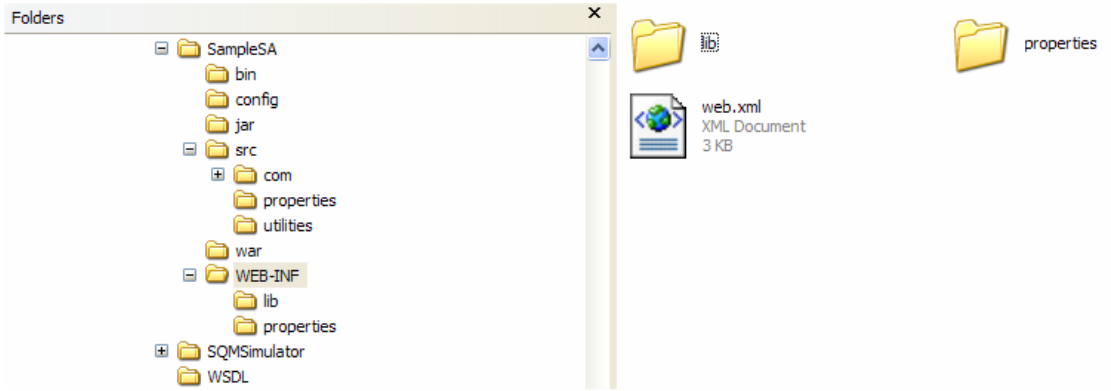
6.1.2 Setting up the project environment

For debugging the Sample Service adapter, before creating the Eclipse project:

1. Copy the SA SDK kit on a Windows Platform:



2. Open a Windows command prompt
3. Set up the Apache Ant environment:
```
>cd <SDK root
directory>\ServiceAdapters\SDK\v1_0\SampleSA>
>set ANT_HOME=<SDK root
directory>\ServiceAdapters\SDK\v1_0\ant-1.5.1
>set PATH=%PATH%;%ANT_HOME%\bin
```
4. Set up the Java environment
```
>set JAVA_HOME=<Java installation directory>
```
5. set up the Eclipse project environment for the SampleSA using the Ant target named 'eclipse-dbg' in the SampleSA build.xml file:
```
> ant eclipse-dbg
```
6. The resulting SampleSA directory should be structured as:

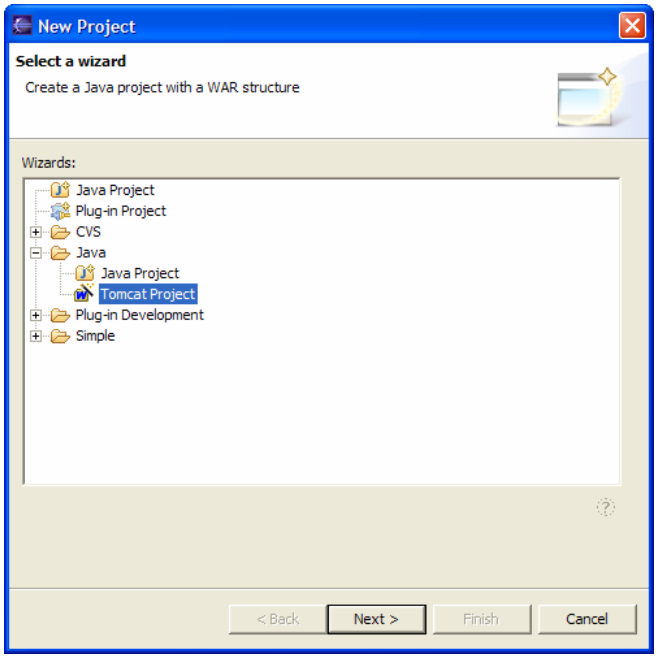


7. Run Eclipse

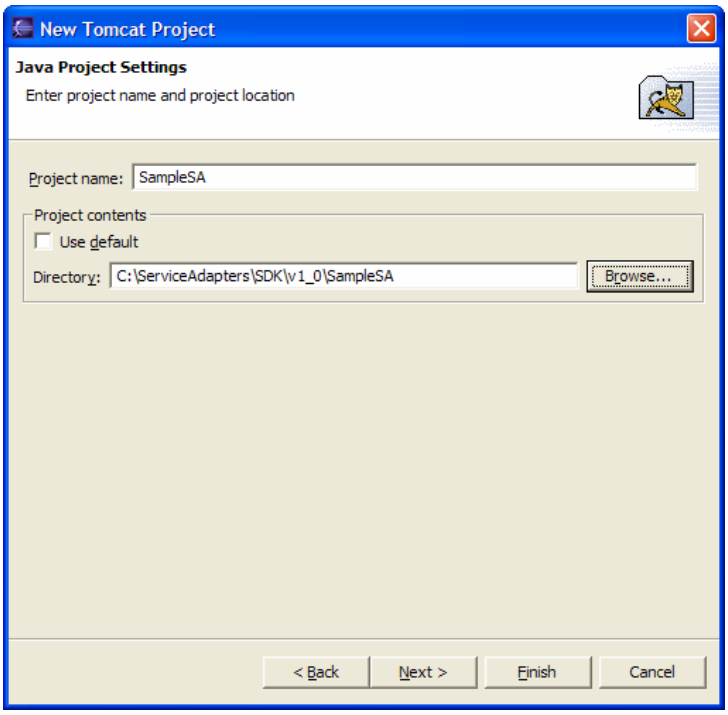
6.1.3 Setting up the Eclipse project

To create the SampleSA project in Eclipse,

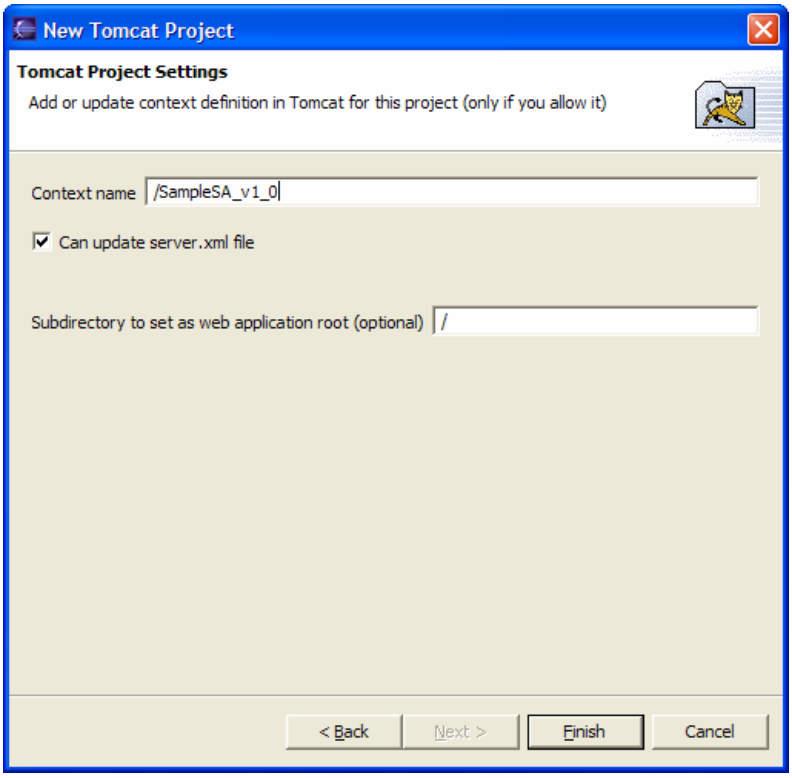
1. Create a TomCat project by using the Eclipse project creation wizard (File->New->Project …) and select 'TomCat Project':



2. Set the project name and browse the project location (it must be: <SA SDK root directory>\ServiceAdapters\SDK\v1_0\SampleSA)

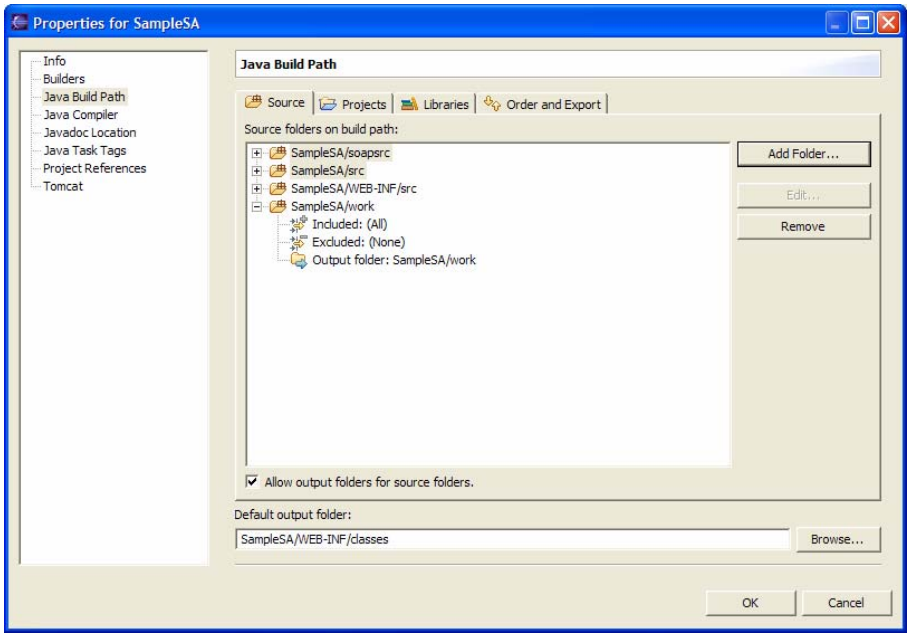


3. Update default Project settings. The context name must be '/SampleSA_v1_0'



4. The project is now created, it is necessary to include the SampleSA source files: Select the project and display project's properties. In the 'Java build path' section

add the SampleSA source files: directory named ‘src’ and directory name ‘soapsrc’ You should obtain:

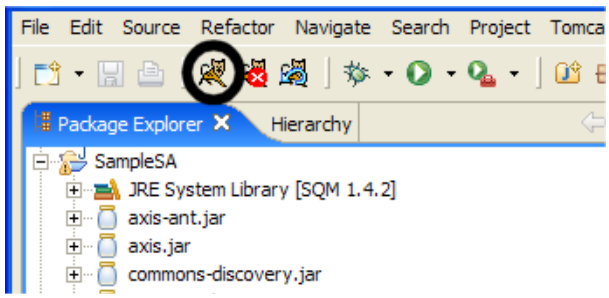


5. The SampleSA project is now ready to be used.

6.1.4 Deploying the SampleSA services

The last step consists in starting the SampleSA and to deploy the WebServices:

1. Start TomCat under eclipse:



2. Deploy the SampleSA services by executing the ‘deploy’ target in the Windows Command prompt:

```
> ant deploy
```

6.1.5 Debugging the Service Adapter

The Service Adapter can now be debugged as any other Java application.

Use the SQMSimulator to call operations to debug (see chapter 3.1.1.6,)


invent


Java™
COMPATIBLE