

Retek® Service Layer™ 11.1.1

Programmer's Guide

Corporate Headquarters:

Retek Inc.
Retek on the Mall
950 Nicollet Mall
Minneapolis, MN 55403
USA
888.61.RETEK (toll free US)
Switchboard:
+1 612 587 5000
Fax:
+1 612 587 5100

European Headquarters:

Retek
110 Wigmore Street
London
W1U 3RW
United Kingdom
Switchboard:
+44 (0)20 7563 4600
Sales Enquiries:
+44 (0)20 7563 46 46
Fax:
+44 (0)20 7563 46 10

The software described in this documentation is furnished under a license agreement, is the confidential information of Retek Inc., and may be used only in accordance with the terms of the agreement.

No part of this documentation may be reproduced or transmitted in any form or by any means without the express written permission of Retek Inc., Retek on the Mall, 950 Nicollet Mall, Minneapolis, MN 55403, and the copyright notice may not be removed without the consent of Retek Inc.

Information in this documentation is subject to change without notice.

Retek provides product documentation in a read-only-format to ensure content integrity. Retek Customer Support cannot support documentation that has been changed without Retek authorization.

The functionality described herein applies to this version, as reflected on the title page of this document, and to no other versions of software, including without limitation subsequent releases of the same software component. The functionality described herein will change from time to time with the release of new versions of software and Retek reserves the right to make such modifications at its absolute discretion.

Retek[®] Service Layer[™] is a trademark of Retek Inc.

Retek and the Retek logo are registered trademarks of Retek Inc.

This unpublished work is protected by confidentiality agreement, and by trade secret, copyright, and other laws. In the event of publication, the following notice shall apply:

©2005 Retek Inc. All rights reserved.

All other product names mentioned are trademarks or registered trademarks of their respective owners and should be treated as such.

Printed in the United States of America.

Customer Support

Customer Support hours

Customer Support is available 7x24x365 via email, phone, and Web access.

Depending on the Support option chosen by a particular client (Standard, Plus, or Premium), the times that certain services are delivered may be restricted. Severity 1 (Critical) issues are addressed on a 7x24 basis and receive continuous attention until resolved, for all clients on active maintenance. Retek customers on active maintenance agreements may contact a global Customer Support representative in accordance with contract terms in one of the following ways.

Contact Method	Contact Information
----------------	---------------------

E-mail	support@retек.com
--------	-------------------

Internet (ROCS)	rocs.retek.com Retek's secure client Web site to update and view issues
-----------------	---

Phone	+1 612 587 5800
-------	-----------------

Toll free alternatives are also available in various regions of the world:

Australia	+1 800 555 923 (AU-Telstra) or +1 800 000 562 (AU-Optus)
France	0800 90 91 66
Hong Kong	800 96 4262
Korea	00 308 13 1342
United Kingdom	0800 917 2863
United States	+1 800 61 RETEK or 800 617 3835

Mail	Retek Customer Support Retek on the Mall 950 Nicollet Mall Minneapolis, MN 55403
------	---

When contacting Customer Support, please provide:

- Product version and program/module name.
- Functional and technical description of the problem (include business impact).
- Detailed step-by-step instructions to recreate.
- Exact error message received.
- Screen shots of each step you take.

Contents

Chapter 1 – Introduction	1
RSL overview	1
Chapter 2 – Technical architecture	3
Overview	3
Architecture layers	4
J2EE application model	5
Oracle PL/SQL-based application model	8
Chapter 3 – Basic operations	11
Overview	11
Simple service call	11
Chapter 4 – Service provider how-to guide	13
Service provider application configuration	13
Service provider application layer code for J2EE models	14
Service provider application layer code for Oracle PL/SQL-based models	15
Chapter 5 – “Client” how-to guide	17
Client application layer configuration	17
Client application layer code	19
Client application runtime requirements	20
Chapter 6 – RSLTestClient utility	21
Overview	21
Installation and configuration	21
Execution	22
Appendix A – Configuration files	23
Services framework configuration	23
Log4j configuration	23
Other configuration	24

Appendix B – Common libraries 25

Platform..... 25

RSL/Integration..... 25

Third party provided 26

Chapter 1 – Introduction

This manual is designed for system administrators, developers, and applications support personnel. It provides a basic understanding of the Retek Service Layer components, including the flow of a synchronous call between two applications. This chapter describes the components that make up the Retek Service Layer (RSL). These components can be distributed via an application server (Oracle OC4J or IBM WebSphere) or can be used in a stand alone environment.

RSL overview

RSL handles the interface between a client application and a server application. The client application typically runs on a different host than the service. However, RSL allows for the service to be called internally in the same program or Java Virtual Machine as the client without the need for code modification.

All services are defined using the same basic paradigm. The input and output to the service, if any, is a single set of values. Errors are communicated via Java Exceptions that are thrown by the services. The normal behavior when a service throws an exception is for all database work performed in the service call being rolled back.

RSL works within the J2EE framework. All services are contained within an interface offered by a Stateless Session Bean. To a client application, each service appears to be a method call.

Some Retek applications, such as RMS, are implemented in the PL/SQL language, which runs inside of the Oracle database. RSL uses a generalized conversion process that converts the input java object to a native Oracle Object and any output Oracle Objects to the returned java object from the service. There is a one-to-one correspondence of all fields contained in the Java parameters as in the Oracle Objects used.

A client does not need to know if the business logic is implemented as an Oracle Stored Procedure or in some other language.

Chapter 2 – Technical architecture

Overview

This chapter describes the overall architecture of the RSL. The RSL architecture is built on J2EE's Java Enterprise Bean technology. It is composed of different architecture layers that perform specific task within the overall workflow of integration between two applications.

RSL provides two different models for service providers. The election of which model to use depends on the type of application the service provider developer is adding the RSL layer to. For applications that follow the J2EE or simple Java architecture, a J2EE model is a better fit. An Oracle PL/SQL model is a better fit for applications that heavily depend on database business logic, such as Oracle Forms-based applications, for example Retek Merchandising System (RMS).

Client application developers do not need to be aware of this distinction. The developer only needs to implement the code that retrieves an instance of the service proxy using the service interface that RSL provides. The ServiceAccessor class makes the calls to an RSL service.

Architecture layers

The RSL is divided into a series of layers:

- **Client Application Layer (CAL):** This layer is developed by client application developers. The code for this is dependent on the business processes of the client.
- **Service Access Layer (SAL):** This layer is generated by the integration team. Its purpose is to provide a typed set of interfaces and implementations for accessing services. The SAL is the only set of interfaces that the CAL developer needs to be aware of. Different implementations of the SAL can be used by the client application, depending on the local or remote location of the required services. The CAL developer doesn't need to be aware of this.
- **Service Provider Layer (SPL):** This layer is implemented by the developers belonging to the service provider or by developers knowledgeable in the service application domain. Each service must implement its corresponding interface as declared in the SAL. For RMS, RDM, or other Oracle Forms applications, each service is offered via a PL/SQL Stored Procedure and uses Oracle Object technology for input and output parameters. For Java J2EE offered services, input and output parameters are generated via Value Object, old fashioned java beans that:
 - implement a defined interface
 - consist of getter, setter, and adder methods



Note: Refer to the following sections for a discussion of the J2EE and Oracle PL/SQL-based models.



Note: For Oracle based applications, a generic “Stored Procedure Caller” class, provided by the Integration Team, is accessible through the platform CommandExecutionServiceEjb Stateless Session Bean. This class handles all RMS provided simple services.



Note: The services offered by a single service should be a logical unit that is functionally cohesive. This has implications if a retailer wishes to use a different implementation, such as a completely home-grown implementation for this functionality.

J2EE application model

Figure 1 depicts the model and workflow you should use when you integrate RSL with a J2EE or simple Java application. The objects on the left side of the dashed lines symbolize the client view of the transaction while the right side characterizes the service provider part. In the diagram, objects in blue are implementations provided by either the client application developer (left side) or service provider developer (right side.) The red objects indicate the interfaces, classes and payload objects provided by the integration team to both developers. Retek's platform objects are denoted in green.

The SPL developer needs to create a POJO (Plain Old Java Object) class that implements the SAL interface provided by the Integration team. This class should be made available in the J2EE environment through the CommandExecutionService EJB. Please refer to “Chapter 4 – Service provider how-to guide” for an in-depth discussion on how to develop the Service Provider Layer and make it available to RSL client applications.

The CAL developer creates the code that makes it possible for the client application to contact the RSL service. This involves using the Platform provided ServiceAccessor class to retrieve an instance of the Service Proxy to communicate with the service. Once this proxy is obtained, the CAL invokes calls as declared in the service interface provided by the integration team as part of the SAL deliverables. “Chapter 5 – Client how-to guide” explains how to develop the client code and the configurations to successfully invoke RSL services.

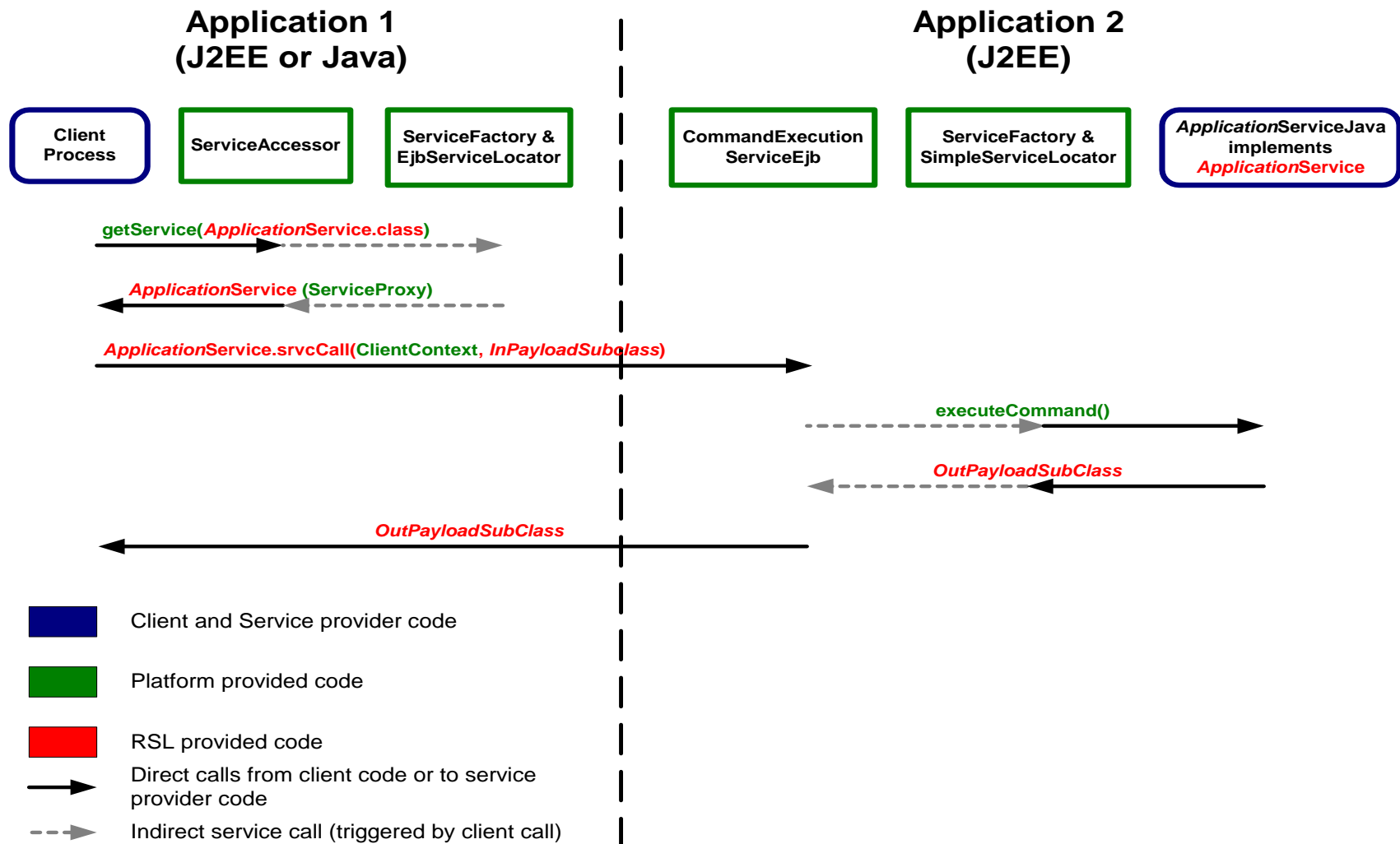


Figure 1

Oracle PL/SQL-based application model

Figure 2 shows the model and workflow that you should use when you integrate RSL with an application based on Oracle PL/SQL language, such as RMS.

- The objects on the far left symbolize the client view of the transaction
- The objects between both dashed lines characterize the service provider part
- The objects on the extreme right represent the Oracle-based application

In this model, the integration team is responsible for distributing the service provider implementation of RSL. In the diagram, objects in blue are implementations provided by either the client application developer (left side) or application team developer (right side.) The red objects indicate the interfaces, classes, and payload objects provided by the integration team to both developers. Retek's platform objects are denoted in green.

The application team is responsible for developing the PL/SQL Stored Procedure following the guidelines provided by the integration team and discussed in “Chapter 4 – Service provider how-to guide”.

The CAL developer creates the code that makes it possible for the client application to contact the RSL service. This involves using the Platform provided ServiceAccessor class to retrieve an instance of the Service Proxy to communicate with the service. Once this proxy is obtained, the CAL invokes calls as declared in the service interface provided by the integration team as part of the SAL deliverables. “Chapter 5 – Client how-to guide” explains how to develop the client code and the configurations to successfully invoke RSL services.

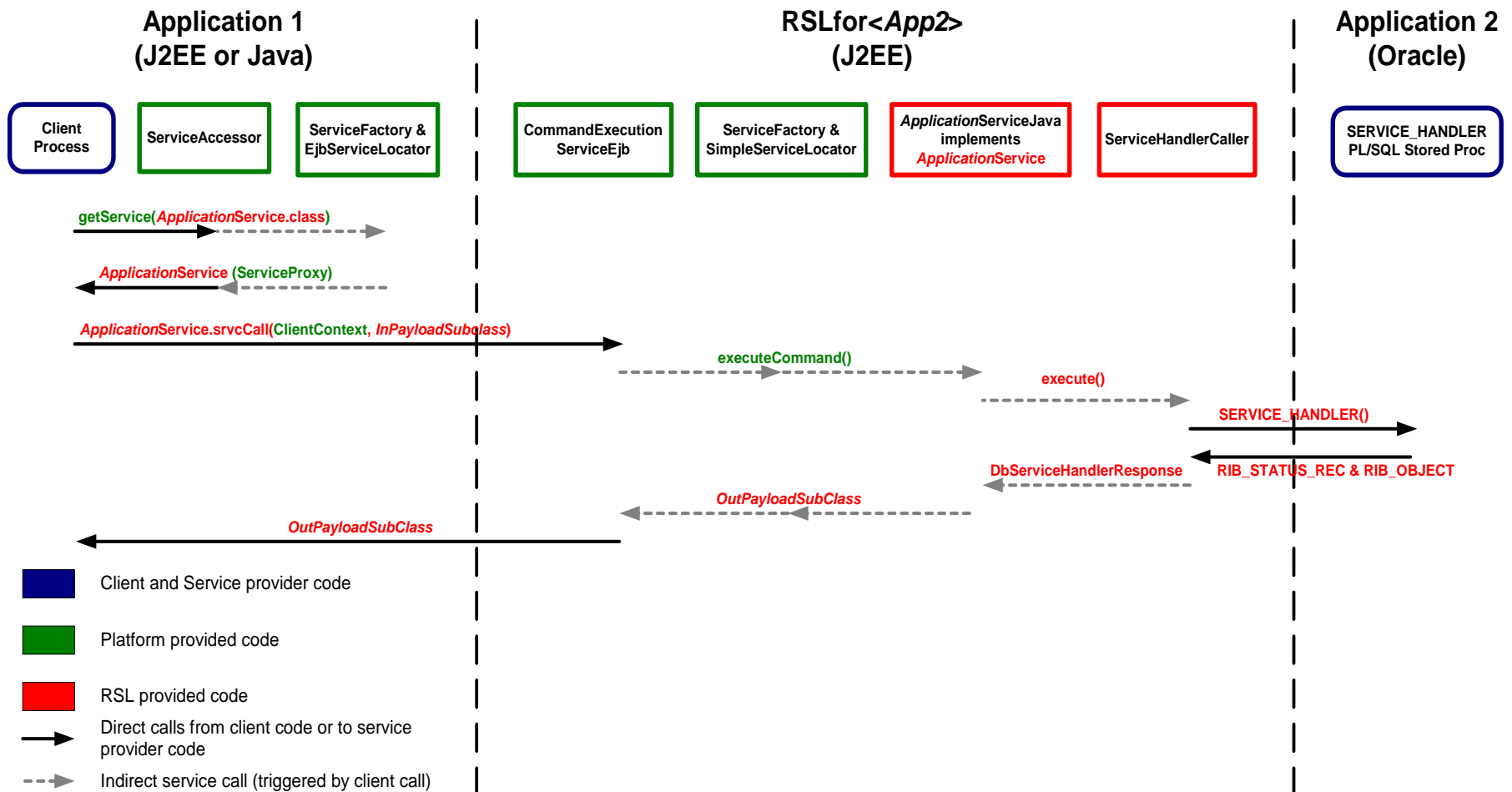


Figure 2

Chapter 3 – Basic operations

Overview

Chapter 3 explains the synchronous data flow of a payload between end applications. The business logic is completely owned by the application implementing the service.

Simple service call

In the client application, some event triggers a service call. The first step is for the client to find the location of the service. This is done by using the ServiceAccessor, which connects to the application server's Java Naming and Directory Interface (JNDI) instance. Once the Remote Home of the service is located, the client creates a Remote instance or handle to the service, which is returned to the client to perform calls to the service.

At this point, the J2EE application server infrastructure takes control and performs a remote method invocation on the CommandExecutionService Stateless Session Bean. This EJB is responsible for looking up the implementation of the service interface, invoking the client requested service method call and returning the result.

The execution and control of database commits and rollbacks are dependent on three factors: the configuration of the Stateless Session Bean, the success or failure of the service call, and whether the transaction is started by the client or the application server.

- **Configuration of the Stateless Session Beans:** Normally, RSL beans are configured with container managed transactions. This means that the application server EJB container decides if database work is committed or not. Furthermore, there is the assumption that the configuration of the database connection is a container managed resource. Within a container each resource has a specific name. A service may use one or more resources during its execution.
- **Success or failure of the service call:** If an error is encountered during the service execution, normal behavior is to roll back all database work. This is performed when an exception is thrown by the service for container managed transactions.
- **Clients starting a transaction:** Most service calls have their transactions started by the service implementation or the application server and *not* by the client. However, it is possible for the client to start a transaction and make multiple service calls within the same transaction.

Chapter 4 – Service provider how-to guide

Service provider application configuration

As mentioned earlier, the service is developed by the service provider application. The following files need to be properly configured in the service provider application.

retек/services_rsl.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<services-config>
  <customizations>
    <interface package="com.retek.rsl.<app>">
      <impl package="<application specific>" />
    </interface>
  </customizations>
</services-config>
```

The name of the package for the interface usually follows the “com.retek.rsl.<app>” convention, where <app> is substituted by the name of the service provider application, for example rsm, rpm.

The implementation package name is provided by the service provider once the code is developed; this name is irrelevant for the client application.

retек/service_flavors.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<services-config>
  <flavors set="server">
    <flavor name="java"
locator="com.retek.platform.service.SimpleServiceLocator"
suffix="Java"/>
  </flavors>
</services-config>
```

Service provider application layer code for J2EE models

An example of a SPL service implementation is seen below. This sample implements the `PriceInquiryService` interface. All of the methods have been stubbed out, as the logic is unknown to anyone outside of the service providing application.



Note: The name of these implementation classes should follow a naming convention such as *ServiceInterfaceNameJava* (see example above). This allows the platform service framework to locate the SPL's implementation class by using the server flavor. This naming convention is configured in the SPL's `service_flavors.xml` file.

```
package com.retek.rpm.service;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

import com.retek.platform.exception.RetekBusinessException;
import com.retek.platform.service.ClientContext;
import com.retek.platform.service.FallbackHandler;
import com.retek.rib.binding.payload.PrcInqDesc;
import com.retek.rib.binding.payload.PrcInqReq;
import com.retek.rsl.rpm.PriceInquiryService;

public class PriceInquiryServiceJava implements PriceInquiryService
{

    protected final Log LOG = LogFactory.getLog(getClass());

    public void attachFallbackHandler(FallbackHandler arg0) {
        LOG.debug("Executing business logic for attachFallbackHandler");
    }

    public PrcInqDesc prcinqqry(ClientContext client, PrcInqReq query)
    throws RetekBusinessException {
        LOG.debug("Executing business logic for query");
        return new PrcInqDesc();
    }

}
```

Service provider application layer code for Oracle PL/SQL-based models

For each service interface in the Service Access Layer (SAL), the application developer must create a package that contains a stored procedure named `SERVICE_HANDLER`. The signature of this stored procedure is as follows:

```
PROCEDURE SERVICE_HANDLER(O_status  OUT  RIB_OBJECT,
                          O_payload  OUT  RIB_OBJECT,
                          I_action   IN   VARCHAR2,
                          I_payload   IN   RIB_OBJECT,
                          I_client   IN   RIB_OBJECT);
```

The `O_status` return object should be an instance of a `RIB_STATUS_REC` Oracle type object in the database. This object contains two variables. A `status_code` variable of type `varchar2` that holds the status of the `SERVICE_HANDLER()` call; a value of 'S' indicates the call was successful; any other status code should be accompanied by a description, assigned to the second `varchar2` variable of the `RIB_STATUS_REC` object.

The `O_payload` return object is the value that will be returned to the client application after the service call.

`I_action` is a `varchar2` representing what type of action to perform for the given payload. Actions should match one-to-one to methods in the service interface. Each method call from the client application will pass a different `I_action` value to the `SERVICE_HANDLER()` stored procedure; this way, the application developer can route the request to different business processes in their application.

`I_payload` corresponds to the object sent by the client application and used by the stored procedure to perform some business process action.

`I_client` is an object of type `RIB_CLIENT_REC` that represents the instance of the `ClientContext` object sent by the client application to the RSL service. This `ClientContext` is translated to a `RIB_CLIENT_REC` Oracle type that can be used by the application developer to identify the context used for the invocation of this call.

Chapter 5 – “Client” how-to guide

Client application layer configuration

As mentioned earlier, the CAL is developed by the “client application” developer. The following files need to be properly configured in the client application.

retek/services_rsl.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<services-config>
  <customizations>
    <interface package="com.retek.rsl.<app>" app="<app>">
      <impl package="" />
    </interface>
  </customizations>
</services-config>
```

An entry like the following should exist in the client application’s services_rsl.xml file. The name of the package for the interface usually follows the “com.retek.rsl.<app>” convention, where <app> will be substituted by the name of the service provider application: rsm, rpm.

The implementation package name is irrelevant in the client application, since this will go through the CommandExecutionService EJB to make the service calls.

retek/service_flavors.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<services-config>
  <flavors set="client">
    <flavor name="ejb"
locator="com.retek.platform.service.EjbServiceLocator" remote-
suffix="Remote" home-suffix="RemoteHome"/>
  </flavors>
</services-config>
```

retек/jndi_providers.xml

```
<?xml version="1.0" ?>
<ejb_context_overrides>
  <!-- For WebSphere -->
  <provider app="<app>"
url="iiop://<was_host>:<was_port>"
factory="com.ibm.websphere.naming. WsnInitialContextFactory">
    </provider>

  <!-- For OC4J -->
  <provider app="<app>"
url="ormi://<oas_host>:<oas_port>"
factory="com.evermind.server.rmiRMIIInitialContextFactory">
    </provider>
</ejb_context_overrides>
```


Client application layer code

An example of a CAL method call is seen below. This sample calls the `prcinqqry` method of the `PriceInquiryService`. The service interface and the input and output payload classes are provided by the Integration Team.

One parameter seen below is the Retek Platform `ClientContext` object. The CAL must create this object. It is used for application logging and tracking purposes. It is a required object on every service interface. Besides the constructor parameters, this object identifies the host name the object was created on, the process ID, the thread ID that created the object, and the locale, or language, of the client.

```
import com.retek.platform.exception.RetekBusinessException;
import com.retek.platform.service.ClientContext;
import com.retek.platform.service.ServiceAccessor;
import com.retek.platform.util.type.security.SecureUser;
import com.retek.rib.binding.payload.PrcInqDesc;
import com.retek.rib.binding.payload.PrcInqReq;

public class ClientCode {
    public PrcInqDesc prcinqqry(PrcInqReq request)
    throws RetekBusinessException {

        // create a client ID. This may be cached and reused.
        ClientContext ctx = ClientContext.getInstance();
        SecureUser user = new SecureUser("username", "firstname",
            "lastname", null);
        ctx.setUser(user);
        ctx.setLocale(Locale.US);

        // create a new client handle.
        PriceInquiryService service =
            ServiceAccessor.getService(PriceInquiryService.class);

        // call the query method on service
        try {
            PrcInqDesc price = service.prcinqqry(request);
            return price;
        } catch (RetekBusinessException rbe) {
            throw rbe;
        }
    }
}
```

Client application runtime requirements

If the RSL service is running within a WebSphere Application Server and the client application is running in a Sun's Java Virtual Machine; the client must pass the following key/value pair as a system property to the application:

```
javax.rmi.CORBA.UtilClass=com.ibm.ws.orb.WSUtilDelegateImpl
```

If not, a `ClassCastException` is thrown when attempting to contact the RSL service.

The reason that the `ClassCastException` is thrown is that WebSphere does not use the Sun provided IIOP implementation for remote EJB communication, but its own, which is incompatible with Sun's implementation. By setting the mentioned system property, the client application is forced to use WebSphere implementation of the IIOP implementation.

The following jar files need to be added to the client classpath to successfully communicate with the RSL service when this is running within an WebSphere Application Server:

- `ibmorb.jar`
- `ibmext.jar`
- `iwsorbutil.jar`
- `naming.jar`
- `namingclient.jar`
- `wsexception.jar`

These are in the lib directory of the WebSphere Application Server.

Chapter 6 – RSLTestClient utility

Overview

The RSL Client Test Application is a tool created to help verify that a client application can successfully establish communication with an RSL Service Provider Application. This tool is compatible with RSL releases 11.1.0 or newer.

The test consist of very simple service calls that, upon success, return a message indicating that a call has been made with a specific timestamp. In case of errors, it displays a message indicating the possible causes for the problem and how to fix it.

This test supports both RSL's J2EE application model and Oracle PL/SQL-based application model. If the RSL service provider follows the J2EE application model, an error message might occur during execution of the last test. As indicated by the error, this is not a concern if the test fails in the J2EE application model, as the last test might not be applicable for such model. If the RSL service provider follows the Oracle PL/SQL-based application model, then make sure that all installation instructions have been followed as outlined.

Installation and configuration

The first two steps are only required if the RSL service providers follow the Oracle PL/SQL-based application model. If not, please ignore these steps and go directly to step three (3).

1. Load the RSL test objects in the database. This can be done by executing the @CreateRslTestObjects command in a SQL Plus terminal. The CreateRslTestObjects.sql script is found in the testapp directory of the RSL server or client pak.
2. Load the RSLSVC_TEST package in the database. This can be done by executing the @RSLSVC_TEST.pkb command in a SQL Plus terminal. The RSLSVC_TEST.pkb script is found in the testapp directory of the RSL server or client pak.
3. If the RSL service provider runs within the Oracle Container for J2EE (OC4J) edit the jndi.properties file located in the testapp directory of the RSL server or client pak and enter the java.naming.security.principal and java.naming.security.credentials values. This must be the same as the username and password of the user that has started the OC4J container. Typically an administrative account is created when OC4J is installed.

4. Edit the `rsctestclient.bat` or `rsctestclient.sh` script and customize it for the client application environment:
 - a. Set the `JAVA_HOME` variable to the location of a 1.4.1 or higher JDK installation.
 - b. If the RSL service provider runs in OC4J, set the `RSL_SERVER_PROVIDER` variable to OC4J; if it runs in WebSphere Application Server, set it to WAS.
 - c. The `CONFIG_DIR` variable should be set to the location of the directory that contains the `retek/jndi_providers.xml`, `retek/service_flavors.xml` and `retek/services_rsl.xml` files to be used by the client application. Do not include "retek" in the path. For this test, it is recommended the use of the same configuration files as the client application uses when released.
 - d. Set `CLIENT_LIB` to the folder that contains all the jar files required for this test. The RSL paks do not distribute these jar files (except for the RSL specifics.) It is assumed that all jar files will be already available in the Client Application library directory.

Execution

Make sure the RSL Service Provider Application is up and running; then execute the `rsctestclient.bat` or `rsctestclient.sh` script. After all tests have been completed, the following message will be shown:

```
>>>> Test completed successfully <<<<
```

indicating that all tests completed successfully, or the following message:

```
>>>> Test completed with possible errors, see sections above <<<<
```

indicating that there might have been an error during the execution of one of the tests. Once an error has been detected, the other tests will not be performed. Please examine the output to determine the cause of the error.

When executing this test against an RSL service provider that follows the J2EE application model, and if steps 1 and 2 in the Installation and Configuration section were not followed, the following message will be shown

```
>>>> Test completed with possible errors, see sections above <<<<
```

even if all tests passed successfully. Ignore this message and assume all tests passed successfully if the last test ran was the `SERVICE_HANDLER` stored procedure test, as this is the last test and is not relevant for J2EE application models.

Appendix A – Configuration files

The RSL uses a variety of configuration files. These are mostly platform related that Retek has packaged as part of the RSL to allow independent configuration.

Services framework configuration

jndi_providers.xml

The file must be in the classpath located in a *retex* directory. The JNDI providers's XML is used by the EJBServletLocator to lookup services in remote JNDI's. Please reference the platform documentation for specific formatting of this file.

services_rsl.xml

The file must be in the classpath located in a *retex* directory. Within this file are the package locations of the service implementations, which are used to configure the ServiceFactory. Please reference the platform documentation for specific formatting of this file.

service_flavors.xml

The file must be in the classpath located in a *retex* directory. Within this file are flavorsets, which are used to configure the ServiceFactory. Please reference the platform documentation for specific formatting of this file

service_context_factory.xml

The file must be in the classpath located in a *retex* directory. This specifies the factory class used by the platform to retrieve a service context implementation for RSL.

Log4j configuration

Retek utilizes the commons-logging framework provided by Jakarta.apache.org. The current property file is configured to use Log4J as its default factory.

log4j.dtd

This file must be in the classpath. This is required for validation of the log4j.xml. For more information regarding the use of this file, please review <http://logging.apache.org/log4j/docs/>.

log4j.xml

This file must be in the classpath. This is the main configuration for log4j. For more information regarding the use of this file, please review <http://logging.apache.org/log4j/docs/>.

Other configuration

commons-logging.properties

This file must be in the classpath. It contains the correct logging factory to instantiate for RSL. We utilize the Log4Jfactory, but you may configure you application differently via this file

implfactory.properties

This file must be in the classpath. This is required for proper operation within a WebSphere environment. It includes the correct factory implementation for specific WebSphere handles.

Appendix B – Common libraries

This section lists the third party jars necessary for RSL functionality. It also describes why each is necessary and what it provides to the framework.

The RSL is built upon multiple existing technologies developed within Retek. Its reuse is essential to interoperability between the RSL and other applications

Platform

platform-server.jar

This jar contains platform specific code for Retek. This includes the Service framework. All services developed are dependant on this jar for its use of AbstractService and Service related classes.

platform-api.jar

This jar contains platform specific code for Retek. This includes exceptions and other objects that could be transferred “over the wire”.

platform-conf.jar

This jar contains platform specific configuration files.

platform-common.jar

This jar contains platform specific code for Retek.

RSL/Integration

retex-payload-typed.jar

This jar contains the java bean representation of business objects. They are collectively referred to as payloads. The payloads are generated utilizing the data in the RIB framework GUI. They are dependant on castor for marshalling and unmarshalling. Payloads located in this jar are type specific, this may cause interoperability issues with legacy applications that are non type specific, but follows in the future path of the project.

retex-rib-support.jar

This jar contains various utilities for use with Retek projects. Included is an OracleStructDumper which aides in the debugging of converting objects. There are also various string and factory objects that may be of use.

rsl.jar

This jar contains utility, helper, and exception classes used by the RSL framework.

rsl-<service provider application>-access.jar

This jar contains the specific interfaces to access services from another application (SPL). These classes ultimately are automatically generated by the RIB framework GUI.

rsl-<service provider application>-business-logic.jar

This jar contains specific RSL implementations of the interfaces. The CommandExecutionServiceEjb looks up these implementations. This jar is provided only for Oracle PL/SQL-based applications where the integration team provides the service provider layer.

Third party provided

Most of these jars are provided via open source. Retek recommends that you use the packaged jars within the RSL ear and not upgrades. The RSL provides updates in related releases that include the updates to these jars

castor-0.9.5.2.jar

This jar contains classes related to the castor subsystem. Payloads for marshalling and un-marshalling between XML and Java Bean representations utilize it. Documentation and related information is located at <http://castor.exolab.org>.

ojdbc14.jar

This jar contains classes related to the Oracle JDBC driver. It is utilized within the conversion utilities in the rsl.jar. This jar file has been updated for JDK 1.4. Documentation and related information is located at http://otn.oracle.com/software/tech/java/sqlj_jdbc/htdocs/jdbc9201.html.

commons-lang-2.0.jar

This jar contains various utility classes for use in math, string, and time operations. Documentation and related information is located at <http://jakarta.apache.org/commons/lang/>.

dom4j.jar

This jar contains XML DOM processing ability. It is needed for processing of the configuration file in xml format. Documentation and related information is located at <http://www.dom4j.org/>.

commons-collections-3.1.jar

This jar contains various utility classes for use with the Java collection API. Documentation and related information is located at <http://jakarta.apache.org/commons/collections/>.

commons-logging.jar and commons-logging-api.jar

These jar contain various utility classes for use with Java Logging. They provide an additional abstraction layer to common logging functionality. Documentation and related information is located at <http://jakarta.apache.org/commons/logging/>.

log4j.jar

This jar contains the log4j subsystem. This is the RSL's logging system of choice and is utilized through the commons-logging.jar. Documentation and related information is located at <http://logging.apache.org/log4j/docs/index.html>.

Other jar files included with the RSL release are required by the Retek Platform.