QPS Programming Guide

This guide shows the steps you must pass through to apply the QoS Provisioning Service (QPS) functionality with any standard CORBA application.

1. The "Hello World" application

The client program invokes a **say_hello** operation on an object in a server program. The server responds by printing "Hello World!" on its standard output.

2. The IDL code

Create a file, which contains the IDL definitions of our application.

```
1   module demos
2   {
3       interface Hello
4       {
5           void say_hello();
6       };
7   };
```

1 Define a module with name "demos" which determine the scope of the Hello application.
3 Interface with name Hello is defined.
5 This interface has only one operation: say_hello.

3. Implementation

3.1 Server implementation

Server implementation looks like:

```
1   package quamj.demos.hello;

2   import quamj.demos.*;

3   public class Hello_impl extends HelloPOA {

4       private org.omg.PortableServer.POA poa_;

5       public Hello_impl(org.omg.PortableServer.POA poa) {
6           poa_ = poa;
7       }

8       public org.omg.PortableServer.POA _default_POA() {
9           if(poa_ != null) {
10              return poa_;
11          } else {
12              return super._default_POA();
13          }
14      }
```

```
15          public void say_hello() {
16              System.out.println("Hello World!");
17          }
18      }
```

---

3 The implementation class Hello_impl must inherit from the generated class HelloPOA.

5-14 Store a reference to the POA where this servant has been created and later re-write the _default_POA CORBA Object method to return the stored POA reference.

15-17 The say_hello operation simply prints “Hello World!” on standard output.

We also have to write a class which holds the server's main() and run() methods. We call this class Server.

```
1       package quamj.demos.hello;
2       import quamj.qps.*;
3       import quamj.demos.*;

4       public class Server {


5           public static void main(String args[]) {

6               java.util.Properties props = System.getProperties();
7               props.put("org.omg.CORBA.ORBClass", "com.ooc.CORBA.ORB");
8               props.put("org.omg.CORBA.ORBSingletonClass",
9                       "com.ooc.CORBA.ORBSingleton");


10              props.put("org.omg.PortableInterceptor.ORBInitializerClass." +
11                      "quamj.qps.QoSORBInitializer_impl", "");

12              System.setProperties(props);

13              int status = 0;
14              org.omg.CORBA.ORB orb = null;

15              try {
16                  orb = org.omg.CORBA.ORB.init(args, props);
17                  status = run(orb, args, props);
18              } catch (Exception ex) {
19                  ex.printStackTrace();
20                  status = 1;
21              }

22              if (orb != null) {
23                  try {
24                      orb.destroy();
25                  } catch (Exception ex) {
26                      ex.printStackTrace();
```

```
27                  status = 1;
28              }
29          }

30          System.exit(status);
31      }
32  }
```

2	*Import the QPS package. You must either import this package, as shown in this example or you must use quamj.qps explicitly.*
7-9	*These properties are necessary to use the ORBacus ORB with JDK 1.2 or later.*
10-11	*QPS implements an ORBIntitializer class (Portable Interceptor specification). In order to use the QPS, that initializer class must be registered with the ORB using that property. When an ORB is initializing, it shall call each registered **ORBInitializer**, passing it an **ORBInitInfo***
16	*The ORB must be initialized using ORB.init.*
17	*Run the helper function run().*
18-21	*This code catches and prints all CORBA exceptions raised by ORB.init() or run().*
22-29	*If the ORB was successfully created, it is destroyed. This releases the resources used by the ORB. If destroy() raises a CORBA exception, this exception is caught and printed. The cast to com.ooc.CORBA.ORB is required when using JDK 1.2, but not when using JDK 1.1 or JDK 1.3.*
30	*The exit status is returned. If there was no error, 0 is returned, or 1 otherwise.*

Now we write the run() method.

```
1   static int run(org.omg.CORBA.ORB orb,
2                  String[] args,
3                  java.util.Properties props)
4       throws org.omg.CORBA.UserException {

5       org.omg.PortableServer.POAManager manager =
6           quamj.qps.plugins.oci_qiop.Util.createPOAManager(
7               "RootPOAManager", orb, args, props);

8       org.omg.CORBA.Object objQOA = orb.resolve_initial_references("QOA");
9       quamj.qps.QOA qoa = quamj.qps.QOAHelper.narrow(objQOA);

10      try {
11          ((quamj.qps.registration.QOA_impl)qoa)._initialization(orb);
12      } catch (quamj.qps.QOAPackage.QOAException qoaEx) {
13          System.out.println("QOA initialization problem" +
14              qoaEx.description + "-- " +  qoaEx.toString());
15      }

16      Hello_impl helloImpl = new Hello_impl(qoa);

17      qoa.register_servant_with_id(helloImpl, "Hello".getBytes());

18      String offered_qos_xml = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>"
19              + "<!DOCTYPE qos SYSTEM \"qml.dtd\">" + "<qos>"
```

```
20                  + "<delay> 130 </delay>" + "<rate> 140 </rate>" + "</qos>";

21          qoa.set_offered_qos_for_id("Hello".getBytes(), offered_qos_xml);

22          org.omg.CORBA.Object hello = qoa.create_reference_with_id(
23              "Hello".getBytes(), "IDL:demos/Hello:1.0");

24          try {
25              String ref = orb.object_to_string(hello);
26              String refFile = "Hello.ref";
27              java.io.FileOutputStream file =
28              new java.io.FileOutputStream(refFile);
29              java.io.PrintWriter out = new java.io.PrintWriter(file);
30              out.println(ref);
31              out.flush();
32              file.close();
33          } catch (java.io.IOException ex) {
34              System.err.println("hello.Server: can't write to `" + ex.getMessage() + "'");
35              return 1;
36          }

37          manager.activate();
38          orb.run();
39          return 0;
40      }
```

5-7 *ORBacus provides a mechanism to plug in a new CORBA transport (in addition to standard IIOP). Here we use a helper function, which initializes the QIOP plug-in and creates the RootPOAManager (the POAManager for the for the RootPOA). This operation must be done prior to resolving the QOA (that constraint is ORBacus specific).*

8-9 *The usual CORBA application uses the ORB to obtain a reference to the RootPOA. Instead we obtain a reference to the QOA. The QOA extends the RootPOA with two operations: 1)an operation to register a servant with user defined object ID; 2) an operation which associates the servant with the QoS it can provide)*

10-15 *So obtained QOA must be explicitly initialized.*
16 *Hello_impl servant is created.*
17 *Use the QOA method to associate the helloImpl servant with user created Object ID (“Hello”).*
18-20 *Create an offered QoS specification as a CORBA string. The format of the specification is defined in a DTD document.*
21 *Associate the QoS specification to the servant using the same object ID – “Hello”.*
22-23 *Use the standard POA method “create_reference_with_id” to create an Object Reference with user defined object id. It is important to use the same id like the one used to register the servant (17).*
24-36 *The object reference is “stringified” and written to a file.*
37-38 *The server enters its event loop to receive incoming requests.*

3.2 Client implementation

```
1   import quamj.qps.*;
2   import quamj.qps.registration.*;

3   public class Client {

4       static int run(org.omg.CORBA.ORB orb, String[] args)
5           throws org.omg.CORBA.UserException {


6           QoSRepository qosRepository = null;
7           try {
8               org.omg.CORBA.Object obj =
9               orb.resolve_initial_references("QoSRepository");
10              qosRepository = QoSRepositoryHelper.narrow(obj);
11              ((QoSRepository_impl)qosRepository)._initialization(orb);
12          } catch (org.omg.CORBA.ORBPackage.InvalidName invNameEx) {
13              System.err.println("Unable to resolve the QoSRepository");
14              return 1;
15          }

16          org.omg.CORBA.Object obj = orb.string_to_object("relfile:/Hello.ref");
17          Hello hello = HelloHelper.narrow(obj);

18          String required_qos_xml = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>"
19                                  + "<!DOCTYPE qos SYSTEM \"qml.dtd\">"
20                                  + "<qos>"
21                                      + "<delay> 140 </delay>"
22                                      + "<rate> 150 </rate>"
23                                  + "</qos>";

24          try {
25              hello = HelloHelper.narrow(
26                  qosRepository.set_required_qos(hello, required_qos_xml));
27          } catch (quamj.qps.QoSRepositoryPackage.RepositoryFault repositoryEx) {
28              System.err.println("QPS is unable to set the required QoS");
29              return 1;
30          }

31          qosRepository.negotiate_qos(hello);

32          hello.say_hello();

33          return 0;
34      }

35      public static void main(String args[]) {
        ……
36      }
37  }
```

---

8 -10 *Obtain a reference to the QoSRepository as a CORBA initial service.*
11 *Perform an explicit initialization of that service.*
16 *The stringified object reference is read and converted to an object.*
17 *Object reference is narrowed to a reference to a Hello object.*

18-23 *XML is used to create a required QoS specification. That specification is validated against a DTD.*

27-33 *The required QoS specification is associated with the Hello object reference using the QoSRepository operation set_required_qos to assign the required QoS with the object reference. The function returns a reference to the same Hello object, which includes the required QoS inside. Then this new reference must be used.*

34 *Use the QoSRepository function to perform an explicit negotiation of the required QoS against the QoS offered by the Server.*

35 *The say_hello operation is invoked. If the QoS negotiation (34) is successful then this invocation will cause the server to print "Hello World" on standard output, else an exception will be thrown.*

38-39 *The code is same as for the Server*

4. Compilation.

Use the build.xml file.

5. Running the application.

5.1 Ensure that any iiop_rsvpd daemon is started per Client and per Server

machines (or run the iiop_rsvpd daemon emulator - quamj\iiop_rsvpd\emulator\run\iiop_rsvpd.bat)

5.2 Deployment configuration

Both the Client and the Sever applications need a deployment configuration file written in XML. The content of this file is validated against a DTD specification provided as a part of the QPS platform (deployment.dtd).

The following document is an XML instance of the **deployment.dtd** format specification

---

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <!DOCTYPE QPS SYSTEM "deployment.dtd">
3   <QPS
4       autoNegotiation="false"
5       negotiationClassName="quamj.qps.plugins.oci_qiop.QoSNegotiator_impl">

6       <PLUGINS>
7           <PLUGIN name="oci_qiop">
8             <CLIENT
9               className=
                    "quamj.qps.plugins.oci_qiop.ClientPerformanceService"
10              rsvpActivationStrategy=
                    "quamj.qps.plugins.oci_qiop.rsvp.IORActivationStrategy"
11              rsvpQoSConversationStrategy=
                    "quamj.qps.plugins.oci_qiop.rsvp.DefaultQoSConversionStrategy"
12              socketFactory="com.kpn.RSVPD.rsvpSocketImplFactory"
13            />
14            <SERVER
15              className="quamj.qps.plugins.oci_qiop.ServerPerformanceService"
16              rsvpActivationStrategy=
                    "quamj.qps.plugins.oci_qiop.rsvp.IORActivationStrategy"
17              rsvpQoSConversationStrategy=
                    "quamj.qps.plugins.oci_qiop.rsvp.DefaultQoSConversionStrategy"
```

```
18                  socketFactory="com.kpn.RSVPD.rsvpSocketImplFactory"
19                  port="0"
20              />
21            </PLUGIN>
22        </PLUGINS>
23    </QPS>
```

---

2 *Set the deployment.dtd as a validation document*

4-5 *These two properties are fixed for release-1*

7-22 *The oci_qiop plug-in configuration properties*

9 *The class name of a class implementing the client plug-in interface. This property is fixed for release-1*

10 *Choose the strategy how to obtain a reference to the IIOP_RSVPD daemon.*

1) *When the RSVP Daemon emulator or the KOM-RSVP Daemon is used this property must be set to: "quamj.qps.plugins.oci_qiop.rsvp.IORActivationStrategy"*
2) *When the W2K RSVP Daemon is used the property must be set to: "quamj.iiop_rsvpd.w2k_rsvp.Daemon"*

11 *There is no one strategy which can be applied to convert the application level QoS specifications (XML's required and offered QoS) into a bottom "rsvp" level. A default conversion mechanism is implemented: "quamj.qps.plugins.oci_qiop.rsvp.DefaultQoSConversionStrategy"*

If you need another conversation strategy. You have to:

1) *implement the interface: (quamj.qps.plugins.oci_qiop.rsvp.QoSConversionStrategyInterface)*
2) *Set the class name of this new implementation as a property value.*

12 *There exist a conflict between windows GQoS and the java sockets implementation. The problem has been solved by custom Java Sockets implementation. Hence If the client or the server are started on a Windows machine and the W2K rsvp daemon is used then that property must by set to: "com.kpn.RSVPD.rsvpSocketImplFactory" Otherwise it must be skipped.*

14-20 *Same as for the Client side (except the line 15 and 19)*

15 *The class name of a class implementing the Server plug-in interface. This property is fixed for release-1*

19 *Set the port number where the server will listen on for new connections. If 0 is set than all ports are allowed.*

Important:

Property	Platform	
	Windows	Linux / Emulator
RsvpActivationStrategy	quamj.iiop_rsvpd.w2k_rsvp.Daemon	quamj.qps.plugins.oci_qiop.rsvp.IORActivation Strategy
SocketFactory	com.kpn.RSVPD.rsvpSocketImplFactory	EMPTY

5.3 Start the Server

- quamj\demos\hello\run\server.bat

5.4 Start the Client

- quamj\demos\hello\run\client.bat