White Paper

Project Persian

Creation and Consumption of High-Performance Web Services in C++

www.roguewave.com

Table of Contents

Abstract

This paper briefly describes Project Persian, Rogue Wave Software's new technology for creating and consuming Web Services in C++. We'll take a look at existing technologies for implementing Web services in C++, compare them with Perisan, and also examine the steps necessary to create, deploy, and consume Web services with Persian.

Introduction

In recent years, many organizations have begun to examine implementing Web services as an interoperability solution for both internal and external integration problems. By using a service-based architecture such as Web services provide, IT departments can more easily create systems that interoperate well and that are maintainable in the future.

There are many tools oriented towards Web services. In nearly every language, libraries or components exist to allow a program to invoke or consume a Web service. Additionally, implementations of enterprise architectures such as .NET, J2EE, and some implementations of CORBA, offer facilities for creating Web services. However, components for Web services in C++ tend to be fairly hard to use. To create Web services from existing C++, these tools often require that C++ be wrapped with another heavyweight middleware product, such as a CORBA ORB (object request broker) implementation.

Project Persian, a key piece of Rogue Wave Software's Web services architecture, enables easy creation and consumption of high-performance Web services in C++, without the overhead of heavyweight middleware. With Persian, developers can quickly take existing C++ code and turn it into a set of Web services without adding additional layers such as a CORBA or JNI (Java Native Interface)wrapper. These Web services are fully interoperable, and can interoperate with code written in any architecture or language. For example, a .NET, J2EE or C++ client could invoke a Persian-based Web Service implemented in C++.

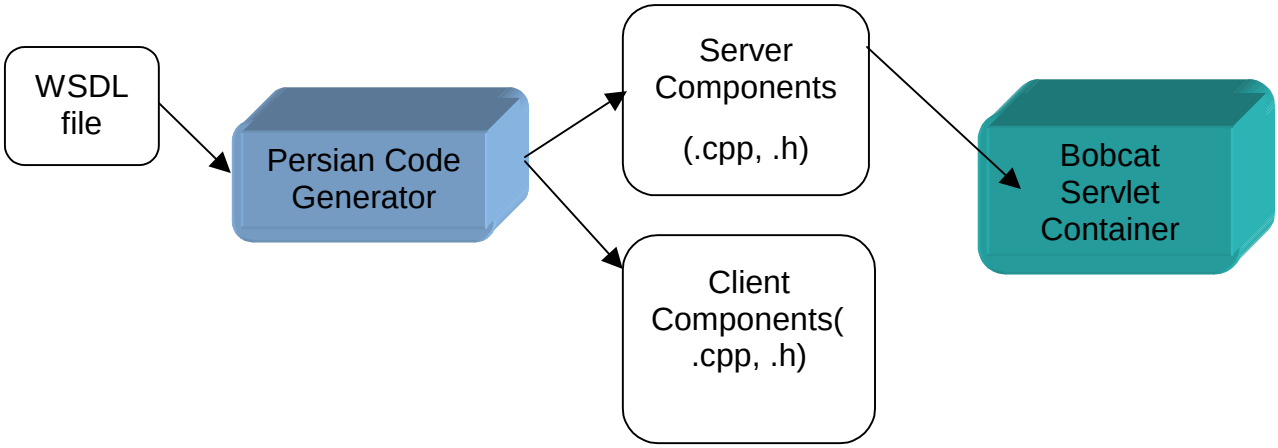
Rogue Wave® Project Persian also enables C++ code to consume other Web services, written in any language, without writing a single line of SOAP or HTTP code. For example, WSDL for a J2EE-based Web service can be given to Persian, and Persian will create a lightweight, cross-platform C++ proxy for that Web service. Persian's code generation technology produces easy-to-use, interoperable code for both the client and server that completely decouples your client and server code from SOAP or HTTP, meaning you don't have to maintain complicated protocol handling code.

Persian provides the most straightforward and direct way to create Web services from C++ code. Instead of needing to wrap existing C++ code in a heavyweight technology such as CORBA or JNI, existing code can be directly turned into a Web service without modification. Additionally, Persian is a great fit in heterogeneous environments because it generates totally interoperable Web services code that can be easily combined with new or existing C++ logic to create high-performance Web services.

Project Persian Overview

Persian enables C++ Web services by combining best-of-breed C++ components with code generation technology. Using a WSDL file as input, Persian creates both client and server components for consuming and creating a Web service that conforms to the given WSDL (Figure 1). These components handle all of the SOAP, WSDL, XML and HTTP details associated with the Web service. By exposing an interface that is free from any of these implementation details, the Persian components make code much more readable and maintainable than with other solutions. Additionally, these components can be regenerated at any time if the Web service changes.

Figure 1- Persian generates Web services components from WSDL



Persian Components

Project Persian is comprised of a set of related technologies to enable Web services in C++. Persian includes:

- **SourcePro™ Net**, a set of libraries built for low-level handling of Internet and Web services protocols, such as HTTP and SOAP. Code generated by Persian uses SourcePro Net internally, but no knowledge of SourcePro Net, or these protocols, is necessary in order to use Persian.
- **Project Bobcat**, the Rogue Wave C++ servlet technology. Persian-generated server components are servlets designed for deployment in Bobcat. Bobcat allows developers to deploy C++ server code in a high-performance, manageable way, in either a standalone configuration or inside nearly any popular web server.

- **Project Ratchet**, the Rogue Wave XML-C++ binding technology. Persian uses Ratchet to generate C++ classes corresponding to complex types that are part of a Web service interface, freeing code from writing and parsing XML directly. The client and server code works with familiar C++ objects and types, and internally are marshaled to and from XML by Ratchet-generated code.

- **Persian Client Generator**, which generates an easy-to-use client proxy from WSDL. Given a WSDL file that defines a Web service, the Persian Client Generator creates a cross-platform proxy class that has straightforward C++ methods corresponding to operations in the Web service. This client proxy may be used in any C++ application as the interface to the Web service and requires no knowledge of SOAP or HTTP.

- **Persian Server Generator,** which generates an easy-to-use servlet base class from WSDL. Given a WSDL file, the Persian Server Generator creates a cross-platform Bobcat servlet that has virtual methods for each operation defined in the WSDL. The developer can then add logic to the Web service simply by subclassing the base class and overriding the virtual methods. Along with the base class, Persian generates a Bobcat deployment descriptor, a sample subclass, and a complete makefile, making it easy to get started creating Web services.

Comparing Persian with CORBA, J2EE and Hand Coding

With these technologies, a solution built with Persian that creates Web services from C++ is far superior to one built with other technologies, such as CORBA and J2EE. Since Persian is a pure C++ technology for implementing Web services, it is far more lightweight and high-performance than other technologies. Also, Persian is built on Rogue Wave's SourcePro libraries, meaning that it generates powerful, easy-to-use classes that are both high-performance and cross-platform.Table 1 shows a comparison of Persian to other industry-standard technologies for creating Web services from C++:

Table 1 - Comparing Persian with Other C++ Web Services Options

	Persian	*CORBA*	*J2EE*	*Hand-Coded*
Easy to learn	**Yes**	No, requires learning a large middleware product	No, requires learning a middleware product and new programming language	No, requires learning and implementing myriad new technologies and protocols
Lightweight	**Yes**	No, requires introducing an ORB (object request broker) into your system	No, requires wrapping C++ in a JNI layer, and then utilizing a J2EE application server for deployment	Yes, if implemented correctly
High-performance	**Yes**	Yes	No, JNI is notoriously slow	Yes, if implemented correctly
Cross-platform	**Yes**	Yes	Yes	Not until ported everywhere
Requires no SOAP or HTTP knowledge	**Yes**	Yes	Yes	No
Maintainable	**Yes**	Yes	Yes	No
Interoperable	**Yes**	Yes	Yes	Not until fully tested

Consuming Web Services in C++

With Persian, you can create easy-to-use proxy classes for existing Web services. As mentioned earlier, given a WSDL file for an existing Web service, Persian generates a C++ proxy class that has a straightforward, intuitive interface for accessing that Web service. Using the proxy class requires no knowledge of SOAP, HTTP, or even Web services – simply invoking a method on the proxy calls the Web service, marshals and sends the request, and receives and unmarshals the response.

To illustrate the use of Persian on the client, this paper will show how to consume existing Web services. Included in the Persian technology preview is a WSDL file for the BabelFish service, a third-party Web service hosted by XMethods (www.xmethods.net). The BabelFish service translates text between languages, for example, from English to Spanish. The WSDL included describes the exact format of SOAP messages expected used for requests and responses with this service.

Hand-coding access to this service is very difficult. First, a programmer must interpret the WSDL, and then write code to properly format a SOAP message corresponding to the message element in the WSDL. Next, the programmer must find or develop HTTP protocol code that sends the created SOAP to the service. The response must be received and parsed with a general-purpose XML parser. Then the developer must write custom code to interpret the response message defined in the WSDL and extract the returned text. Making the system robust would require the development of an error model; and if the WSDL for the service changes, all of the code written would need to be inspected and tested to make sure it still conforms to the interface defined in the WSDL.

With Persian, all of these steps are removed. Since Persian automatically generates C++ classes from WSDL, no hand-coding is necessary. The WSDL file is input to Persian, and the Web service can be immediately used in any C++ program.

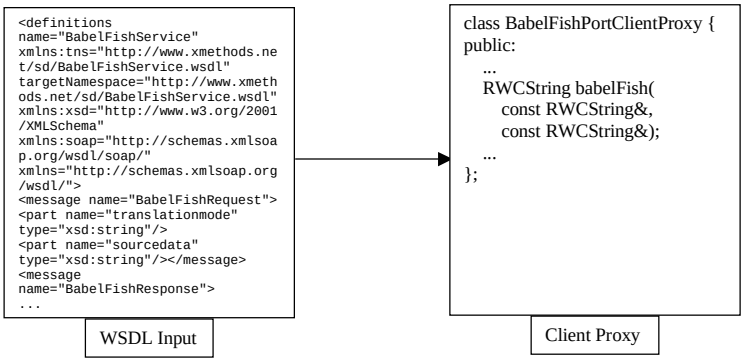


Figure 2 - Persian turns WSDL into C++ Classes

Code Examples

In just a few easy steps, a client proxy can be created and used. To create the proxy for the BabelFish service, Persian's code generator is used, invoked as "wsdl2cpp." In the example, it is invoked with the "-noserver" option, so a server will not be generated for this service (see the next section for more information on creating a server). The following output is from a Windows command prompt:

```
C:\persian\examples\persian\BabelFish>wsdl2cpp -noserver BabelFishService.wsdl
Persian C++ Code Generator
Rogue Wave Software

Generating client...
Generating makefile...
Generating samples...
C:\persian\examples\persian\BabelFish>dir
 Volume in drive C has no label.
 Volume Serial Number is 07D1-080F

 Directory of C:\persian\examples\persian\BabelFish

03/26/2002  04:24p      <DIR>          .
03/26/2002  04:24p      <DIR>          ..
03/26/2002  04:13p               1,657 BabelFishService.wsdl
04/19/2002  02:39p                 685 BabelFishPortClientProxy.h
04/19/2002  02:39p               2,096 BabelFishPortClientProxy.cpp
04/19/2002  02:39p               2,172 makefile_BabelFishPort
04/19/2002  02:39p               1,027 BabelFishPortClient.cpp.sample
               5 File(s)          7,637 bytes
               2 Dir(s)  11,567,939,584 bytes free
```

After running wsdl2cpp, Persian produces a client proxy (`BabelFishPortClientProxy.h` and `.cpp`) and makefile (`makefile_BabelFishPort`), along with a sample usage of the Web service(BabelFishPortClient.cpp.sample). The generated client proxy is an easy-to-use encapsulation of the Web service, and its generated implementation handles all of the details of creating the SOAP request, and interpreting the SOAP response. Here is the interface for the generated BabelFishPortClientProxy class:

```
/**
 * Declaration for Web Service client proxy class BabelFishPortClientProxy
```

```
 * Translates text of up to 5k in length, between a variety of languages.

 * (Proxy Generated by Persian, Rogue Wave Software)

 */

class BabelFishPortClientProxy : public RWWebServiceClient {

public:

    BabelFishPortClientProxy(const RWCString&
location="http://services.xmethods.net:80/perl/soaplite.cgi");



    RWCString babelFish(const RWCString& translationmode_in, const RWCString&
sourcedata_in);



};
```

In this class, the one operation defined in the given WSDL file maps to the `babelFish` method. To use the Web Service, simply create an instance of ***BabelFishPortClientProxy***, and call the desired operation, in this case `babelFish`:

```
#include “BabelFishPortClientProxy.h”

#include <iostream>

using namespace std;


int main()

{

  try {

    BabelFishPortClientProxy b;

    cout << b.babelFish(“en_es”, “Hi there, friend!”) << endl;

  } catch (RWxmsg &x) {

    cout << “Error: “ << x.why() << endl;

  }

  return 0;

}
```

As the previous example shows, in just a few lines of code, one can connect to and invoke the Web service.As seen, the code is completely separate from the HTTP, SOAP and WSDL logic necessary to use the Web service. Errors in invocation of the Web Service are thrown as exceptions deriving from the Rogue Wave base exception class, ***RWxmsg***.

While this Web service is totally functional, average Web services are typically much more complex. Many Web services use data types other than the primitives defined in

XML Schema – they define their own types inside WSDL. Persian supports this because it is based on Project Ratchet, the Rogue Wave XML to C++ binding technology. With Ratchet, Persian is able to generate classes from complex types defined in WSDL.

The following WSDL snippets show an operation that returns a complex type, `WeatherSummary` (taken from the AirportWeather service at capescience.capeclear.com), and the XML Schema definition for this complex type:

```
<types>

<xsd:schema targetNamespace="http://www.capeclear.com/AirportWeather.xsd"

                    xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/"

                    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"

                    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:complexType name="WeatherSummary">

<xsd:sequence>

<xsd:element maxOccurs="1" minOccurs="1" name="location" nillable="true"
type="xsd:string"/>

<xsd:element maxOccurs="1" minOccurs="1" name="wind" nillable="true"
="xsd:string"/>

<xsd:element maxOccurs="1" minOccurs="1" name="sky" nillable="true"
type="xsd:string"/>

<xsd:element maxOccurs="1" minOccurs="1" name="temp" nillable="true"
type="xsd:string"/>

<xsd:element maxOccurs="1" minOccurs="1" name="humidity" nillable="true"
type="xsd:string"/>

<xsd:element maxOccurs="1" minOccurs="1" name="pressure" nillable="true"

type="xsd:string"/>

<xsd:element maxOccurs="1" minOccurs="1" name="visibility"
nillable="true" type="xsd:string"/>

</xsd:sequence>

</xsd:complexType>

</xsd:schema>

</types>

...

<message name="getSummaryResponse">

<part name="return" type="xsd1:WeatherSummary"/>

</message>

...
```

```
<operation name="getSummary">
<input message="tns:getSummary"/>
<output message="tns:getSummaryResponse"/>
</operation>
```

For this service, hand-coding a client would not only require intricate knowledge of SOAP and HTTP to invoke and get useful results from this Web service, but it would additionally require writing generic XML parsing logic to parse the `WeatherSummary` complex type present in the return message. With Persian, all of these details are handled inside of the generated code, meaning that your code doesn't need to directly work with the complex `WeatherSummary` type.

Given the AirportWeather WSDL, Persian generates not only a client proxy for the Web service, but uses Ratchet to generate a C++ class for the `WeatherSummary` type, and uses this class in the applicable method of client proxy. Let's take a look at the declaration for the `StationClientProxy` class generated by Persian for this Web service:

```
/**
 * Declaration for Web Service client proxy class StationClientProxy
 * AirportWeather
 * (Proxy Generated by Persian, Rogue Wave Software)
 */
class StationClientProxy : public RWWebServiceClient {
public:
    StationClientProxy(const RWCString&
location="http://live.capescience.com:80/ccx/AirportWeather");
...
    WeatherSummary getSummary(const RWCString& arg0_in);
...
};
```

Note that the `getSummary()` method returns a `WeatherSummary` instance. `WeatherSummary` is the class automatically generated, corresponding to the `WeatherSummary` complex type defined in the WSDL. The generated class contains `get` and `set` methods corresponding to each of the subelements defined in the WSDL. So for `WeatherSummary`, which has `location`, `wind`, `sky`, `temp`, `humidity`, `pressure`, and `visibility` subelements, the generated class is:

```
class RW_RATCHET_DECLSPEC WeatherSummary {
```

```
public:
...
  RWCString getLocation() const;
  void setLocation(const RWCString& Location);
  RWCString getWind() const;
  void setWind(const RWCString& Wind);
  RWCString getSky() const;
  void setSky(const RWCString& Sky);
  RWCString getTemp() const;
  void setTemp(const RWCString& Temp);
  RWCString getHumidity() const;
  void setHumidity(const RWCString& Humidity);
  RWCString getPressure() const;
  void setPressure(const RWCString& Pressure);
  RWCString getVisibility() const;
  void setVisibility(const RWCString& Visibility);
  ...

};
```

Note how each of the sub-elements corresponds to a `get` and `set` method in the generated class. This straightforward interface makes it easy to use Web services that include complex type definitions, such as this one. Here's some example code that uses the Persian-generated client proxy to obtain a weather summary for KLAX, Los Angeles International Airport, and prints out the summary:

```
StationClientProxy proxy;
WeatherSummary s = proxy.getSummary("KLAX"); // exception thrown on
error.

std::cout << "Weather Summary for " << s.getLocation()
          << "\nTemperature: " << s.getTemp()
          << "\n       Wind: " << s.getWind()
          << "\n        Sky: " << s.getSky()
          << "\n   Humidity: " << s.getHumidity()
```

```
        << "\n   Pressure: " << s.getPressure()
        << "\n Visibility: " << s.getVisibility() << std::endl;
```

Creating Servers from WSDL

In addition to creating client components, Persian can also create server components for implementing Web services in C++. Using WSDL as input, Persian creates a servlet base class that can be derived from, implemented, and deployed in the Bobcat C++ servlet container, also part of Persian.

Persian offers the best choice for creating C++ Web services. Before Persian, the choices were limited. One option was to wrap C++ code in CORBA by purchasing, installing and administering a CORBA ORB that supports Web service, along with writing IDL for the code. Another option was using a J2EE application server by wrapping C++ code in a JNI layer, and exposing it through the Java application server. Both of these options produce working systems, but at the cost of purchasing and learning and using heavyweight systems that aren't really needed to solve the problem. With Persian, creating a Web service in C++ is as simple as writing the WSDL for the service, writing the pure C++ implementation of those services, and then deploying it to the Bobcat C++ servlet container.

Code Example

The first step in creating a Web service is to define its interface by writing WSDL for the Web service. In the below example, a WSDL file is used that is included with the Persian technology download, `DayOfWeek.wsdl`. The WSDL defines a single service port, `DayOfWeekPort`, which has a single operation, `GetDayOfWeek`, which takes a date and returns a string describing which day of the week that date falls on. When we run this WSDL through `wsdl2cpp`, the following files are generated:

```
C:\persian\examples\persian\DayOfWeek>wsdl2cpp DayOfWeek.wsdl

Persian C++ Code Generator

Rogue Wave Software


Generating client...

Generating server...

Generating makefile...

Generating samples...


C:\persian\examples\persian\DayOfWeek>dir

 Volume in drive C has no label.
```

```
 Volume Serial Number is 07D1-080F

 Directory of C:\persian\examples\persian\DayOfWeek

04/19/2002  03:55p      <DIR>          .
04/19/2002  03:55p      <DIR>          ..
04/19/2002  03:55p               1,788 DayOfWeek.wsdl
04/19/2002  03:55p                 609 DayOfWeekPortClientProxy.h
04/19/2002  03:55p               1,354 DayOfWeekPortClientProxy.cpp
04/19/2002  03:55p                 857 DayOfWeekPortServletBase.h
04/19/2002  03:55p               1,565 DayOfWeekPortServletBase.cpp
04/19/2002  03:55p               2,458 makefile_DayOfWeekPort
04/19/2002  03:55p               1,027 DayOfWeekPortClient.cpp.sample
04/19/2002  03:55p                 535 DayOfWeekPortServlet.h.sample
04/19/2002  03:55p                 548 DayOfWeekPortServlet.cpp.sample
04/19/2002  03:55p                 377 DayOfWeekPortServlet_web.xml
              10 File(s)         11,118 bytes
               2 Dir(s)  11,562,663,936 bytes free
```

As noted earlier, a client proxy (`DayOfWeekPortClientProxy.h`, `.cpp`) and client sample (`DayOfWeekPortClient.cpp.sample`) are generated along with a makefile. Additionally, Persian generates a base class for a servlet that implements this Web service (`DayOfWeekServletBase.h`, `.cpp`), along with a sample subclass of this servlet to help developers get started. Once the sample subclass is renamed by removing the "`.sample`" extension, simply edit the existing subclass to easily create the Web service. Here's the relevant portion of the sample subclass's implementation (in `DayOfWeekPortServlet.cpp.sample`):

```
RWCString DayOfWeekPortServlet::getDayOfWeek(const RWDate& date_in)
{
  throw ServerFault("Sorry: the requested operation \"getDayOfWeek\" is not
implemented");
  return RWCString(); // (never executed)
}
```

When the Web service's `GetDayOfWeek` operation is invoked, this method will be automatically called by the generated code that is responsible for marshalling, unmarshalling, and dispatching SOAP messages. To implement the Web service, only fill in the implementation of this method, and then build and deploy the service. Below, the

Web services is implemented with this method and using the `weekDayName()` method of ***RWDate***:

```
RWCString DayOfWeekPortServlet::getDayOfWeek(const RWDate& date_in)
{
  return date_in.weekDayName();
}
```

We'll also rename the client sample similarly, and edit it to invoke the new Web Service. Here is the code that was generated for the client sample:

```
int main()
{
  // Instantiate DayOfWeekPortClientProxy.
  // If you need to override the URL given in the WSDL file for
  // the service, pass it in the constructor like this:
  //   DayOfWeekPortClientProxy proxy("http://www.somehost.net/xyz");

  DayOfWeekPortClientProxy proxy;

  // Use a C++ try block to handle errors that can occur when invoking
  // Web services:
  try {
      // TODO: invoke a method on "proxy"
  }
  catch (RWxmsg &x) {
      std::cerr << "Error invoking web service: "
                << x.why() << std::endl;
  }

  return 0;
}
```

To use the Web service, simply change the line marked //TODO: ... to invoke the desired operation (today's name:)

```
std::cout << proxy.getDayOfWeek(RWDate()) << std::endl;
```

Now, using the included makefile, we can build our servlet and deploy it in one step:

```
C:\persian\examples\persian\DayOfWeek>nmake -f makefile_DayOfWeekPort deploy
```

The "deploy" target of the generated makefile automatically builds the sample client and servlet, and then deploys the servlet to the Bobcat servlet container included with the Persian distribution. Now, we'll start the container in stand-alone mode in one window, and try our client out in another:

```
C:\persian\examples\persian\DayOfWeek>DayOfWeekPortClient

Friday
```

That's it! In those few easy steps, WSDL has been used and turned into a fully-functional, high-performance Web Service, and generated an easy-to-use C++ client for that service.

Conclusion

Web services are an exciting new way to create a service-based architecture for interoperable applications. Using Web services, it is possible to integrate applications and components across language and platform boundaries, using components and systems available today, and without the overhead of large middleware products.

Using the right components and libraries for implementing Web services is key to their success in any project, because the tools chosen will directly impact performance, robustness, and maintainability of the resultant Web services. In C++, a few ways exist for creating Web services, including using CORBA and J2EE wrappers. However, Rogue Wave Project Persian allows existing or new C++ code to be taken easily and directly to the Web in the highest-performance manner, without sacrificing robustness or maintainability. By using Persian's pure C++ solution, developers will not only get C++ code to the Web quicker, but the resultant architecture will be faster, cleaner, more maintainable and more robust than it would be with other technologies. Additionally, using Persian doesn't require the additional learning curve of a new piece of middleware because it is designed from the ground up to enable Web services in C++.

Persian also offers the best way to enable C++ code to work with Web services in any language, making it invaluable for use in mixed-language environments. Because Persian is tested to work with clients and servers in other languages, and fully conforms to XML, HTTP, SOAP and WSDL standards, Persian offers the most direct route to creating truly interoperable systems that work with today's technologies such as .NET and J2EE, and whatever the future may bring.

Download and Participate

Rogue Wave is offering you the opportunity to work with our developers on this exciting new technology. To gain first-hand access to this emerging technology and take part in its development, we encourage you to participate in the Persian evaluation program.

To participate, visit the Technology Access Center at:
http://www.roguewave.com/developer/tac/persian.

Your feedback and evaluation of Persian will help guide its development and implementation.

www.roguewave.com

Corporate Headquarters
Toll-free: (800) 487-3217
E-mail: sales@roguewave.com

The Netherlands
Rogue Wave Software B.V.
Telephone: +31 20 301 26 26
sales@roguewave.nl

Germany
Rogue Wave Software GmbH
Telephone: +49 6103-59 34-0
sales@roguewave.de

France
Rogue Wave Software S.A.R.L.
Telephone: +33 1 41 96 26 26
sales@roguewave.fr

United Kingdom
Rogue Wave Software U.K. Ltd.
Telephone: +44 118 9358600
sales@roguewave.co.uk

Italy
Rogue Wave Software S.R.L.
Telephone: +39 02 4125.081
sales@roguewave.it

Japan
Rogue Wave Software Japan K.K.
Telephone: +81 3 3512-5012
jpinfo@roguewave.com

Asia Pacific
Telephone: +61 2 8923 2515

sales@roguewave.com

Printed in USA
1-PSWP-5/02