Business Grid Components
for
WebSphere Extended Deployment

Development Guide

November 14, 2004

Contents

Overview

This guide describes the activities needed to develop and deploy an application into the Business Grid Components for WebSphere Extended Deployment. Throughout these sections, a sample application – the Mandelbrot application – is used as a concrete example of the information discussed to reinforce the material presented.

1 Application Development

This section describes the development activities needed to integrate an existing native application into the Business Grid Components for WebSphere Extended Deployment.

1.1 Before you begin

The `BGrid.tar.gz` file includes the `BGrid-develop.tar.gz` file which contains all of the artifacts used for client application development and business grid native application development and packaging.

Copy and untar this development tar file on the workstation where you will be doing your client and business grid native application development.

In the following sections, the development process is described as a sequence of steps. The result of those steps result in two components:

- The client application (client app)
- The Business Grid native application (grid app)

Each step below identifies which of these two components it is related (client app or grid app).

The development tar file includes two sample business grid applications:

1. The Mandelbrot sample application (used as the example in this documentation).

   A sample application which includes a web application which allows specification of a number of parameters (e.g. location and size of the rectangle, number of tiles, the pixel dimensions of the generated image,…) and which invokes a native application that generates a PNG image showing the results of applying the Mandelbrot algorithm to a specific rectangles in the complex plane. The Mandelbrot sample is comprised of:

- `MandelbrotSampleApp.ear` (the client app) – a J2EE application used to invoke the backend Mandelbrot grid application
- `MandelbrotGrid.ear` (the grid app) – a .ear file containing the grid application suitable for deployment to the Business Grid

2. The Echo sample application.

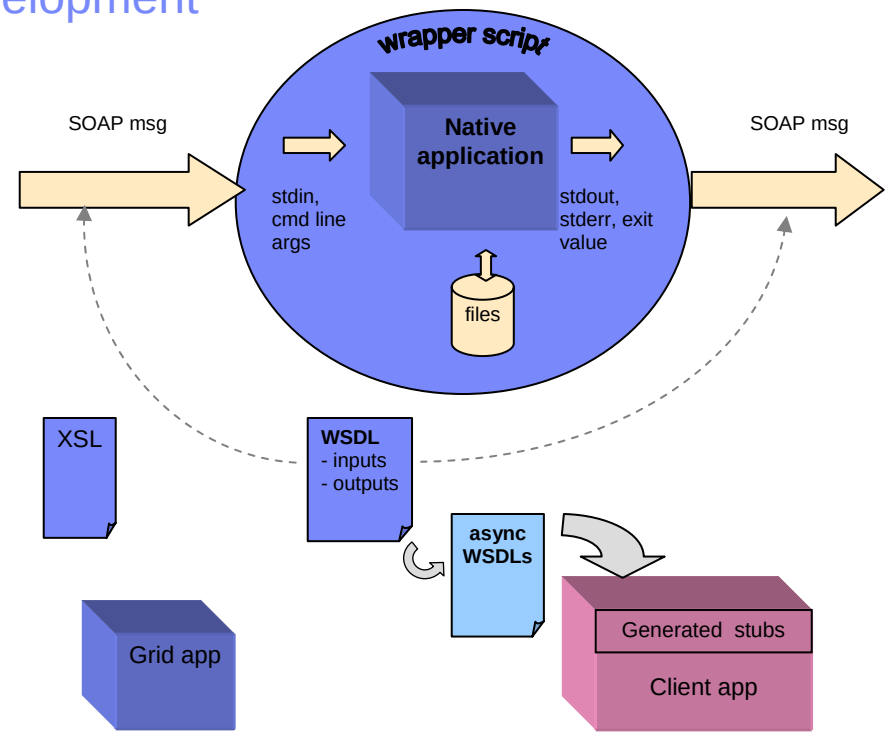
   A sample application which includes a web application which allows specification text and which invokes a simple backend application that echoes the text back as a response. The Echo sample is comprised of:

   - `EchoSampleApp.ear` (the client app) – a J2EE application used to invoke the backend Echo grid application
   - `EchoGrid.ear` (the grid app) – a .ear file containing the grid application suitable for deployment to the Business Grid

1.2 Development Overview

An overview of the development activities is as follows:

Development



- Begin with a native application that is to be run on Business Grid.
- Native application takes input from stdin, command line arguments and files and produces output as files, stdout, stderr and an exit value.
- Formally describe inputs and outputs in WSDL document.
- Construct wrapper script that accepts a SOAP message on stdin and produces a SOAP message on stdout. SOAP messages conform to WSDL.
- Use Business Grid tooling to generate equivalent asynchronous WSDL document(s).
- Use standard tooling to construct client application based on asynchronous WSDL document(s).
- Construct XSL transform to transform SOAP request message to scheduler job.
- Use Business Grid tooling to package native executable, XSL transform and wrapper script as a grid application.

1.3 Native application (grid app)

The development process begins with an existing native application which is to be enabled to run in the Business Grid Components for WebSphere Extended Deployment.

In general, the characteristics of this native application are an executable (or script) which:

- takes input from standard input (stdin), command line arguments and/or files
- produces output as files, standard output (stdout), standard error (stderr) and an exit value

Mandelbrot sample: For the Mandelbrot sample, the native application is a fairly simple C application that generates a PNG image showing the results of applying the Mandelbrot algorithm to a specific rectangle in the complex plane. The location and size of the rectangle, the pixel dimensions of the generated image, filename for the resulting image and maximum number of iterations of the Mandelbrot algorithm are specified as command line parameters. As the Mandelbrot algorithm is not especially taxing to current hardware, an additional parameter was added to cause the program to repeat the calculations a specified number of times to simulate more intense workloads. Here is the usage output from the application:

```
$ ./bgmandel --help
Usage: ./bgmandel <args>
where args can be any of the following:

   -h <horiz_samples>  number of horizontal samples to evaluate
   -v <vert_samples>   number of vertical samples to evaluate
   -i <max_iterations> maximum number of iterations per pixel
   -r <repeat_count>   repeats computation specified number of times
   -minr <min_real>    minimum real value to evaluate
   -maxr <max_real>    maximum real value to evaluate
   -mini <min_imag>    minimum imaginary value to evaluate
   -maxi <max_imag>    maximum imaginary value to evaluate
   -png <fname>        filename for the generated image
```

1.4 WSDL document (pertains to both the client app and grid app)

The business grid architecture specifies that we expose the function of this application as a service or set of services. The first step in the process is to formally describe the application functionality (i.e. the inputs and outputs) that should be exposed as services. In web services, this formal description is contained in a WSDL document. From the client perspective of this application (which is how WSDL documents are stated), the interaction with the service behaves like a traditional request/response.

Mandelbrot sample: The WSDL document for the Mandelbrot native application follows.

```
<?xml version="1.0" encoding="UTF-8"?>

<!--

This file contains the WSDL description of the bgmandel application.
It describes the processing done by the application as a
request/response web service.

-->

<wsdl:definitions name="BGMandelbrot"
  targetNamespace="http://tempuri.org/BGMandelbrot/"
```

```
  xmlns:bgmandel="http://tempuri.org/BGMandelbrot/"
  xmlns:bgrid="http://www.ibm.com/websphere/business-grid/2004/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  >

  <wsdl:types>
    <xsd:schema elementFormDefault="qualified"
      targetNamespace="http://tempuri.org/BGMandelbrot/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">

      <!-- A complex type containing the parameters needed by the
           bgmandel application. -->
      <xsd:complexType name="MandelbrotParmsType">
        <xsd:sequence>
          <xsd:element name="horizontalSamples" type="xsd:positiveInteger"/>
          <xsd:element name="verticalSamples" type="xsd:positiveInteger"/>
          <xsd:element name="maximumIterations" type="xsd:positiveInteger"/>
          <xsd:element name="repeat" type="xsd:positiveInteger"/>
          <xsd:element name="minimumRealValue" type="xsd:double"/>
          <xsd:element name="maximumRealValue" type="xsd:double"/>
          <xsd:element name="minimumImaginaryValue" type="xsd:double"/>
          <xsd:element name="maximumImaginaryValue" type="xsd:double"/>
        </xsd:sequence>
      </xsd:complexType>

      <!—- An element containing the input parameters. -->
      <xsd:element name="MandelbrotParms" type="bgmandel:MandelbrotParmsType">
      </xsd:element>

      <!—- An element containing the result output by the bgmandel application.
-->
      <xsd:element name="MandelbrotResult">
        <xsd:complexType>
          <xsd:sequence>
            <!—- The input parameters contained on the request. -->
            <xsd:element name="MandelbrotParms"
              type="bgmandel:MandelbrotParmsType"/>
            <!—- A string specifying which host the PNG image was generated. -
->
            <xsd:element name="generatedBy" type="xsd:string"/>
            <!—- The generated PNG image. -->
            <xsd:element name="mandelbrotPNG" type="xsd:base64Binary"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
```

```
    </xsd:schema>
  </wsdl:types>

  <!—- The request message. -->
  <wsdl:message name="GenerateMandelbrotRequest">
    <wsdl:part name="generateMandelbrotParms"
element="bgmandel:MandelbrotParms">
    </wsdl:part>
  </wsdl:message>

  <!—- The response message. -->
  <wsdl:message name="GenerateMandelbrotResult">
    <wsdl:part name="mandelbrotResultParms"
element="bgmandel:MandelbrotResult">
    </wsdl:part>
  </wsdl:message>

  <!—- The fault message. -->
  <wsdl:message name="IncorrectParameters">
  </wsdl:message>


  <!—- The port type specifying the abstract operations and messages -->
  <wsdl:portType name="MandelbrotImageGenerator">
    <wsdl:operation name="generateMandelbrot">
      <wsdl:input message="bgmandel:GenerateMandelbrotRequest"
        name="generateParameters">
      </wsdl:input>

      <wsdl:output message="bgmandel:GenerateMandelbrotResult"
        name="generateResult">
      </wsdl:output>

      <wsdl:fault message="bgmandel:IncorrectParameters"
        name="incorrectParameters">
      </wsdl:fault>
    </wsdl:operation>
  </wsdl:portType>


  <!—- The binding specifying the message format and protocol details -->
  <wsdl:binding name="MandelbrotSOAPBinding"
    type="bgmandel:MandelbrotImageGenerator">

    <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http"/>
```

```
    <wsdl:operation name="generateMandelbrot">
      <soap:operation
        soapAction="http://tempuri.org/BGMandelbrot/generateMandelbrot"/>

      <wsdl:input name="generateParameters">
        <soap:body use="literal" parts="generateMandelbrotParms"/>
      </wsdl:input>

      <wsdl:output name="generateResult">
        <soap:body use="literal" parts="mandelbrotResultParms"/>
      </wsdl:output>

    </wsdl:operation>
  </wsdl:binding>

  <!—- The service specification -->
  <wsdl:service name="MandelbrotService">
    <wsdl:port name="MandelbrotSOAPPort"
      binding="bgmandel:MandelbrotSOAPBinding">
      <soap:address location="http://localhost:9080/not/used"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

1.5 Wrapper script (grid app)

The next step is to construct an application wrapper script that accepts a SOAP message that conforms to the WSDL on its standard input. The wrapper script is responsible for parsing this request and invoking the native application with the necessary parameters to perform the requested actions.

Although the wrapper script is free to parse the incoming XML SOAP message in whatever fashion it wishes, XSL stylesheets have proven to be a useful tool for performing this parsing. For example, if your wrapper script is a shell script, RedHat Linux comes with a tool called xsltproc that can apply XSL stylesheets to XML documents. Other scripting (e.g. PERL) or programming languages (e.g. C or C++) also have xsl processing libraries which can be used.

Once processing of the request is complete, the wrapper script must generate a SOAP message containing the results of the processing. The message may indicate a fault if there was a problem performing the requested action. The generated SOAP message must conform to the WSDL.

Again, XSL stylesheets and the xsltproc utility (or other appropriate xsl processing libraries) can be used to generate the response SOAP message.

Mandelbrot sample: The script for the Mandelbrot native application includes the main script plus an xsl file to convert the request parameters in the message to the command-line format expected by the native application.

.

bgmandelwrapper.sh

```
#!/bin/bash

# figure out where this script is located
MYDIR=$(dirname "$0")
MYDIR=$(cd "$MYDIR"; pwd)
XSLDIR="$MYDIR/xsl"

# locations of temporary files used to hold interim data
TMP=/tmp
TMPPREFIX="$TMP/bgmandelwrapper-$$"
TMPINPUT="$TMPPREFIX-input.xml"
TMPERROR="$TMPPREFIX-error.txt"
TMPOUTPUTPNG="$TMPPREFIX-output.png"


# capture the incoming SOAP request message and convert the request
# parms in the message to the command-line format expected by the
# native application
ARGS=$(cat >"$TMPINPUT"; xsltproc "$XSLDIR/convert-args.xsl" "$TMPINPUT")


# invoke the native application
echo "$MYDIR/bgmandel" $ARGS -png "$TMPOUTPUTPNG" >&2
"$MYDIR/bgmandel" $ARGS -png "$TMPOUTPUTPNG" 2>"$TMPERROR"
NATIVERC=$?

cat "$TMPERROR" >&2


# generate the SOAP response message to stdout
cat <<EOF
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:bgmandel="http://tempuri.org/BGMandelbrot/"
  >

  <soap:Body>
EOF


# if the native application returned 0, build a normal response
# message
if [ $NATIVERC = 0 ]
then
    cat <<EOF
```

```
    <bgmandel:MandelbrotResult>
      <bgmandel:MandelbrotParms>
EOF

    # include a copy of the request parameters
    echo "$ARGS" | sed \
      -e '/^[  ]*$/d' \
      -e 's#[  ]*-h[     ][   ]*\([0-9][0-9]*\)#
<bgmandel:horizontalSamples>\1<\/bgmandel:horizontalSamples>#' \
      -e 's#[  ]*-v[     ][   ]*\([0-9][0-9]*\)#
<bgmandel:verticalSamples>\1<\/bgmandel:verticalSamples>#' \
      -e 's#[  ]*-i[     ][   ]*\([0-9][0-9]*\)#
<bgmandel:maximumIterations>\1<\/bgmandel:maximumIterations>#' \
      -e 's#[  ]*-r[     ][   ]*\([0-9][0-9]*\)#
<bgmandel:repeat>\1<\/bgmandel:repeat>#' \
      -e 's#[  ]*-minr[  ][   ]*\([0-9.-][0-9.]*\)#
<bgmandel:minimumRealValue>\1<\/bgmandel:minimumRealValue>#' \
      -e 's#[  ]*-maxr[  ][   ]*\([0-9.-][0-9.]*\)#
<bgmandel:maximumRealValue>\1<\/bgmandel:maximumRealValue>#' \
      -e 's#[  ]*-mini[  ][   ]*\([0-9.-][0-9.]*\)#
<bgmandel:minimumImaginaryValue>\1<\/bgmandel:minimumImaginaryValue>#' \
      -e 's#[  ]*-maxi[  ][   ]*\([0-9.-][0-9.]*\)#
<bgmandel:maximumImaginaryValue>\1<\/bgmandel:maximumImaginaryValue>#' \


    cat <<EOF
      </bgmandel:MandelbrotParms>
      <bgmandel:generatedBy>
EOF

    # set generatedBy to the host name of this machine
    uname -n

    cat <<EOF
      </bgmandel:generatedBy>
      <bgmandel:mandelbrotPNG>
EOF

    # include a uuencoded copy of the generated PNG image
    uuencode -m junk <"$TMPOUTPUTPNG" | sed -e '1d' -e '$d'

    cat <<EOF
      </bgmandel:mandelbrotPNG>
    </bgmandel:MandelbrotResult>
EOF

# if the native application returned non-zero, build a fault response
# message
else
    cat <<EOF
    <soap:Fault>
      <faultcode>soap:Client</faultcode>
      <faultstring>
EOF
```

```
    # include the stderr output from the native application
    cat "$TMPERROR"

    cat <<EOF
      </faultstring>
      <detail>
      </detail>
    </soap:Fault>
EOF

fi

cat <<EOF
  </soap:Body>
</soap:Envelope>
EOF


# remove temporary files
rm -f "$TMPINPUT" "$TMPERROR" "$TMPOUTPUTPNG"


# propagate return code from the native application
exit $NATIVERC
```

convert-args.xsl

```
<?xml version="1.0"?>

<!--
  This stylesheet converts the information in a bgmandel:MandelbrotParms
  element to the command-line argument format expected by the bgmandel
  application.
-->

<xsl:stylesheet version="1.0"
  id="bgmandelwrap-parseinput"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:bgmandel="http://tempuri.org/BGMandelbrot/"
  >

  <xsl:output method="text"/>

  <!-- we only care about the MandelbrotParms elements -->
  <xsl:template match="/">
    <xsl:apply-templates
select="/soap:Envelope/soap:Body/bgmandel:MandelbrotParms/*"/>
```

```
  </xsl:template>

  <!-- horizontalSamples -> -h -->
  <xsl:template match="bgmandel:horizontalSamples">
    -h <xsl:value-of select="."/>
  </xsl:template>

  <!-- verticalSamples -> -v -->
  <xsl:template match="bgmandel:verticalSamples">
    -v <xsl:value-of select="."/>
  </xsl:template>

  <!-- maximumIterations -> -i -->
  <xsl:template match="bgmandel:maximumIterations">
    -i <xsl:value-of select="."/>
  </xsl:template>

  <!-- repeat -> -r -->
  <xsl:template match="bgmandel:repeat">
    -r <xsl:value-of select="."/>
  </xsl:template>

  <!-- minimumRealValue -> -minr -->
  <xsl:template match="bgmandel:minimumRealValue">
    -minr <xsl:value-of select="."/>
  </xsl:template>

  <!-- maximumRealValue -> -maxr -->
  <xsl:template match="bgmandel:maximumRealValue">
    -maxr <xsl:value-of select="."/>
  </xsl:template>

  <!-- minimumImaginaryValue -> -mini -->
  <xsl:template match="bgmandel:minimumImaginaryValue">
    -mini <xsl:value-of select="."/>
  </xsl:template>

  <!-- maximumImaginaryValue -> -maxi -->
  <xsl:template match="bgmandel:maximumImaginaryValue">
    -maxi <xsl:value-of select="."/>
  </xsl:template>

</xsl:stylesheet>
```

1.6 Generating Client WSDL documents (client app)

With native applications, it is not uncommon for computationally intensive work to require many minutes or even hours to complete. To support this interaction, Business Grid provides tooling to generate multiple client WSDL documents supporting both HTTP and JMS as the transport protocols for SOAP messages. Business Grid supplies stylesheets and other artifacts to generate Client WSDL documents by invoking xsltproc on these stylesheets.

1.6.1 SOAP messages over HTTP

SOAP over HTTP has the advantage of making the service available to the widest range of client types. The HTTP protocol and web application servers, however, are not designed to deal with requests that take significant amounts of time to generate responses. Given these constraints, it is necessary to model the request and the response as asynchronous services using one of the following invocation models:

- two separate message exchanges (Asynchronous request/response)

  with Asynchronous request/response, the request is sent in as a one-way SOAP over HTTP input message from the client to the Business Grid gateway. The response is later sent as a one-way SOAP over HTTP input message from the Business Grid gateway to a web service provided by the client and specified on the request as where the response should be sent (see 1.7.1 for details on this mechanism)

- a polling model where the request is sent and the response is retrieved by polling (Asynchronous request/polling).

  With Asynchronous request/polling, the request is sent in as a one-way input message from the client to the Business Grid gateway (like Asynchronous request/response). However, the response is retrieved by the client issuing a request-response operation where the input message specifies this is a polling request and the response is contained in the output message.

For Asynchronous request/response, the following stylesheets are provided:

- synchronous-to-request-wsdl.xsl (used to generate the one-way request WSDL)
- synchronous-to-response-wsdl.xsl (used to generate the one-way response WSDL)

For Asynchronous request/polling, the following stylesheet is provided:

- synchronous-to-poll-wsdl.xsl (used to generate the one-way request and the polling request-response WSDL)

Mandelbrot sample: For example, to generate the one-way request WSDL and one-way response WSDL for the Asynchronous request/response model for the Mandelbrot application (Mandelbrot.wsdl), issue the following commands:

xsltproc --output mandelbrot-request.wsdl synchronous-to-request-wsdl.xsl mandelbrot.wsdl

xsltproc --output mandelbrot-response.wsdl synchronous-to-response-wsdl.xsl mandelbrot.wsdl

The output of these commands will be two new asynchronous WSDLs (mandelbrot-request.wsdl and mandelbrot-response.wsdl) which can then be used to generate a web-service client for the asynchronous request and a web-service to receive the asynchronous response.

Note: the following files contained in the developer's wsdl-transforms directory are required for the processing of these transformations:

- addressing.xsd
- bgrid.wsdl
- wsdl-util.xsl

1.6.2 SOAP messages over JMS

SOAP over JMS is inherently asynchronous providing a more suitable messaging mechanism for Business Grid applications. While it is no yet standardized and hence is not as interoperable, SOAP over JMS offers more reliable and scalable messaging support than SOAP over HTTP. When using SOAP over JMS, the request and response flows are modeled as:

- two separate message exchanges (Asynchronous request/response)

  with Asynchronous request/response, the request is sent in as a one-way SOAP over JMS input message from the client to the Business Grid gateway. The response is later sent as a one-way SOAP over JMS input message from the Business Grid gateway to a jms queue specified on the request as where the response should be sent (see 1.7.1 for details on this mechanism). The response is retrieved by the client by making a synchronous call where the queue to be read is specified as an input argument and the output arguments are returned as a result of reading the SOAP response off of the specified queue.

For Asynchronous request/response, the following stylesheet is provided:

- synchronous-to-async-jms.wsdl

Mandelbrot sample: For example, to generate the one-way request and request-response WSDL for the SOAP over JMS Asynchronous request/response model for the Mandelbrot application (Mandelbrot.wsdl), issue the following commands:

xsltproc --output mandelbrot-asyncjms.wsdl synchronous-to-async-jms.xsl mandelbrot.wsdl

The output of this command will be a new asynchronous WSDL (mandelbrot-asyncjms.wsdl) which can then be used to generate a web-service client for the asynchronous request and to retrieve the asynchronous response.

Note: the following files contained in the developer's wsdl-transforms directory are required for the processing of these transformations:

- addressing.xsd
- bgrid.wsdl

- wsdl-util.xsl

1.7 Constructing the Client Application (client app)

Once the appropriate client WSDLs have been generated, the client application may be constructed from these WSDLs. Standard tooling can be used to aid in constructing the application, such as IBM WebSphere Studio Application Developer (WSAD) or the Apache AXIS WSDL2Java tool.

Each of the generated WSDLs can be used to generate Web Service Clients except for the HTTP response WSDL which should be used to generate a Web Service to receive the SOAP over HTTP response message.

1.7.1 Programming Considerations

Use of WS-Addressing

WebSphere Business Grid supports multiple protocols (HTTP and JMS) in interacting with the native application via SOA. To support this interaction using a transport-neutral mechanism, WS-Addressing is utilized to provide a set of standardized SOAP headers over these transports. The WS-Addressing specification is available at http://www-106.ibm.com/developerworks/library/specification/ws-add/

The table below lists the WS-Addressing header elements that need to be present in the request and the purpose of each. These header elements are presented as parameters when the Client WSDLs are used to generate web-service code.

WS-Addressing header	Purpose
Action	Identifies the XSL transform for the application that corresponds to the request. The Action should begin with the Business Grid URI scheme 'bgrid:'
To	Identifies the host:port of one of the servers in the Dynamic Cluster where the grid app has been deployed
MessageID	Client-defined correlator that will be included in the response message. Must be expressed as a URI.
ReplyTo, FaultTo	Specifies the web service endpoints to which the gateway should send responses and faults. For Asynchronous request/response model ReplyTo is required. If FaultTo is not specified, faults will be sent to ReplyTo. For Asynchronous request/polling model, ReplyTo and FaultTo are not specified.

The table below summarizes the WS-Addressing headers that are included in the response SOAP message.

WS-Addressing header	Purpose
Action	Present but not used
To	Identifies the endpoint that will receive the response. Copied from either the ReplyTo or FaultTo header in the request.
RelatesTo	Client-defined correlator copied from the MessageID in the request.

1.7.1.1 SOAP over JMS Clients

When standard tooling is used to generate a web-service client from the transformed client WSDL documents, a java Service Locator source file is created. For SOAP over JMS Clients, this source file must be modified by adding the following new constructor:

```
    public
ClientAppServiceAsyncJmsLocator(com.ibm.ws.webservices.engine.EngineConfigurat
ion config) {
        super(config);
    }
```

The SOAP over JMS Client depends on the bgridjms transport to send and receive messages with the gateway using JMS. The following code must be added to the client application to initialize the bgridjms transport, deploy it into the web service config and instantiate the Service Locator using this config object (for both request and reply stubs) for each instance of the client stubs which will be used by the application (each set of stubs may be used to drive multiple requests and responses).

```
     com.ibm.ws.webservices.engine.client.Connection.setTransportForProtocol("
bgridjms", com.ibm.ws.bgrid.protocol.bgridjms.BGridJmsTransport.class);


     com.ibm.ws.webservices.engine.configuration.SimpleEngineConfigurationProv
ider config = new
com.ibm.ws.webservices.engine.configuration.SimpleEngineConfigurationProvider(
);
     config.deployTransport("bgridjms", new
com.ibm.ws.webservices.engine.SimpleTargetedChain(new
com.ibm.ws.bgrid.protocol.bgridjms.BGridJmsSender()));
     org.tempuri.ClientAppServiceAsyncJmsLocator loc = new
org.tempuri.ClientAppServiceAsyncJmsLocator(config);
```

Mandelbrot sample: The modified Service Locator code for the Mandelbrot SOAP over JMS Client application follows:

```
/**
 * MandelbrotServiceAsyncJmsLocator.java
 *
 * This file was auto-generated from WSDL
 * by the IBM Web services WSDL2Java emitter.
 * cf20411.06 v32504192757
 */

package org.tempuri;

public class MandelbrotServiceAsyncJmsLocator extends
com.ibm.ws.webservices.engine.client.Service implements
org.tempuri.MandelbrotServiceAsyncJms {

    // Use to get a proxy class for mandelbrotSOAPPortAsyncJms
    private final java.lang.String mandelbrotSOAPPortAsyncJms_address =
"http://localhost:9080/not/used";

    public MandelbrotServiceAsyncJmsLocator() {
        super();
    }

    // following constructor added for bgrid SOAP over JMS
```

```
    public
MandelbrotServiceAsyncJmsLocator(com.ibm.ws.webservices.engine.EngineConfigura
tion config) {
        super(config);
    }
  :
  :
}
```

Here's some example client code from the Mandelbrot sample which initializes the bgridjms transport, deploys it into the web service config and instantiates the Service Locator using this config object:

```
          URL mandelbrotServiceURL;
            MandelbrotImageGeneratorAsyncJms migaj;
            org.tempuri.MandelbrotImageGeneratorAsyncJms migajTemp = null;
            MandelbrotImageGeneratorRequest migr;
            MandelbrotImageGeneratorRequest migrTemp = null;

com.ibm.ws.webservices.engine.client.Connection.setTransportForProtocol(
                                  "bgridjms",
com.ibm.ws.bgrid.protocol.bgridjms.BGridJmsTransport.class);
                              mandelbrotResponses = new
MandelbrotResponses(gatewayQueue, replyToQueue);
                              mandelbrotResponses.start();

com.ibm.ws.webservices.engine.configuration.SimpleEngineConfigurationProvider
config =
                     new
com.ibm.ws.webservices.engine.configuration.SimpleEngineConfigurationProvider(
);
                  config.deployTransport("bgridjms", new
com.ibm.ws.webservices.engine.SimpleTargetedChain(new
com.ibm.ws.bgrid.protocol.bgridjms.BGridJmsSender()));
                  MandelbrotServiceAsyncJmsLocator msajl = new
org.tempuri.MandelbrotServiceAsyncJmsLocator(config);
```

Here's some example code from the Mandelbrot sample which sets the Mandelbrot parameters and the needed header values, and invokes the request.

```
             MandelbrotParmsType mpt = new MandelbrotParmsType();

               mpt.setHorizontalSamples(new
                   PositiveInteger(String.valueOf(horizontalSamples)));
               mpt.setVerticalSamples(new
                   PositiveInteger(String.valueOf(verticalSamples)));
               mpt.setMaximumIterations(new
```

```
                PositiveInteger(String.valueOf(maximumIterations)));
            mpt.setRepeat(new PositiveInteger(String.valueOf(repeat)));

            mpt.setMinimumRealValue(minimumRealValue);
            mpt.setMaximumRealValue(maximumRealValue);
            mpt.setMinimumImaginaryValue(minimumImaginaryValue);
            mpt.setMaximumImaginaryValue(maximumImaginaryValue);

            EndpointReferenceType ertReplyTo = new EndpointReferenceType();
            ertReplyTo.setAddress(new AttributedURI(replyToQueue));

            AttributedURI auToURL = new AttributedURI(serviceURI.toString());
            AttributedURI auActionURI = new
AttributedURI("bgrid:MandelbrotGrid/Mandelbrot/" + "BGMandelbrot.xsl");
            AttributedURI auMessageID = new AttributedURI("mandel:" + id);
            mandelbrotServiceURL = new URL(gatewayQueue);

            migaj = msajl.getMandelbrotSOAPPortAsyncJms(mandelbrotServiceURL);
            migaj.generateMandelbrot(auActionURI, auToURL, ertReplyTo,
                                     ertReplyTo, auMessageID, mpt);
```

And finally, some Mandelbrot sample application code which sets the parameters and invokes the service to retrieve the asynchronous jms response. In this call, the replyToURL specifies the queue information that bgridjms transport should use to retrieve the one-way SOAP response from the Bgrid gateway.

```
            URL mandelbrotServiceURL = new URL(gatewayQueue);

             MandelbrotImageGeneratorAsyncJms migaj =
mrsl.getMandelbrotSOAPPortAsyncJms(mandelbrotServiceURL);

             EndpointReferenceType ertReplyTo = new EndpointReferenceType();
             ertReplyTo.setAddress(new AttributedURI(replyToURL));

             AttributedURIHolder auToURIHolder = new AttributedURIHolder();
             AttributedURIHolder auActionURIHolder = new
AttributedURIHolder();
             org.xmlsoap.schemas.holders.RelationshipHolder relatesToHolder =
new org.xmlsoap.schemas.holders.RelationshipHolder();
             org.tempuri.holders.MandelbrotResultHolder
mandelbrotResultHolder = new org.tempuri.holders.MandelbrotResultHolder();
             try
             {
                 migaj.generateMandelbrot(ertReplyTo,
                                          auActionURIHolder,
                                          auToURIHolder,
                                          relatesToHolder,
```

```
                                                mandelbrotResultHolder);
                    System.out.println("got a response:  relatesToParm=" +
relatesToHolder.value + ", generatedBy=" +
mandelbrotResultHolder.value.getGeneratedBy());
                }
                catch (Exception e)
                {
                     System.out.println("MandelbrotResponses.getMandelbrotResult
Exception:" + e);
                }
           }
```

1.8 XSL transform (grid app)

When the gateway receives the SOAP message with the service request, it uses an application-provided XSL stylesheet to transform that request into a Load Leveler job.

The stylesheet is applied to the entire SOAP envelope, so all SOAP header and body elements are available during processing. The output of the transform is submitted directly to Load Leveler with no further processing by the gateway. Refer to chapters 6 and 12 of the Load Leveler "Using and Administering" guide for a detailed discussion of Load Leveler job syntax.

In your XSL stylesheet, the following parameters should be specified:

- define a parameter named clientRespURL. Your stylesheet must place this value inside the WS-Addressing ReplyTo field.
- define a parameter named faultRespURL. Your stylesheet must place this value inside the WS-Addressing FaultTo field.
- define a parameter named bgridExecute. Your stylesheet must place this value as the name of the script to invoke, passing the native application wrapper as a command-line parameter.

Mandelbrot sample: The xsl transform for the Mandelbrot application is as follows:

```
<!--
  This stylesheet converts an incoming SOAP request for the Mandelbrot
  sample application to a Load Leveler job.
-->

<xsl:stylesheet version="1.1"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing"
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  >

  <!-- HTML is the closest output type to a job file that contains XML -->
```

```
  <xsl:output method="html"/>


  <!-- parameters passed to this script by the Business Grid gateway -->
  <xsl:param name="clientRespURL">http://tempuri.org</xsl:param>
  <xsl:param name="faultRespURL"></xsl:param>
  <xsl:param name="bgridExecute"></xsl:param>


  <!-- matches the root element of the SOAP request message -->
  <xsl:template match="/">

    <!-- a Load Leveler job shell script with some generic job parameters -->
    <xsl:text>#!/bin/sh

#@ class = small
#@ notification = error
#@ output = /tmp/wrapper.job.stdout
#@ error = /tmp/wrapper.job.stderr
</xsl:text>

    <!-- let the Business Grid wrapper script take care of parsing the
         WS-Addressing headers and sending the SOAP response message itself
         and have it invoke our wrapper script to handle the actual
application
         execution -->
    <xsl:value-of select="$bgridExecute"/>
    <xsl:text disable-output-escaping="yes"> -v ./bgmandelwrapper.sh
&lt;&lt;EOF
&lt;?xml version="1.0" encoding="utf-8"?&gt;
</xsl:text>

    <!-- put a modified copy of the SOAP message we're looking at now into
         the job for later examination -->
    <xsl:apply-templates select="node() | @*"></xsl:apply-templates>

    <!-- mark the end of both the XML and the job -->
    <xsl:text>
EOF

#@queue
</xsl:text>
  </xsl:template>


  <!-- All of the following templates are used to modify the SOAP message that
       is placed in the job. -->
```

```
  <!-- except as overridden below, deep copy everything -->
  <xsl:template match="node() | @*">
    <xsl:copy>
      <xsl:apply-templates select="@* | node()"/>
    </xsl:copy>
  </xsl:template>


  <!-- special processing for soapenv:Header -->
  <xsl:template match="soapenv:Header">
    <xsl:copy>

      <!-- except as suppressed below, copy the existing elements -->
      <xsl:apply-templates select="@* | node()"/>

      <!-- override wsa:To and wsa:Action -->
      <wsa:To>not:used</wsa:To>
      <wsa:Action>not:used</wsa:Action>

      <!-- use wsa:ReplyTo and wsa:FaultTo values supplied by the gateway -->
      <wsa:ReplyTo>
        <wsa:Address>
          <xsl:value-of select="$clientRespURL"/>
        </wsa:Address>
      </wsa:ReplyTo>

      <wsa:FaultTo>
        <wsa:Address>
          <xsl:value-of select="$faultRespURL"/>
        </wsa:Address>
      </wsa:FaultTo>
    </xsl:copy>
  </xsl:template>


  <!-- templates for WS-Addressing headers that we DON'T want to pass on -->
  <xsl:template match="wsa:ReplyTo"></xsl:template>
  <xsl:template match="wsa:FaultTo"></xsl:template>
  <xsl:template match="wsa:Action"></xsl:template>
  <xsl:template match="wsa:To"></xsl:template>

</xsl:stylesheet>
```

1.9 Assembling the Business Grid native application (grid app)

Once all of the native application artifacts have been developed, WebSphere Business Grid provides a script to package the native application as a J2EE .ear file suitable for deployment.

An assemblegridapp.sh script is provided with the developer's tar file and is used to assemble the native executable, the wrapper script, the XSL transform and other supplied files required by the Business Grid native application.

Note: assemblegridapp.sh requires a jar executable on its PATH which supports the update option (-u option). The jar executable which ships with WebSphere (*WAS_Install*_Dir\java\bin) supports update, or alternatively the IBM JDK can be downloaded to provide this jar.

assemblegridapp.sh has the following arguments:

```
$ ./bin/assemblegridapp.sh  --help

./bin/assemblegridapp.sh --appname <app_name> --ear <ear_filename> (--url
<relative_url>)+ (--xsl <xsl_filename>)+ [--appsetup <appsetup_filename>]
<app_file>*
```

where:

--appname <app_name> the name of the application used in naming various artifacts of the grid app, such as the display name and the name of the generated .war file

--ear <ear_filename> the name of the generated .ear file which is output as a result of running this script

--url <relative_url> The --url parameter specifies URLs that will be logically mapped to the grid application. You should omit the protocol when specifying the URL; for example, "--url foo/bar". For the first URL specified, everything before the first '/' character is used as the context root for the generated EAR file. For all the URLs, everything after the first '/' character is mapped to a dummy servlet in the generated WAR file. For example, if you specify "--url foo/bar --url abc/def/ghi", the EAR file's context root will be "foo" and "bar/*" and "def/ghi/*" will be mapped to a dummy servlet in the WAR file. When the client application sends a request to the Business Grid gateway, it specifies one of these URLs in the WS-Addressing Action header, along with the name of the grid application XSL file that should process the request. Continuing the previous example, a client application could specify an Action of "bgrid:foo/bar/UpdateData.xsl" to cause the gateway to process the request using the grid application's UpdateData.xsl transform.

--xsl <xsl_filename> the name of the XSL transform file(s) for this grid application

--appsetup <appsetup_filename> the name of a script which will be run the first time the application is invoked. Normally this script is used to make the grid application program files executable (i.e. chmod +x). This script is always run from the root user id.

<app_file>* additional files to be included in the grid executable. These files normally include the grid application native executable file, the wrapper script for the application, and any other scripts or xsl transformation files needed by the application.

Mandelbrot sample: For an example of invocation of the assemblegridapp.sh script to build the Mandelbrot grid application, see the top level ant build.xml contained in the developer's tar file for the Mandelbrot sample application (the dist-gridapp2 target within this build.xml builds the Mandelbrot grid application).

2 Deployment Considerations

2.1 SOAP over JMS Clients

The SOAP over JMS Client depends on the bgridjms transport to send and receive messages with the gateway using JMS. The WebSphere Business Grid provided BGridJmsTransport.jar file must be deployed as a URL provider and made available on the classpath of the client container's JVM.

The following steps may be followed when the client container is a WebSphere Application Server:

- **Add the transport code to WebSphere**.

Place BGridJmsTransport.jar in the Client's *WebSphere_Install_Root*\lib\ext directory

- **Define a new URL Provider to the app server using the WebSphere Administrative Console**:

Resources->URL Providers
Name: BGridJmsTransport
Handler class: com.ibm.ws.bgrid.protocol.bgridjms.Handler
Protocol: bgridjms

2.2 Load Leveler

Many applications will require changes to the Load Leveler runtime environment. Such changes might include the definition of new job classes or resources that are used by the application's jobs. These Load Leveler changes will

need to be performed manually by the deployer when the application is deployed into a new WebSphere Business Grid environment.

Grid applications can use MPI (Message Passing Interface) for communication between application components. See the Load Leveler documentation for information on MPI libraries which Load Leveler supports.

3 Sample Messages

This section provides messages that flow during execution of a grid app using the Mandelbrot sample application.

3.1 Client to Business Grid Gateway

The following SOAP message flows to the WebSphere Business Grid Gateway when the Mandelbrot Sample application is invoked with default values specifying JMS as the transport

```
<soapenv:Envelope
    xmlns:soapenv=http://schemas.xmlsoap.org/soap/envelope/
    xmlns:soapenc=http://schemas.xmlsoap.org/soap/encoding/
    xmlns:xsd=http://www.w3.org/2001/XMLSchema
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Header>
        <Action xmlns="http://schemas.xmlsoap.org/ws/2003/03/addressing">
            bgrid:MandelbrotGrid/Mandelbrot/BGMandelbrot.xsl
        </Action>
        <To xmlns="http://schemas.xmlsoap.org/ws/2003/03/addressing">
            http://bgrid.dyn.webahead.ibm.com:9082
        </To>
        <ReplyTo xmlns="http://schemas.xmlsoap.org/ws/2003/03/addressing">
            <Address>
jms:/queue?destination=jms/BGGWOUTQ&amp;connectionFactory=jms/BGGWQCF&amp;jndi
ProviderURL=iiop://bgrid00.dyn.webahead.ibm.com:9811/
            </Address>
        </ReplyTo>
        <FaultTo xmlns="http://schemas.xmlsoap.org/ws/2003/03/addressing">
            <Address>
jms:/queue?destination=jms/BGGWOUTQ&amp;connectionFactory=jms/BGGWQCF&amp;jndi
ProviderURL=iiop://bgrid00.dyn.webahead.ibm.com:9811/
            </Address>
        </FaultTo>
        <MessageID xmlns="http://schemas.xmlsoap.org/ws/2003/03/addressing">
mandel:1100526083656-0
        </MessageID>
    </soapenv:Header>
    <soapenv:Body>
        <MandelbrotParms xmlns="http://tempuri.org/BGMandelbrot/">
            <horizontalSamples>200</horizontalSamples>
```

```
            <verticalSamples>200</verticalSamples>
            <maximumIterations>75</maximumIterations>
            <repeat>5</repeat>
            <minimumRealValue>-2.5</minimumRealValue>
            <maximumRealValue>-1.5</maximumRealValue>
            <minimumImaginaryValue>0.5</minimumImaginaryValue>
            <maximumImaginaryValue>1.5</maximumImaginaryValue>
        </MandelbrotParms>
    </soapenv:Body>
</soapenv:Envelope>
```

3.2 Business Grid Gateway to Load Leveler / Grid application script

Based on the input message for the Mandelbrot application, the following job file is constructed and submitted to Load Leveler. The bgjobwrapper script in turn extracts the SOAP message from the job file and invokes the application script (bgmandelwrapper.sh) with this SOAP message as standard input.

```
#!/bin/sh

#@ class = small
#@ notification = error
#@ output = /tmp/wrapper.job.stdout
#@ error = /tmp/wrapper.job.stderr

/opt/WebSphere/AppServer/bgrid/bin/bgjobwrapper.sh -v ./bgmandelwrapper.sh
<<EOF

<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header>
    <MessageID xmlns="http://schemas.xmlsoap.org/ws/2003/03/addressing">
      mandel:1100526083656-0
    </MessageID>
    <wsa:To
      xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing">
        not:used
    </wsa:To>
    <wsa:Action xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing">
      not:used
    </wsa:Action>
    <wsa:ReplyTo xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing">
      <wsa:Address>
jms:/queue?channel=SYSTEM.DEF.SVRCONN/TCP&amp;host=localhost(1415)&amp;queueNa
me=ENDPT.REPLYQ&amp;queueManager=bgridept.queue.manager
      </wsa:Address>
    </wsa:ReplyTo>
```

```
    <wsa:FaultTo xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing">
      <wsa:Address>
jms:/queue?channel=SYSTEM.DEF.SVRCONN/TCP&amp;host=localhost(1415)&amp;queueNa
me=ENDPT.REPLYQ&amp;queueManager=bgridept.queue.manager
      </wsa:Address>
    </wsa:FaultTo>
  </soapenv:Header>
  <soapenv:Body>
    <MandelbrotParms xmlns="http://tempuri.org/BGMandelbrot/">
      <horizontalSamples>200</horizontalSamples>
      <verticalSamples>200</verticalSamples>
      <maximumIterations>75</maximumIterations>
      <repeat>5</repeat>
      <minimumRealValue>-0.5</minimumRealValue>
      <maximumRealValue>0.5</maximumRealValue>
      <minimumImaginaryValue>0.5</minimumImaginaryValue>
      <maximumImaginaryValue>1.5</maximumImaginaryValue>
    </MandelbrotParms>
  </soapenv:Body>
</soapenv:Envelope>
EOF

#@queue
```

3.3 Application script / BGrid wrapper script to Business Grid Gateway

The standard output of the native grid application is directed to a file. Once the native application completes the bgjobwrapper script takes the output SOAP message contained in the file, fixes up the SOAP headers based on the header received on the input SOAP message, and sends the following SOAP message back to the BGrid Gateway.

```
<soap:Envelope
    xmlns:soap=http://schemas.xmlsoap.org/soap/envelope/
    xmlns:bgmandel="http://tempuri.org/BGMandelbrot/">
    <soap:Header
        xmlns:bgrid=http://www.ibm.com/websphere/business-grid/2004/
        xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing">
        <wsa:Action>bgridctl:response/ClientResponse</wsa:Action>
        <wsa:To>
jms:/queue?channel=SYSTEM.DEF.SVRCONN/TCP&amp;host=localhost(1415)&amp;queueNa
me=ENDPT.REPLYQ&amp;queueManager=bgridept.queue.manager
        </wsa:To>
        <wsa:RelatesTo> mandel:1100526083656-0</wsa:RelatesTo>
        <bgrid:ExecutionTime>3227</bgrid:ExecutionTime>
    </soap:Header>
    <soap:Body>
      <bgmandel:MandelbrotResult>
        <bgmandel:MandelbrotParms>
          <bgmandel:horizontalSamples>200</bgmandel:horizontalSamples>
          <bgmandel:verticalSamples>200</bgmandel:verticalSamples>
          <bgmandel:maximumIterations>75</bgmandel:maximumIterations>
```

```
          <bgmandel:repeat>5</bgmandel:repeat>
          <bgmandel:minimumRealValue>-1.5</bgmandel:minimumRealValue>
          <bgmandel:maximumRealValue>-0.5</bgmandel:maximumRealValue>
          <bgmandel:minimumImaginaryValue>0.5</bgmandel:minimumImaginaryValue>
          <bgmandel:maximumImaginaryValue>1.5</bgmandel:maximumImaginaryValue>
        </bgmandel:MandelbrotParms>
        <bgmandel:generatedBy>
bgrid04.dyn.webahead.ibm.com
        </bgmandel:generatedBy>
      <bgmandel:mandelbrotPNG>
iVBORw0KGgoAAAANSUhEUgAAAMgAAADICAIAAAAiOjnJAAALX0lEQVR4nO2d
T4gU2R2Av5esLnR17wqxhenADmJoFVvveBp184SQg5rhJxW9xKQjLdEcC9
DuphySWChFzG4NHZnHc8bsx4XHAu
        [3725 characters omitted]
3hixyvpGK2hSzJxK7ZW6CVZNqKKG
Nk7WVTubmXscNtBtWP1YqXsAF3f2He5jFf/thtQ3ykC2U78k2az6qMPiYCrb
ZgqrmhD9m6gDdVgwD4RdAeiybdVu/g859WdzSuKg8wAAAABJRU5ErkJggg==
      </bgmandel:mandelbrotPNG>
    </bgmandel:MandelbrotResult>
  </soap:Body>
</soap:Envelope>
```

3.4 Business Grid Gateway to Client

The Business Grid Gateway then forwards the SOAP response message to the originating client using the information on the ReplyTo SOAP header element of the original request as the destination.

```
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:bgmandel="http://tempuri.org/BGMandelbrot/">
  <soap:Header
    xmlns:bgrid="http://www.ibm.com/websphere/business-grid/2004/"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing">
    <wsa:Action>
      http://tempuri.org/
    </wsa:Action>
    <wsa:To>
jms:/queue?destination=jms/BGGWOUTQ&connectionFactory=jms/BGGWQCF&jndiProvider
URL=iiop://bgrid00.dyn.webahead.ibm.com:9811/
    </wsa:To>
    <wsa:RelatesTo>mandel:1100526083656-0</wsa:RelatesTo>
    <bgrid:ExecutionTime>3227</bgrid:ExecutionTime>
  </soap:Header>
  <soap:Body>
    <bgmandel:MandelbrotResult>
      <bgmandel:MandelbrotParms>
        <bgmandel:horizontalSamples>200</bgmandel:horizontalSamples>
        <bgmandel:verticalSamples>200</bgmandel:verticalSamples>
        <bgmandel:maximumIterations>75</bgmandel:maximumIterations>
        <bgmandel:repeat>5</bgmandel:repeat>
```

```
        <bgmandel:minimumRealValue>0.5</bgmandel:minimumRealValue>
        <bgmandel:maximumRealValue>1.5</bgmandel:maximumRealValue>
        <bgmandel:minimumImaginaryValue>0.5</bgmandel:minimumImaginaryValue>
        <bgmandel:maximumImaginaryValue>1.5</bgmandel:maximumImaginaryValue>
      </bgmandel:MandelbrotParms>
      <bgmandel:generatedBy>
bgrid04.dyn.webahead.ibm.com
      </bgmandel:generatedBy>
      <bgmandel:mandelbrotPNG>
iVBORw0KGgoAAAANSUhEUgAAAMgAAADICAIAAAAiOjnJAAALX0lEQVR4nO2d
T4gU2R2Av5esLnR17wqxhenADmJoFVvveBp184SQg5rhJxW9xKQjLdEcC9C
DuphySWChFzG4NHZnHc8bsx4XHAu
[3725 characters omitted]
3hixyvpGK2hSzJxK7ZW6CVZNqKKG
Nk7WVTubmXscNtBtWP1YqXsAF3f2He5jFf/thtQ3ykC2U78k2az6qMPiYCrb
ZgqrmhD9m6gDdVgwD4RdAeiybdVu/g859WdzSuKg8wAAAABJRU5ErkJggg==
      </bgmandel:mandelbrotPNG>
    </bgmandel:MandelbrotResult>
  </soap:Body>
</soap:Envelope>
```

IBM Corporation
Software Group
Route 100
Somers, NY 10589
U.S.A.

Produced in the United States of America
11-04