IBM WebSphere Business Connection

Web Services Technical Reference

Version 1.1.1

Note!

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 41.

Second Edition (December 2002)

This edition applies to Version 1, Release 1, Modification 1, of *IBM® WebSphere® Business Connection* (5724-D26) and to all subsequent releases and modifications until otherwise indicated in new editions.

IBM welcomes your comments. You can send them to the following address:

IBM Canada Ltd. Laboratory
Information Development
8200 Warden Avenue
Markham, Ontario, Canada L6G 1C7

Include the title and order number of this book, and the page number or topic related to your comment.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

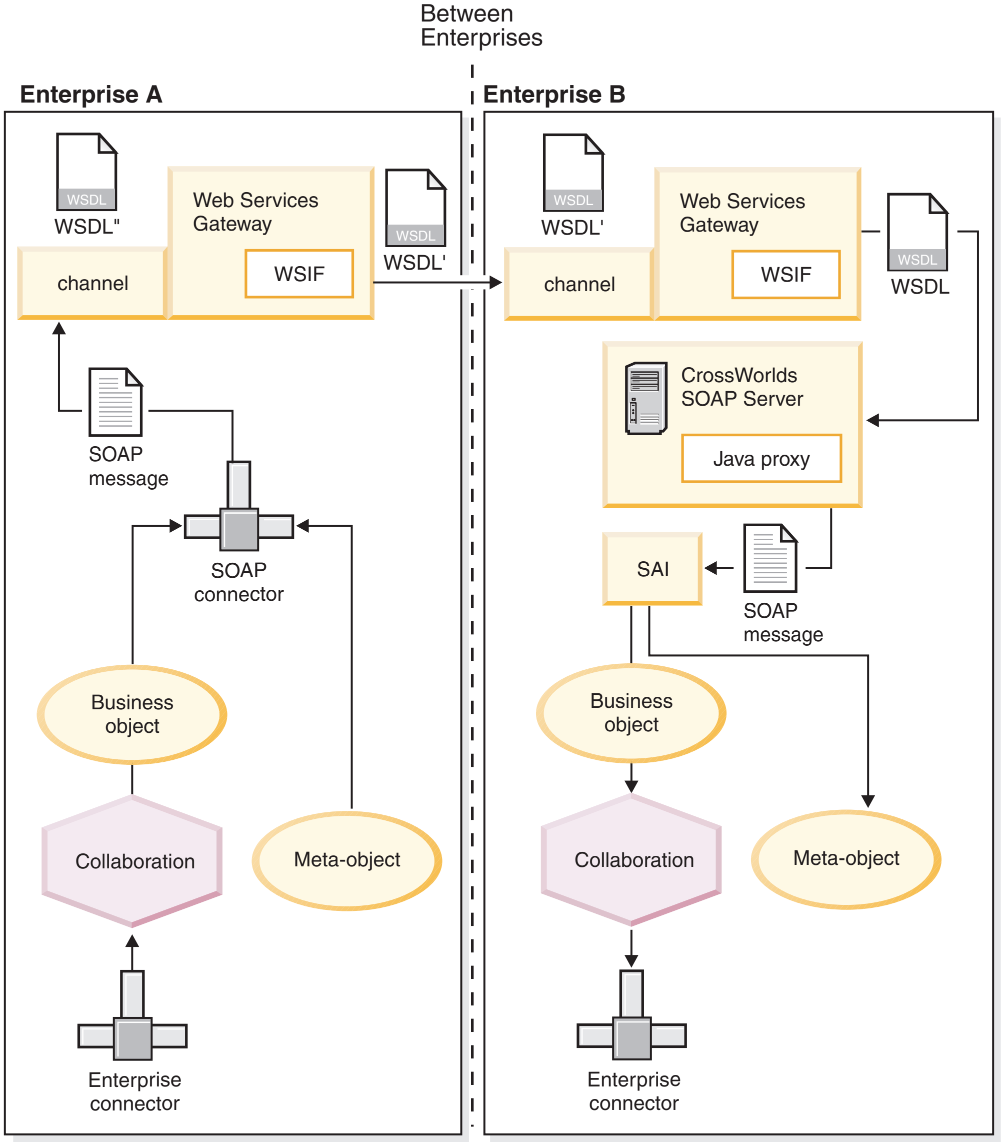
Contents

Web Services Technical Reference

This document provides you with technical information to help you develop Web services on a Windows(R) platform. The document describes the relationship between CrossWorlds(R) artifacts, provides details of the naming conventions for document-style and RPC-style messaging, and describes the CWGenUtility. It provides a review of topics and expands on topics covered in earlier Web Services documentation.

A review of communication flow

The following figure represents two enterprises, each with a CrossWorlds instance. Communication is flowing between the CrossWorlds instances.



Flow between two enterprises using CrossWorlds

In an actual production environment, message flows will start in Enterprise A and flow to Enterprise B. However, for the source connector in Enterprise A to send out a message, a collaboration has to be defined in Enterprise A. Similarly, for the SOAP connector in Enterprise A to send a message, the Web Services Gateway A has to be set up, and for Web Services Gateway A to be set up, Web Services Gateway B has to be set up, and so forth all the way to Target Connector B.

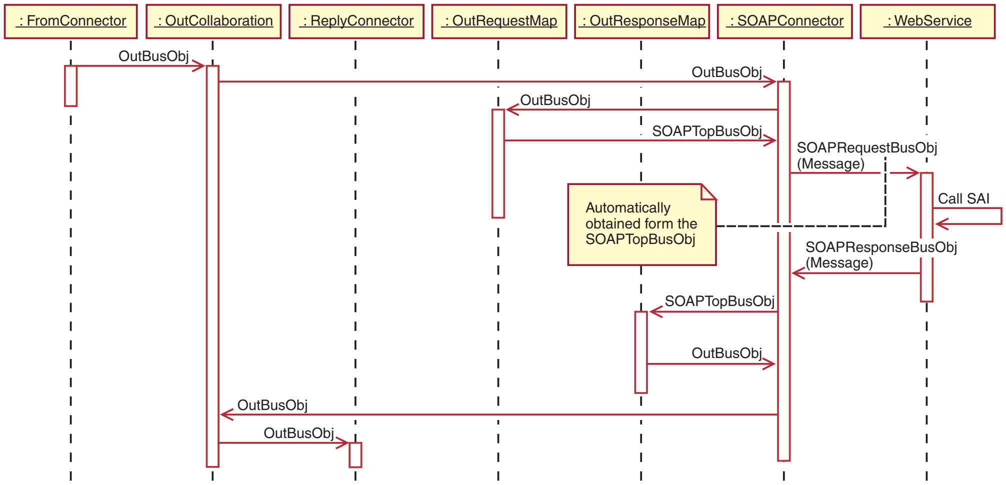
For a message to flow from A towards B, therefore, the upstream parts (starting with Source Connector A) have a dependency on the downstream parts, all the way to Target Connector B. Development, then, has to start on B and propagate to A, as follows:

1. **On Enterprise B:**
   a. Define business objects.
   b. Define maps for the request and response (success/fault).
   c. Define a collaboration template.
   d. Define a collaboration object and bind it to ports.
   e. Define meta-objects (which are used by the data handler).
2. **On Enterprise B:**
   a. Run WSGenUtility to create a Web-service Java(TM) proxy and WSDL.
   b. Create a deployment descriptor for the Web-service Java proxy.
   c. Deploy the Web-service Java proxy and deployment descriptor into a SOAP-enabled application server (for example, WebSphere(R) Application Server).
3. **On Enterprise B**, deploy the WSDL into the Web Services Gateway, which creates WSDL'.
4. **On Enterprise A**, deploy the WSDL' into Enterprise A.
5. **On Enterprise A:**
   a. Deploy the business objects from Enterprise B into CrossWorlds.
   b. Deploy the client meta-objects from Enterprise B into CrossWorlds. (Create the meta-objects if they have not already been created.)
   c. Configure the SOAP connector top-level meta-objects.

Flow overview

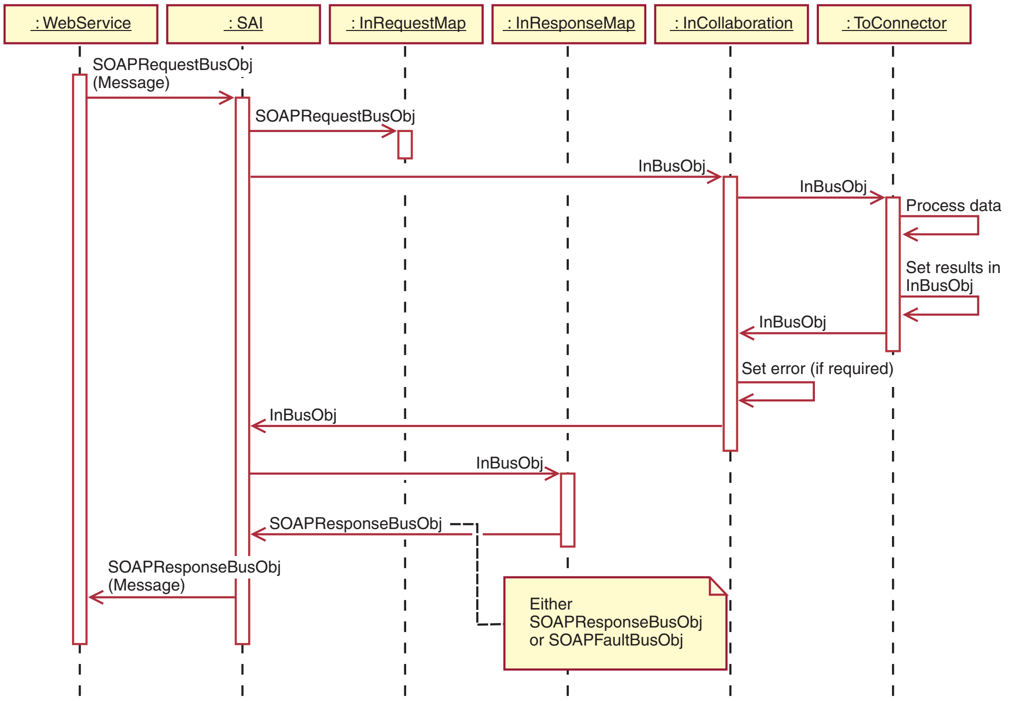
The information in the samples is based on the Outbound and Inbound sequence diagrams. These are generic diagrams that are common to all flows.

The following diagram shows the outbound flow to and from the SOAP connector. The flow of the samples was described in Web Services Overview and Samples.



Outbound flow to and from the SOAP connector

The following diagram shows the inbound flow. The flow of the samples was described in Web Services Overview and Samples.



Inbound flow through the SAI

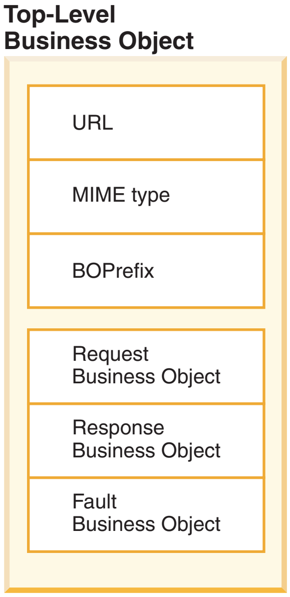
Object relationships

The previous section showed the outbound flow from a SOAP connector and the inbound flow through an SAI. This section describes in detail the relationships between the objects that are part of that flow, including the relationship between:

- A SOAP application-specific business object and a Document-style general business object
- A SOAP application-specific business object and an RPC-style application-specific business object
- A SOAP connection meta-object and the object and data of a business object
- An SAI meta-object and the object and data of a business object

SOAP-object-to-generic-business-object relationship (Document style)

A top-level SOAP application-specific business object is required when a business object is communicating with the SOAP connector. This SOAP top-level object has a particular structure that mimics the parts of a SOAP message, as shown in the following illustration:



Top-level business object structure

SOAP messages consist of a:

- Request - The message being sent
- Response - The reply to the request
- Fault - A special type of response containing errors that occurred while the request was processed

Normally, for communication between CrossWorlds to CrossWorlds, the generic business object that is sent can be assigned to the Response and Request attributes of the SOAP top-level application-specific business object. If Web Services Gateway is being used, though, this will not work because the Web Services Gateway uses RPC style, which can return only a single object. Because a generic business object may contain multiple attributes, these attributes have to be encapsulated into a single object.

To create a single object, you create a wrapper SOAP application-specific business object. This object contains a single attribute called child, which contains the generic business object.

Note: The attribute does not have to be called *child*.

If SOAP headers are required, such as for routing or authentication in Web Services Gateway, additional attributes in the SOAP Request application-specific business object are required to represent the header information. The attribute, which can be any name, contains a generic business object with the structure that mimics the XML structure of the header. Pre-defined header generic business objects are provided with the samples for use with the Web Services Gateway Routing and Authentication filters.

SOAP-object-to-generic-business-object relationships (RPC style)

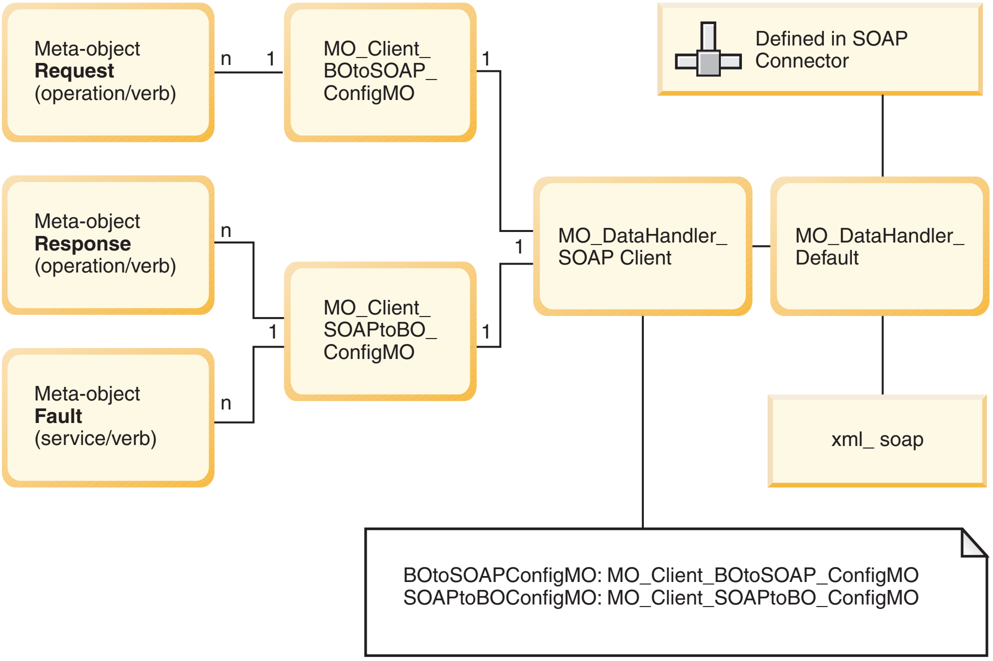
As you learned in Web Services Advanced Topics, for RPC calls, each method input parameter and return type needs to be represented by an application-specific business object. Those business objects are:

- The parameter application-specific business objects are encapsulated by the SOAP application-specific business object Request.
- The return application-specific business object is encapsulated by the SOAP application-specific business object Response.
- The SOAP Fault application-specific business object contains the same attributes as a normal SOAP Fault message. The SOAP Fault detail attribute is optional.

Normally the Request parameter application-specific business objects would be associated with the Detail attribute. If so and if there is more than one parameter, the application-specific business object needs to be a superset of all the parameters. The limitation is that none of the attribute names within the parameters can be the same.

SOAP connector meta-object relationships

Meta-objects are used by the SOAP connector data handler to marshal and un-marshal the SOAP XML messages. The Request, Response, and Fault meta-objects need to be created for each SOAP application-specific business object being sent to the SOAP connector. This relationship is shown in the following illustration:



How the data handler marshals and unmarshals SOAP messages

Meta-objects are based on the business object and verb combination. The business objects are the ones associated with the SOAP top-level application-specific business object Request, Response and Fault attributes.

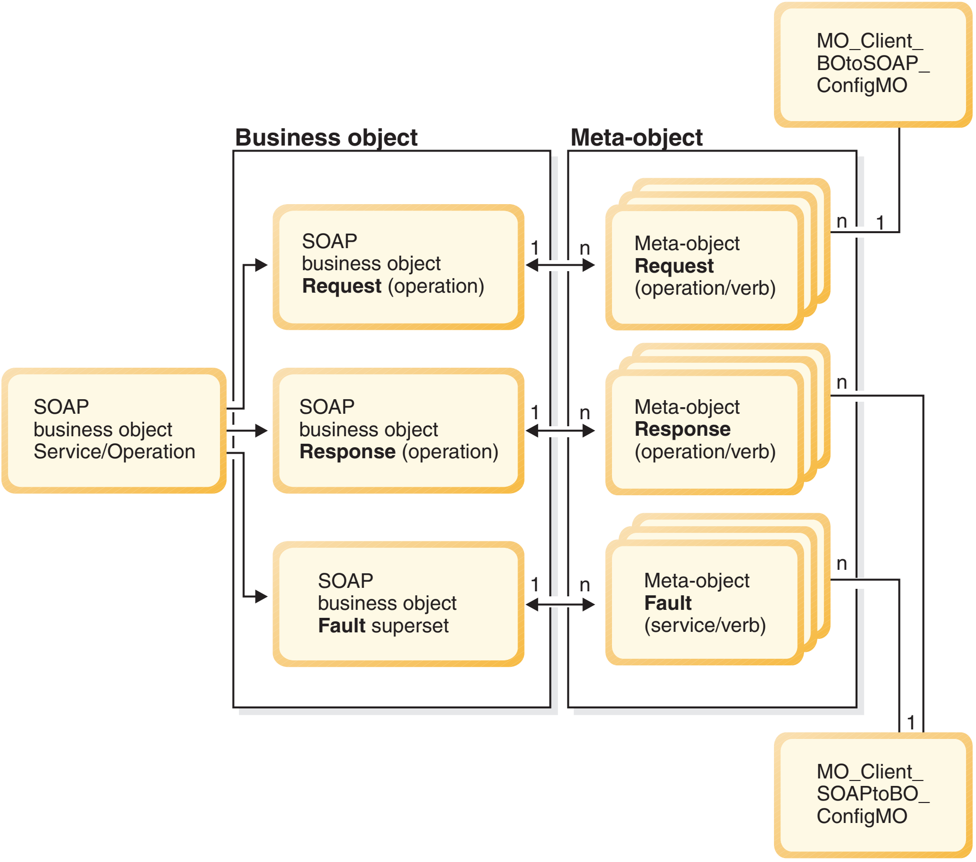
SOAP Connector - meta-object to business object relationships

This section describes the object relationships and data relationships in SOAP connector operations.

SOAP object relationships

Each SOAP application-specific business object representing the Request, Response and Fault attributes of the SOAP top-level application-specific business object requires a corresponding SOAP meta-object. These meta-objects have a fixed structure and are used by the CrossWorlds data handlers for marshalling and un-marshalling the SOAP application-specific business objects to and from SOAP XML.

Meta-objects are specific to the name of the business object plus the verb. Each combination of SOAP application-specific business object and verb requires a meta-object. Meta-objects are grouped by requests and responses. Requests go out from CrossWorlds through the SOAP connector; the replies to the requests are the responses. Responses are returned from the SOAP connector into CrossWorlds, as shown in the following illustration:

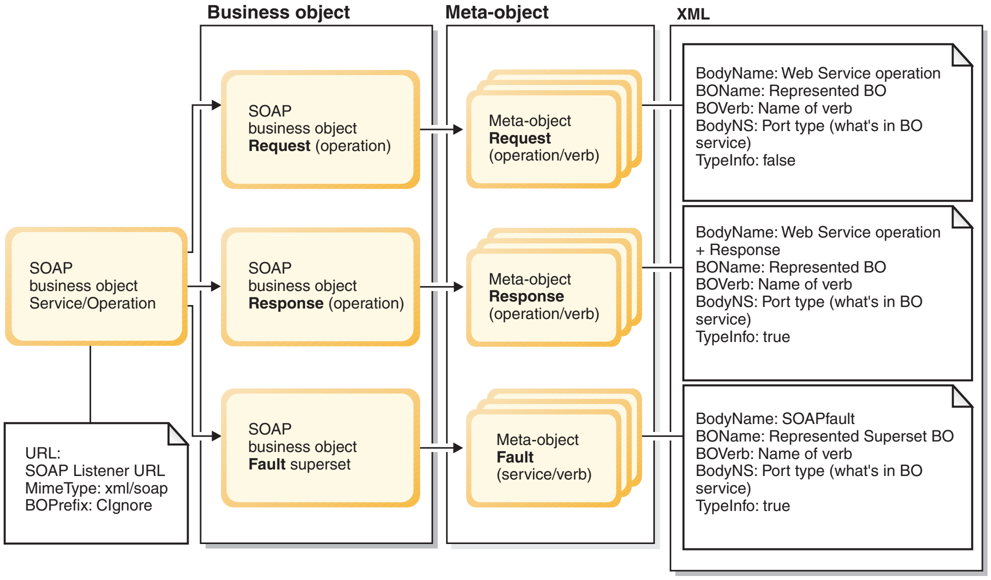

MO_Client_ BOtoSOAP_ ConfigMO
Business object
Meta-object
SOAP business object Request (operation)
SOAP business object Response (operation)
SOAP business object Fault superset
SOAP business object Service/Operation
Meta-object Request (operation/verb)
Meta-object Response (operation/verb)
Meta-object Fault (service/verb)
1
n
MO_Client_ SOAPtoBO_ ConfigMO

Relationship between business objects and meta-objects

Request meta-objects are used for converting business objects to SOAP XML messages. Response and Fault meta-objects are used for converting SOAP XML messages to business objects. The SOAP Request application-specific business object is associated with a meta-object for request messages (business object-to-SOAP XML). SOAP Response and Fault application-specific business objects are associated with meta-objects for reply messages (SOAP XML-to-business object).

SOAP Data relationships

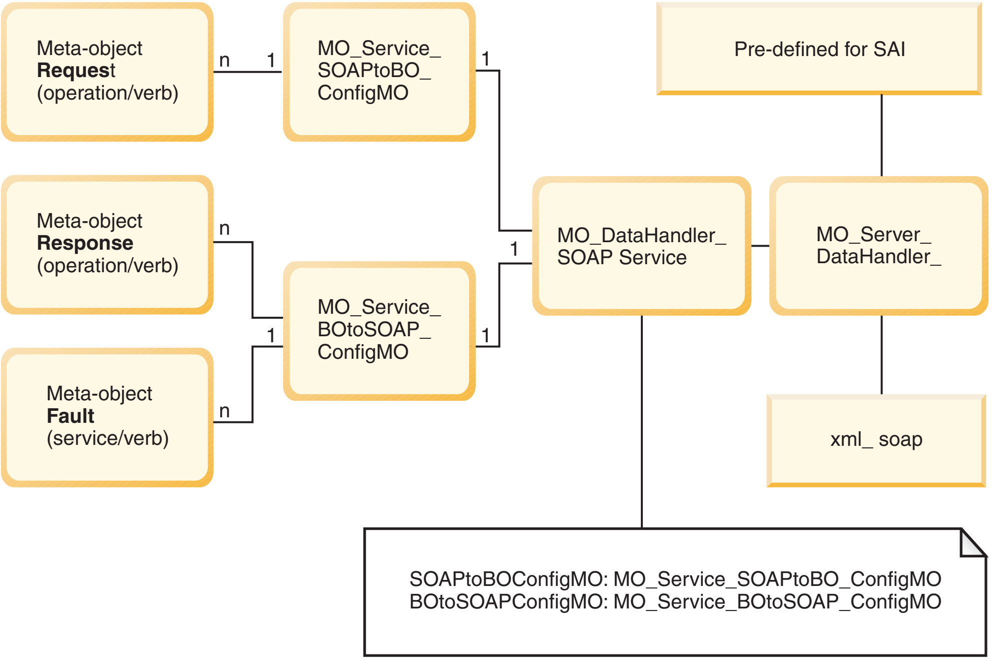
The following illustration shows the data relationships that correspond to the object relationships discussed in the previous section, including the XML that is marshalled and unmarshalled to and from the business object via the SOAP data handler:



Relationship of objects and corresponding XML

SAI meta-object relationships

Meta-objects are used by the Service Access Interface (SAI) data handler to marshal and un-marshal the SOAP XML messages. The Request, Response, and Fault meta-objects need to be created for each SOAP application-specific business object being sent to the SAI. Meta-objects are based on the business object and verb combination. This relationship is shown in the following illustration:



How the SAI data handler marshals and unmarshals SOAP messages

The business objects are the ones associated with the SOAP top-level application-specific business object Request, Response and Fault attributes.

SAI - meta-object-to-business-object relationships

This section describes the object relationships and data relationships in SAI operations.

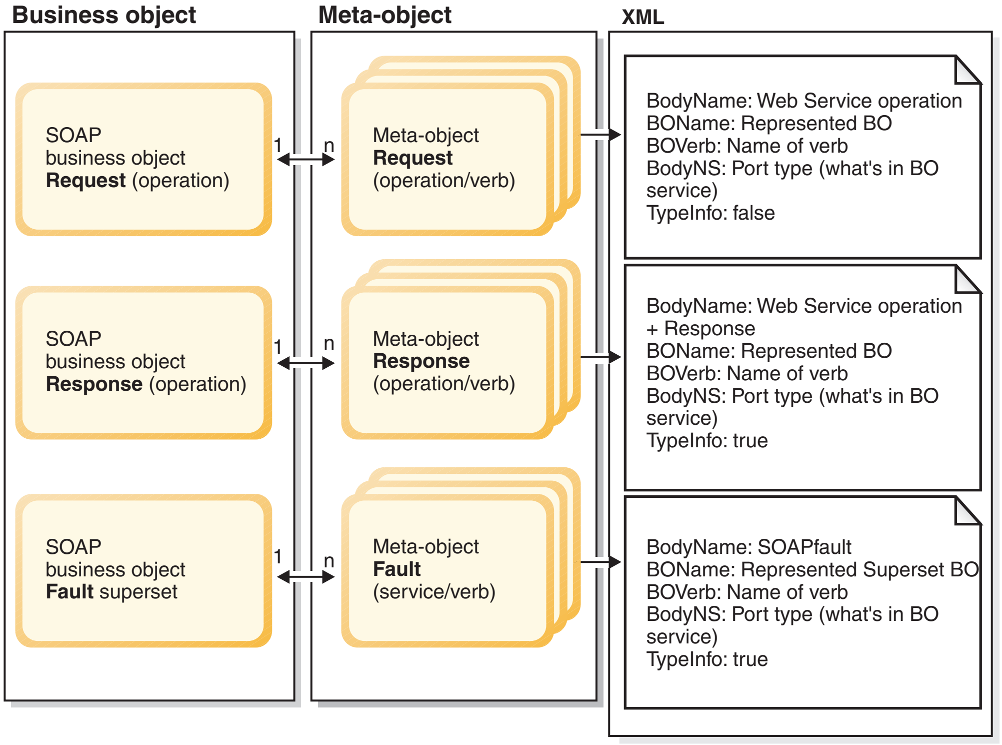
SAI object relationships

The relationship between SOAP application-specific business objects and meta-objects is similar to that of the SOAP connector. Meta-objects are grouped by requests and responses.

In the SAI case, requests go from SAI into CrossWorlds and responses (the replies to the requests) go from CrossWorlds to SAI. Request meta-objects are used for converting SOAP XML messages to business objects. Response and Fault meta-objects are used for converting business objects to SOAP XML messages. The SOAP Request application-specific business object is associated with a meta-object for request messages (SOAP XML-to-business object). SOAP Response and Fault application-specific business objects are associated with meta-objects for reply messages (business object-to-SOAP XML).

SAI data relationships

The following illustration shows the data relationships that correspond to the object relationships discussed in the previous section, including the XML that is marshalled and unmarshalled to and from the business object via the SAI data handler:


Business object
Meta-object
XML
SOAP business object **Request** (operation)
1
n
Meta-object **Request** (operation/verb)
BodyName: Web Service operation
BOName: Represented BO
BOVerb: Name of verb
BodyNS: Port type (what's in BO service)
TypeInfo: false
SOAP business object **Response** (operation)
1
n
Meta-object **Response** (operation/verb)
BodyName: Web Service operation + Response
BOName: Represented BO
BOVerb: Name of verb
BodyNS: Port type (what's in BO service)
TypeInfo: true
SOAP business object **Fault** superset
1
n
Meta-object **Fault** (service/verb)
BodyName: SOAPfault
BOName: Represented Superset BO
BOVerb: Name of verb
BodyNS: Port type (what's in BO service)
TypeInfo: true

Relationship of objects and corresponding XML

Samples

Collaboration templates are provided for inbound and outbound flows to provide a common behavior for SOAP-style interactions. Copy them to the new name and change as required.

Sample scenarios are provided for the following:

- Basic outbound and inbound.
- Basic outbound and inbound with Web Services Gateway Routing capability. This selects a spoke destination when multiple spokes are available for the same Web service.
- Outbound RPC-style call through the SOAP connector.

Naming conventions and values

SOAP business objects are specific to the Web service they represent. You can think of a SOAP business object as an application-specific business object. In addition, the Web-service-specific SOAP application-specific business object requires meta-object definitions (business objects themselves) that are specific to the SOAP application-specific business object.

The type of information required in the meta-object is the name of the Web-service Java class, name space, method to invoke, and business object verb (for example, Create, Retrieve, Update, Delete). Because of these dependencies between a SOAP application-specific business object and meta-object, a naming convention is required. There are two conventions required depending on whether the SOAP request is an RPC style or document style. For CrossWorlds-to-CrossWorlds communication, a document style is used.

Document-style conventions

CrossWorlds to CrossWorlds via Web services uses a SOAP document-style messaging interaction, and Web Services Gateway uses a SOAP RPC-style message interaction. Even with the two message styles, Web Services Gateway can still be used between two CrossWorlds instances, but there are some limitations placed on the data.

Generic business objects are the inputs and outputs of collaborations. Mapping objects convert generic business objects to application- specific business objects. For CrossWorlds to CrossWorlds, a generic business object can be part of the SOAP object. This simplifies the naming convention, which is based on the generic business object.

The Web Services Gateway uses RPC-style message exchange. This means that it cannot return data with multiple parts. Since a generic business object may have multiple attributes, the generic business object must be wrapped with a business object that contains only one attribute—the generic business object.

Refer to "SOAP-object-to-generic-business-object relationship (Document style)" on page 5 for a figure depicting the structure.

Generic business object naming convention

The generic business object name is usually composed of an object type and a name that describes the purpose of the collaboration.

object type*_*descriptive name

For example, the components that make up the Business Connection offering begin with **BCT_** (the object type) followed by a descriptive name.

Recall that in the samples programs, the generic business object is BCT_TestAllTypes.

SOAP application-specific business object naming convention

The general structure of a SOAP application-specific business object is:

Top business object
    Response business object
    Request business object
    Fault business object

Top-level business object

SOAP application-specific business objects need to be defined for the top-level SOAP object and for the Request, Response, and Fault attributes. The business objects are named by adding prefix information to the generic business name, as follows:

object type*_SOAP_*generic business object

For example, if the generic business name is BCT_TestAllTypes (with the object type being BCT_), the SOAP top-level business object is named BCT_SOAP_BCT_TestAllTypes.

Response and Request business objects

The Response and Request business objects are named by adding the same prefix information (as in the top-level business object) to the generic business object plus also adding **Wrapper** as a suffix, as follows:

***object type*_SOAP_*generic business object*_Wrapper**

In the example, the Response and Request business objects are named BCT_SOAP_BCT_TestAllTypes_Wrapper.

Fault business object

The Fault business object is similarly named, except that **Fault** rather than **Wrapper** is used as the suffix:

***object type*_SOAP_*generic business object*_Fault**

In the example, the Fault business object is named BCT_SOAP_BCT_TestAllTypes_Fault.

In addition, the Fault business object requires the following specific attributes:

- faultcode
- faultstring
- faultfactor
- detail - set to the same type as the Request generic business object

AppSpecificInfo attribute

The CrossWorlds WSGenUtility, which is used to create a Web-service Java proxy, requires that the application-specific business objects have an entry in the AppSpecificInfo attribute section giving the business-object name. This build-time requirement is needed only for inbound calls through the SAI. The SOAP connector does not require this information. Most business objects are used for inbound *and* outbound, however, so the information is provided for all business objects. Meta-objects, which are not application-specific business objects, do not require this information.

Review of business-object naming

To review, SOAP business objects follow the convention shown in this table, where ***name*** is the generic business name:

Table 1. SOAP business-object naming convention

SOAP business object	Application-specific business object name/type
Top Level	BCT_SOAP_*name*
Response	BCT_SOAP_*name*_Wrapper
Request	BCT_SOAP_*name*_Wrapper
Fault	BCT_SOAP_*name*_Fault

SOAP header naming convention

SOAP headers require some special information. To add information to a SOAP header, you need an attribute in the SOAP Request application-specific business object that contains a generic business object with a structure mimicking the SOAP header message XML.

The attribute name used is requestHeader, and the generic business object representing the header has been pre-defined to work with the Router and Authentication filters. The requestHeader attribute requires the application-specific information entry of **soap_location=SOAPHeader,** which must go before the type value entry.

For example, using the BCT_TestAllTypes sample, the requestHeader attribute would be:

```
soap_location=SOAPHeader;type=BCT_SOAP_BCT_TestAllTypes_HDR
```

The header element also requires a namespace. The namespace entry being used for Business Connection is:

```
elem_ns=http://www.ibm.com/wbc
```

SOAP meta-object naming convention

SOAP meta-objects need to be defined for the Request, Response and Fault attributes. In addition, they are specific to the verb. There is a naming convention to indicate whether the data flow direction is outbound (business object-to-SOAP message) or inbound (SOAP-to-business object), and to indicate whether the meta-object is associated with the SOAP connector (client) or SAI (service).

Table 2. SOAP meta-object naming convention

meta-object business object	meta-object name
SOAP Application-Specific Top Level	Not required
Request	MO_*ddd*_BOtoSOAP_*sss*_*verb*
Response	MO_*ddd*_SOAPtoBO_*sss*_Response_*verb*
Fault	MO_*ddd*_SOAPtoBO_*sss*_Fault_*verb*

where:

- ***ddd*** indicates whether the name is associated with the SOAP connector or SAI service. SOAP connector uses the value **client** and the SAI service uses the value **service**.
- ***sss*** represents the corresponding SOAP application-specific business object (for example, BCT_SOAP_BCT_TestAllTypes or BCT_SOAP_BCT_TestAllTypes_Fault

Further meta-object naming convention

This section describes the BodyName and BodyNamespace attributes for SOAP meta-objects.

BodyName Attribute

The BodyName is the name of the SOAP Body. In SOAP v2.2, this corresponds to the method name to be invoked in the service.

For Request meta-objects, the BodyName is the name of the generic business object prefixed with **m_.**

m_*generic business object*

For example, if the generic business object name is BCT_TestAllTypes, the resulting BodyName is **m_ BCT_TestAllTypes**.

For Response meta-objects, the BodyName is the name of the Request method with the word **Response** added as a suffix.

***request method name*Response**

For example, if the Request method name is m_BCT_TestAllTypes, the resulting BodyName is **m_ BCT_TestAllTypesResponse**.

BodyNS Attribute

The BodyNS is the namespace of the SOAP Body. This namespace can be anything as long as it is consistently used in the various meta-objects associated with the SOAP application-specific business objects. In SOAP v2.2, the Body Namespace is used by the SOAP Server (servlet) to determine the ID of the service to use for processing a received message.

The Java proxy created by the CrossWorlds WSGenUtility is specific for a generic business object and verb. Therefore, the naming convention is the generic business object plus an underscore (_) plus the verb.

generic business object_verb

For example, if the generic business object is named BCT_TestAllTypes and the verb is Retrieve, the resulting namespace is **BCT_TestAllTypes_Retrieve**.

This works fine for CrossWorlds directly to CrossWorlds. If Web Services Gateway is used, however, a modification to this Namespace is required. When Web Services Gateway is installed, one of the configuration parameters is the Namespace URI for Services, which defaults to **urn:ibmwsgw**.

Web Services Gateway uses a namespace of its URI plus **#** plus the service name. For example, if the generic business object is BCT_TestAllTypes and the verb is Retrieve, the resulting namespace (using the default urn) is **urn:ibmwsgw#BCT_TestAllTypes_Retrieve**.

Map naming convention

Maps are used to convert generic business objects to application-specific business objects and vice-versa. Map names are based on the:

- Source object, followed by the word “to”
- Target object

***SourceBusinessObject*_to_*TargetBusinessObject*.**

For example, if the source is BCT_TestAllTypes and the target is BCT_SOAP_BCT_TestAllTypes, the resulting map name is:

BCT_TestAllTypes_to_BCT_SOAP_ BCT_TestAllTypes.

Several maps are required:

Table 3. Maps required for request/response flows

Flow	**Map required**
Outbound request	Generic business object to SOAP application-specific-business-object top level
Outbound response	SOAP application-specific business object top level to generic business object
Inbound request	SOAP application-specific business object wrapper to generic business object
Inbound response	Generic business object to SOAP application-specific business object fault Polymorphic generic business object to SOAP application- specific business object wrapper

A special map is required for the response application-specific business object coming from the SAI. This map has to produce either a normal SOAP response application-specific business object or a SOAP fault application-specific business object.

This map name consists of:
- The word “Poly”
- The generic business object, followed by the word “to”
- The “normal” response object

Poly_*generic business object*_to_*response object*

For example, if the source is BCT_TestAllTypes and the target can be either BCT_SOAP_ BCT_TestAllTypes_Wrapper *or* BCT_SOAP_ BCT_TestAllTypes_Fault, the resulting map name is:
Poly_ BCT_TestAllTypes _to_ BCT_SOAP_ BCT_TestAllTypes_Wrapper

Collaboration template naming convention

Collaborations are named based on two factors:
- The generic business object on which it operates
- The direction (from a Web-services viewpoint), which can be either:
  - **Outbound** - Calling out to the Web service through the SOAP connector
  - **Inbound** - Calling from a Web service into the SAI

The result is a combination of the generic business object and direction.

For example, if the generic business object is named BCT_TestAllTypes and the direction is outbound through the SOAP connector, the resulting name is:

BCT_TestAllTypesOutbound

Collaboration object naming convention

This name represents an instance of a collaboration template. Instances of collaboration templates are bound to specific physical ports (for example, SOAPConnector, BCTSampleConnector). The name is based on the names of the:

- From port connector, followed by the word “to”
- To port connector
- Collaboration template

The result looks like this:

FromConnector*_to_*ToConnector_CollaborationTemplateName

For example, if the **From** connector is BCTSampleConnector1, the **To** connector is BCTSampleSOAPConnector, and the collaboration template name is BCT_TestAllTypesOutbound, the resulting name is:

BCTSampleConnector1_to_BCTSampleSOAPConnector_BCT_TestAllTypesOutbound

Project naming convention

Project names follow the same convention used for collaboration template names.

RPC-style conventions

RPC style is very similar to document style. The differences are:

- Multiple methods may be encompassed within a single service.
- The generic business objects must match the input parameters for the RPC call.
- There can be multiple input parameters that will require multiple generic business objects.
- The name of the generic business object must be the same as the Java type, including the package name.
- Since the RPC method call return type will probably be different from the input parameter, the SOAP wrapper for the response is different.

Parameter application-specific business object naming convention

Each parameter in an RPC call or return requires an application-specific business object to represent the Java class. For example, com.ibm.bct.ws.samples.rpc.RequestList represents a business object.

Java parameter objects in Web services must follow the Java bean pattern for introspection purposes. The application-specific business objects needs to mimic the Java data types. Attribute names follow the name of the method parameter.

- Attribute names will be the same as the Java attribute name.
- Data types name will be the same as the Java data type. If the Java data type is a class, an application-specific business object needs to be created that matches the class.

RPC SOAP application-specific business object naming convention

The general structure of a SOAP application-specific business object is:

Top BO
    Response BO
    Request BO
    Fault BO

SOAP application-specific business objects need to be defined for the top, request, response and fault attributes.

Table 4. SOAP application-specific business object naming convention

SOAP business object	**Application-specific business object name**
Top level	BCT_SOAP_*www_mmm*
Request	BCT_SOAP_*www_mmm*_Request
Response	BCT_SOAP_*www_mmm*_Response
Fault	BCT_SOAP_*www_mmm*_Fault

where:

- ***www*** is the Web-service name (for example, Stockquote)
- ***mmm*** is the method name (for example, getQuote)

Java parameter objects in Web services are required to follow the Java bean pattern for introspection purposes. SOAP application-specific business object naming conventions follow the bean pattern.

- Attribute names are the same name as the method parameter. The attribute name for the RPC method return has to be **return**.
- Data types name are the same as the Java data type. If the Java data type is a class, an application-specific business object needs to be created that matches the class.

SOAP meta-object naming convention

SOAP meta-objects follow the document-style naming convention. See "SOAP meta-object naming convention" on page 17.

Further meta-object naming convention

This section describes the BodyName and BodyNamespace attributes for SOAP meta-objects.

BodyName Attribute

For Request meta-objects, the *BodyName* is the name of the RPC method call.

RPC method call

For example, if the RPC method name is getQuotes, the resulting name is **getQuotes.**

For Response meta-objects, the *BodyName* is the name of the Request method name with a suffix of **Response**.

Request method name**Response**

For example, if the Request method name is getQuotes, the resulting name is **getQuotesResponse**

BodyNS Attribute

The BodyNS is the Namespace that will be used for the namespace of the SOAP Body. This namespace can be anything as long as it is consistently used in the various meta-objects associated with the SOAP application-specific business objects.

In SOAP v2.2, the Body Namespace is used by the SOAP Server (servlet) to determine which service to call for a received message. This value can be obtained from the id attribute of the service element, which can be found in the Web-service deployment descriptor (dds.xml). For example, ***http://tempuri.org/com.ibm.bct.ws.samples.rpc.Stockquote***

Map naming convention

Maps follow the document-style naming convention. See "Map naming convention" on page 18.

Note: Maps cannot be created using business objects with periods (.) in the business-object name. Any mapping against the parameter application-specific business objects will need to be done with custom maps.

Collaboration template naming convention

Collaboration templates follow the document-style naming convention. See "Collaboration template naming convention" on page 19.

Collaboration object naming convention

Collaboration objects follow the document-style naming convention. See "Collaboration object naming convention" on page 20.

Project naming convention

Project names follow the same convention used for collaboration template names.

Creating CrossWorlds-to-CrossWorlds flow

For CrossWorlds to CrossWorlds using a SOAP service, the following steps are required.

1. Define the generic business objects on the source and destination side.
2. Define the SOAP application-specific business objects for the SOAP connector and SAI.
3. Define the meta-objects for the SOAP application-specific business objects.
4. Define the mappings to convert from generic business objects to SOAP application-specific business objects and from SOAP application-specific business objects to generic business objects.
5. Define the collaboration templates.
6. Define the collaboration objects.
7. Run the WSGenUtility to create the Java proxy class for invoking the SAI.
8. Create the SOAP service.
9. Deploy the CrossWorlds artifacts and configure:
   a. The URL for the SOAP message to be sent to
   b. The supported business objects by the SOAP connector and other connectors involved
   c. The meta-objects used by the SOAP connector and SAI so that they have references to your specific meta-objects
10. Deploy the SOAP service and configure the Java proxy class by setting:
    a. log4j for tracing
    b. Configuration information for the Java proxy to use
11. If Web Services Gateway is being used, deploy the WSDL generated by the WSGenUtility into Web Services Gateway.

If you are using the suggested naming conventions as described in “Naming conventions and values” on page 13, the WSGenUtility along with the predefined templates can very easily be used for creating the initial meta-objects, SOAP application-specific business objects, and custom maps. The predefined templates are based on a solution pattern developed for the samples. Refer to section “Solution pattern” for the solution pattern. If different naming conventions and business objects are being used, you can create your own templates.

Solution pattern

The solution pattern is based on sending a generic business object from one CrossWorlds instance via the SOAP connector to another CrossWorlds instance that has a collaboration exposed as a SOAP service using the SAI. The same generic business object is used in both collaborations for the request and the response. This means that the generic business object can be encapsulated with a SOAP application-specific business object for the request and response.

Outbound SOAP connector flow

For the outbound flow through the SOAP connector, mappings are required for mapping the generic business object to the SOAP top-level application-specific business object and for mapping the response SOAP top-level application-specific business object to the generic business object.

For mapping to the SOAP top-level application-specific business object, two custom maps are used to:

- Dynamically obtain the URL. A samples utility is provided for obtaining the URL from a properties file based on the generic business object name and verb.
- Create the SOAP Request application-specific business object that populates the child attribute with the generic business object and creates the SOAP header objects.
  SOAP header objects are used by the Web Services Gateway Routing and Authentication filters.

For mapping the SOAP top-level application-specific business object reply to the generic business object, one custom map is used. This custom map checks the Fault attribute of the SOAP top-level application-specific business object and, if it is populated, throws an exception. Otherwise, the SOAP top-level application-specific business object Response object will have its child attribute object copied into the generic business object.

Inbound SAI flow

For the inbound flow through the SAI, one map is used to convert the SOAP Request application-specific business object to the generic business object. Two maps are used to convert the generic business object to the SOAP Response application-specific business object.

For mapping from the SOAP Request application-specific business object, the map uses custom code to copy the SOAP Request application-specific business object child attribute object to the generic business object.

Note: the generic business object from the outbound collaboration is expected to be the same as the generic business object for the inbound collaboration.

For mapping the generic business object to the SOAP Response application-specific business object, two maps are used:

- A polymorphic map
- A generic-business-object-to-SOAP-Fault-application-specific-business-object map

Custom code in the polymorphic map checks the error attribute of the generic business object. If the attribute value is null or BCTNOFAULT, the SOAP Response application-specific business object is created with the generic business object set as the child attribute; otherwise, an error is assumed and the generic-business-object-to-SOAP-Fault-application-specific-business-object map is called. Custom map code in the generic-business-object-to-SOAP-Fault-application-specific-business-object will create the SOAP Fault application-specific business object, setting the generic business object into the SOAP Fault application-specific business object detail attribute.

Using CWGenUtility for CrossWorlds-to-CrossWorlds

This section describes the pre-defined templates provided to use with the CWGenUtility. The templates follow the solution pattern described in the previous section. A more detailed description of the CWGenUtility is provided in "Using the CWGenUtility" on page 37. Note that the CWGenUtility can be used only on a Windows platform.

Predefined names, director, and template files are provided for creating meta-objects, SOAP application-specific business objects, and custom maps that follow the naming convention and solution pattern. In addition, a text file is created with all of the generated names that can be used for inputs when you configure or run the WSGenUtility.

The generated business objects can be imported into CrossWorlds. The code for the maps can be pasted into the custom-map input screens.

The following table describes the input files used:

Table 5. Input file description

File name	Description
bctwsnames.txt	Variables defined for use in the director file and templates
RequestResponseWithHeader.xml	Director file for directing the operations performed
*.tpt	Template files

The output files are in the directory: Generated\gbo

The file names are based on the generic business object name and verb. The following table describes the output files:

Table 6. Output file description

File name	Description
gbo.verb.bo.in	CrossWorlds import file containing all the generated business objects
gbo.verb.names.txt	Text file containing the generated names
cw_wsgenutility_inputfile.txt	Contains the inputs for the WSGenUtility
map_outbound_gbo_to_soap_top_url _request.txt	Custom map used on the SOAP connector side for obtaining the URL for the outbound SOAP top-level application-specific business object
map_outbound_gbo_to_soap_top_request _request.txt	Custom map used on the SOAP connector side for creating the outbound SOAP request application-specific business object
map_outbound_soap_top_to_gbo _response.txt	Custom map used on the SOAP connector side for converting the returned SOAP Response application-specific business object to the generic business object
map_inbound_soap_wrapper_to_gbo _request.txt	Custom map used on the inbound SAI side for converting the SOAP Request application-specific business object to the generic business object

Table 6. Output file description (continued)

File name	Description
map_inbound_sub_gbo_to_soap_fault_response.txt	Custom map used on the inbound SAI side for converting an error in the Response generic business object into a SOAP Fault application-specific business object
map_inbound_poly_gbo_to_soap_wrapper_response.txt	Custom map used on the inbound SAI side for converting the Response generic business object into the SOAP Response application-specific business object. If there is an error, the generic business object to SOAP Fault map will be called.

To invoke CWGenUtility, enter (as one string):

```
java com.ibm.bct.ws.cw.BCTWSDirector -director ./RequestResponseWithHeader.xml
-names ./bctwsnames.txt BASE_DIR_NAME=./REPOS_VERSION_FILE=c:\crossworlds\
repository\reposversion.txt gbo=gbo_name verb=verb_name
cwcollabtemplate=collab_template_name CWICS=cw_instance_namewhere:
```

- ***gbo_name*** is the name of the generic business object (for example, BCT_TestAllTypes)
- ***verb_name*** is the name of the verb (for example, Retrieve)
- ***collab_template_name*** is the base collaboration template name (for example, BCT_TestAllTypes)
- ***cw_instance_name*** is the CrossWorlds ICS name

Creating the SOAP service

The samples provide a starter SOAP service that you can use to create your own Web service from the class created by WSGenUtility. Once the basic Web service is created, you can easily add in other generated classes. The basic steps are:

1. Change the name of the provided sample Web service to your Web service name. Use the WebSphere Application Assembly tool to change the name.
2. Import your renamed Web service EAR file into WebSphere Application Developer so that your classes can be added.
3. Add in your classes.
4. Update the configuration information.
5. Export and deploy the ear file.

After you complete this procedure, only steps 3 through 5 need to be done for any additional classes.

In this example, the name of your SOAP service is *Your_Name* and the sample *BCT_TestAllTypes* will be used to embed into the service.

Step 1: Change the name of the EAR and WAR files

In order to import the BCT_WS_WebService_Template_EAR file into WebSphere Application Developer, you must change the names of the EAR and WAR files. Use the WebSphere Application Assembly Tool as follows:

1. Open the WebSphere Application Assembly Tool and select the **BCT_WS_WebService_Template_EAR**.
2. In the **General** tab, change the **Display** name to *Your_Name*_EAR.

3. Select the **BCT_WS_WebService_Template_EAR>Web Modules>BCT_WS_WebService_Template**
4. In the General tab
   a. Change the File name, Context root, and Display name to your new name (***Your_Name*_Web**). **Note**: *Context root* is part of the SOAP Service URL that will be used. See the section "Context root-to-URL relationship" on page 30 for a fuller explanation.
   b. Click **Apply**.
5. Save the new EAR file (using **File>Save as**) to your new name (***Your_Name*_EAR.ear**).

Step 2: Import the file into WebSphere Application Developer

1. Import the new EAR file into WebSphere Application Developer
2. After you import the file, there will be compilation errors. To resolve these errors, add the soap.jar file to the Web module (***Your_New*_Web**) *Java build path*. You can use the predefined WebSphere Application Developer variable SOAPJAR, or you can obtain the file from the directory: WebSphere\AppServer\lib

Step 3: Add the WSGenUtility generated classes

1. In the Navigator view, add the WSGenUtility generated Java classes.
   a. Select the ***Your_Name*_Web > *source*.**
   b. Click **File>Import>File System**.
   c. Click the **Next** button.
   d. Navigate to the Java source file and select the Java file (not the compiled class file). Make sure that **Create complete folder structure** is **not** checked.
   e. Click **Finish**.
2. Remove the TestDummyBean.
   a. Right-click on the **TestDummyBean.java** file in the *source* folder and delete it. This should also cause the TestDummyBean.class file in the classes directory to be removed.
   b. Right-click on the ***Your_Name*_Web_classes.jar** in the lib folder and delete it. This step removes another copy of the TestDummyBean.class.

Step 4: Update configuration information

1. Update the Deployment Descriptor. From a received SOAP message, the SOAP server matches the body namespace and method to the deployment descriptor to determine which Java class to invoke.
   a. Double-click the **dds.xml** file to open the file in the editor.
   b. Copy and paste the example service element lines to between the root element; then edit the indicated values. The values can be obtained either from the wsdl or the *xxx*_Readme.txt files generated by the WSGenUtility. See the section "WSDL-to-deployment descriptor relationship" on page 30 for details.
2. Update the Servlet initialization parameters. This will contain the configuration file name and location required by the WSGenUtility generated Java proxy.
   a. Double-click the **web.xml** file to open the file in the editor.
   b. Click the **servlets** tab.
   c. Click **messagerouter** to highlight it, and then click **Intialization**.

   d. Add the init parameter from the WSGenUtility generated *xxx*_Readme.txt file. Because no directory is specified, the Java proxy will look in the WebSphere Application Server instance working directory.
   e. Click **OK**.
   f. Save the web.xml file.
3. Update the log4j.properties file, which is used for configuring the WSGenUtility-generated Java proxy trace settings.
   a. In the *source* folder, double-click **log4j.properties** to open the file in an editor. **Note**: Be sure to use the copy in the *source* folder and not the *classes* folder.
   b. Copy and paste the two lines containing the *servicename* and replace the *servicename* with the category value in the WSGenUtility-generated *xxx*_Readme.txt file.

Step 5: Export and deploy the Ear file

1. Right-click the Ear file (***Your_Name*_EAR**) and select **Export EAR File**.
2. Enter the EAR file name (***your_name*.ear**), and click **Finish**.
   Deployment is similar to the description for the samples.

Context root-to-URL relationship

The context root helps identify the SOAP service location and is part of the URL location in the WSDL. In the following example, the WSDL has a soap:address entry of:

```
<service name="BCT_TestAllTypes_Retrieve">
   <port name="BCT_TestAllTypes_RetrievePort" binding=
   "tns:BCT_TestAllTypes_RetrieveBinding">
  <soap:address location=
"http://localhost/bctwssamplesweb/servlet/messagerouter"/>
   </port>
</service>
```

- The URL is: http://*localhost*/bctwssamplesweb/servlet/messagerouter
- The context root is: bctwssamplesweb

WSDL-to-deployment descriptor relationship

This section shows the relationship among the SOAP service Deployment Descriptor (dds.xml), the WSDL, and the information in the WSGenUtility-generated xxx_Readme.txt file. The contents of the files are shown first.

Deployment descriptor fragment

```
<isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment"
                id="urn:ibmwsgw#BCT_TestAllTypes_Retrieve"
                type="message"
                checkMustUnderstands="false">
   <isd:provider type="java" scope="Request" methods="m_BCT_TestAllTypes">
        <isd:java class="BCT_TestAllTypes_Retrieve" static="false"/>
   </isd:provider>
</isd:service>
```

WSDL fragment

```
<wsdl:binding name="BCT_TestAllTypes_RetrieveBinding"
  type="tns:BCT_TestAllTypes_RetrievePortType">
  <soap:binding style="rpc" transport=
  "http://schemas.xmlsoap.org/soap/http"/>
```

```
<wsdl:operation name="m_BCT_TestAllTypes">
 <soap:operation soapAction=""/>
 <wsdl:input>
  <soap:body use="encoded"
      namespace="urn:ibmwsgw#BCT
      _TestAllTypes_Retrieve"
     encodingStyle="http:
  //schemas.xmlsoap.org/soap
  /encoding/">
  </soap:body>
</wsdl:input>
```

xxx_Readme.txt fragment

```
SERVICE ID                        ClassName             Methods
============================================================================
urn:ibmwsgw#BCT_TestAllTypes_Retrieve BCT_TestAllTypes_Retrieve m_BCT_TestAllTypes
```

Relationship

The following table shows where each of the values in the first column is found in the three files.

Table 7. How values appear in dds.xml, WSDL, and Readme.txt files

Value	Deployment Descriptor	WSDL	Readme.txt
urn:ibmwsgw#BCT_TestAllTypes_Retrieve	id	soap:body namespace	SERVICE ID
m_BCT_TestAlltypes	methods	wsdl:operation name	Methods
BCT_TestAllTypes_Retrieve	isd:java class		ClassName

Enabling an RPC Web service

This section describes what needs to be done to enable an RPC service to work with the CrossWorlds SOAP connector, focusing on using WebSphere Application Developer.

There are limitations to the SOAP connector's ability to create SOAP XML messages. Some limitations are internal to the SOAP connector design and others are limited by CrossWorlds business objects. This means that not any RPC Web service can be invoked from CrossWorlds, but only those that fit within CrossWorlds's capabilities.

Limitations to RPC calls are:

- Only one name space is supported for SOAP Body Namespace and the data types. This means that the namespace for data types has to be the same as the Body namespace.
- Only data types that are supported in a generic business object can be used. This means no:
  - Arrays of primitives, but only arrays of complex types
  - Vectors, hash tables, and so on

Creating an RPC Web service

To create the RPC service:

1. Create the Web service as normal within WebSphere Application Developer. The WSDL files are not used and can be ignored.
2. Modify the Web-service Deployment Descriptor file (dds.xml). See "Modifying RPC Web service DDS file"for details.
3. Export the EAR file and deploy the Web service.
4. Create the CrossWorlds parameter application-specific business objects, SOAP application-specific business objects, and meta-objects. See "RPC-style conventions" on page 21 for details.
   Business-object names cannot be created with periods by the CrossWorlds Business Object Designer. However, a business object that contains names with periods can be imported. See "Using the CWGenUtility" on page 37 for help in creating these business objects.
5. Create any required maps, collaborations, and so on.
   Note: Maps cannot be made using business objects with periods (.) in the business-object name. Any mapping against the parameter application-specific business objects will need to be done with custom maps.

Modifying RPC Web service DDS file

WebSphere Application Developer creates a Service deployment descriptor (dds.xml) file with multiple namespaces. This violates the single namespace requirement and so needs to be modified. There will be a namespace for the identifier attribute (id) and namespaces for each mapping. Mappings are provided for each Java type to tell the SOAP Server how to serialize the datatype.

This example shows an extract from WebSphere Application Developer dds.xml file, with two namespaces highlighted:

```
<root>
<isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment"
 id="http://tempuri.org/com.ibm.bct.ws.samples.rpc.Stockquote"
 checkMustUnderstands="false">
  <isd:provider type="java" scope="Application"
methods="getCompanyQuotes getQuote getQuotes getCompanyQuote">
<isd:java class="com.ibm.bct.ws.samples.rpc.Stockquote" static="false"/>
  </isd:provider>
  <isd:mappings>
    <isd:map encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:x="http://www.stockquote.com/schemas/StockquoteRemoteInterface"
qname="x:com.ibm.bct.ws.samples.rpc.ResponseList"
javaType="com.ibm.bct.ws.samples.rpc.ResponseList"
xml2JavaClassName="org.apache.soap.encoding.soapenc.BeanSerializer"
java2XMLClassName="org.apache.soap.encoding.soapenc.BeanSerializer"/>
    <isd:map encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
```

For each namespace (xmlns:x), change the value to be the same as the id. The following is a sample extract to the dds.xml file with modifications highlighted.

```
<root>
<isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment"
 id="http://tempuri.org/com.ibm.bct.ws.samples.rpc.Stockquote"
checkMustUnderstands="false">
  <isd:provider type="java" scope="Application"
       methods="getCompanyQuotes getQuote getQuotes getCompanyQuote">
       <isd:java class="com.ibm.bct.ws.samples.rpc.Stockquote" static="false"/>
  </isd:provider>
  <isd:mappings>
  <!-- Changed the namespace xmlns:x="http://www.stockquote.com/schemas
       /StockquoteRemoteInterface"-->
  <!-- to be the same as the above "id" value in the following 4 lines -->
  <isd:map encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                xmlns:x="http://tempuri.org/com.ibm.bct.ws.samples.rpc.Stockquote"
                qname="x:com.ibm.bct.ws.samples.rpc.ResponseList"
                javaType="com.ibm.bct.ws.samples.rpc.ResponseList"
  xml2JavaClassName="org.apache.soap.encoding.soapenc.BeanSerializer"
   java2XMLClassName="org.apache.soap.encoding.soapenc.BeanSerializer"/>
```

Product data mappings

This section contains mappings between the different products and how the different data pieces relate to each other.

Table 8. Product/data mappings

Data	**CrossWorlds**	**CrossWorlds Web service**	**WSGenUtility**	**Web Services Gateway**	**WSDL**	**SOAP XML**
Service name			WSGU1	GW1	WSDL1	
Method name	CW2	CWWS2	WSGU2		WSDL2	S2
Target Name space	CW3	CWWS3	WSGU3	GW3	WSDL3	S3
Service URL	CW4	CWWS4	WSGU4	GW4	WSDL4	
Java Class name		CWWS5	WSGU5			

Service Name

- WSGU1 - The GUI Service Namespace entry; puts the name into WSDL1
- WSDL1 - The *service* element *name* attribute value
- GW1 - Use in the Web Services Gateway Deployment *GateWay Service Name* field

Method Name

- CW2 - Used in the meta-object *BodyName* attribute.
- CWWS2 - Used in the DDS *provider* element *methods* attribute value.
- WSGU2 - Obtains this value from CW2.
- WSDL2 - Used in the *operation* element *name* attribute value. WSGU2 also uses this value as the root for the *message* element *name* attribute value.
- S2 - The name of the first element under the SOAP Body element.

Target Namespace

- CW3 - Used in the meta-object *BodyNS* attribute.
- CWWS3 - Used in the DDS *service* element *id* attribute value. This is used by the SOAP v2.2 Server to determine what service to call.
- WSGU3 - The GUI Target Namespace entry. Puts the name into WSDL3.
- WSDL3 - The *definitions* element *targetNamespace* attribute value.
- GW3 - If using Web Services Gateway, this value needs to be the same as what Web Services Gateway creates, which is the GW1 value prefixed with the Gateway Namespace URI plus #
- S3 - The SOAP Body Namespace value for S2.

Service URL (service end point)

- CW4 - Used in the SOAP top-level application-specific business object *URL* attribute value. SOAP connector uses this value for the HTTP address when sending the message.
- CWWS4 - This Web service is deployed in a SOAP Server that this URL is the address of.
- WSGU4 - The GUI Binding URL entry. Puts the value into WSDL4.
- GW4 - Obtains this value from the WSDL. Web Services Gateway provides a channel URL to use.
- WSDL4 - The *address* element *location* attribute value. This element is under the port *element* that is under the *service* element.

Java Class Name

- WSGU5 - The GUI Proxy Class Name entry. This is the Java proxy class name for the Java class that the WSGenUtility creates for deploying as a Web service.
- CWWS5 - The *java* element *class* attribute value.

Using the CWGenUtility

Use the CWGenUtility for creating:

- Generic business objects
- SOAP application-specific business objects
- Meta-objects
- Custom maps
- Names

This utility provides a standard, repeatable way to create these artifacts (on a Windows system) so that they can be imported into CrossWorlds.

Running the utility

To run the utility, you enter the following:

```
java com.ibm.bct.ws.cw.BCTWSDirector <-director value> [-names value] [name=value]*
```

where:

-director ***value*** is the name of the XML file containing the flow sequence

-names ***value*** is the name of the optional file containing the names file

name=value, which can occur multiple times, replaces or adds to the entries in the Names file

The tool works using three types of input files:

- Names file
  This file, which is in the format of a Java properties file, contains substitutable parameters for use in the Director file and Template files. The Names file can contain substitutable entries.
- Director file
  This file, which is in XML format, controls the sequence of steps that will occur. The Director file can contain substitutable entries.
- Template files
  Template files, which are text files, are used for generating the output files. Template files can contain substitutable entries.

The process is as follows:

1. CWGenUtility reads the Names file (if one is specified) and stores the entries in memory.
   - Substitution occurs from the top down.
   - Any command-line-parameter substitutions override the Names-file entries.
2. The utility then reads the Director file and processes a task at a time, working from the top down.
   Any substitutions are done before executing the task. There can be **any number** of tasks. The two basic tasks are:
   - File operations for opening a file (new or append), closing a file, and creating a new directory

- Template processing, in which the designated template file is read, one line at a time, and in which substitutions are performed on each line before the line is output to the referenced file

Names file

The Names file contains any substitutable values. This file is not required.

The format of the file is a series of *name=value* pairs. Note also that:

- A # sign in the first column is treated as a comment and the line is not processed.
- Values to be substituted can be any name, as long as they start and end with a % sign.
- Substitutable values are processed from the top down.

An example of a Names file and an explanation of the processing steps in the file follow:

```
# This is a comment
GBO=BCT_TestAllTypes
Verb=Retrieve
SOAP_Request_ASBO=BCT_SOAP_%GBO%_Request
MO_Client_Out=MO_Client_BOtoSOAP_%SOAP_Request_ASBO%
```

The processing is as follows:

1. The first line is a comment and therefore ignored.
2. In the second line, the value **BCT_TestAllTypes** is assigned to GBO.
3. In the third line, the value **Retrieve** is assigned to Verb.
4. In the fourth line, the value BCT_SOAP_TestAllTypes_Request is assigned to SOAP_Request_ASBO. The **%GBO%** from within the name is replaced by the value that was assigned in the second line (**BCT_TestAllTypes**).
5. In the last line, the value MO_Client_BOtoSOAP_BCT_SOAP_BCT_TestAllTypes_Request is assigned to MO_Client_Out. The **%SOAP_Request_ASBO%** from within the name is replaced by the value that was assigned in the fourth line.

Note: Any command line *name=value* arguments override equivalent values in the Names file.

Director file

The Director file is used for controlling the tasks to perform. Each task is read in and the task performed. Any substitutions are made before the task is called. The order of operation is from the top down.

The structure is:

```
<bctwsrootcw>
     <task name=task_name>
 <param name="param_name1">value for param</param>
 <param name="param_name2">value for param</param>
 <param name="param_name3">value for param</param>
    </task>

    <task name=task_name>
```

```
  <param name="param_name1">value for param</param>
    <param name="param_name2">value for param</param>
      </task>
</bctwsrootcw>
```

Allowed tasks name and their purpose are described in the following table.

Table 9. Task descriptions

Task Name	Purpose
fileoutput	Used for opening files, closing files, and creating directories
template	Used for reading in a template file and writing out the results to an output file opened using the fileoutput task

Each task has one or more *param* sub-elements that have a *name* attribute value and text providing the data associated with that *param*. The number of param entries is specific to the task. The *value for param* can be any valid character string for that parameter's purpose. Refer to the subsequent sections for more information on each task and its associated *param* elements.

Fileoutput task

The **fileoutput** task is used for opening files, closing files, and creating directories. The following table contains a description of the possible entries.

Table 10. Fileoutput parameters

Parameter name	Required	Values	Purpose
reference	Required	Any string	Used in subsequent tasks for writing, closing, and so on
action	Required	new, append, close, mkdir	new - open a new file for writing; will overwrite an existing file append - open a file for writing; will append if the file already exists; otherwise, a new file is created close - close an open file mkdir - create a directory
filename	Used with these action parameters: New - Optional Append - Optional Close - Not used mkdir - Required	Any valid file name or directory name	new - File name to open for writing; if not provided, defaults to console append - File name to open for writing; if not provided, defaults to console mkdir - name of directory to create. Creates subdirectories that do not exist.

Creating a directory

Note: %BCTWSOUTPUTDIRECTORY% is set to *../dir1/dir/*

```
<task name="fileoutput">
       <param name="filename">%BCTWSOUTPUTDIRECTORY%</param>
       <param name="reference">file1</param>
       <param name="action">mkdir</param>
</task>
```

Opening a file for output, replacing any existing files

Note: %BCTWSOUTPUTDIRECTORY% is set to *../dir/dir2/*

```
<task name="fileoutput">
 <param name="filename">%BCTWSOUTPUTDIRECTORY%myBOs.in</param>
<param name="reference>file1</param>
<param name="action">new</param>
</task>
```

In this example, a file name of myBOs.in is specified, along with the directory.

Closing a file

```
<task name="fileoutput">
 <param name="reference>file1</param>
 <param name="action">close</param>
</task>
```

In this example, no file name is entered because the close action works only on the current file associated with the reference value.

Template task

The **template** task is used for the actual creation of your files. The input template file is read in a line at a time, any substitutions are made, and the output is written to the file reference. A # character in the first column designates a comment and is ignored.

The following table contains a description of the possible entries.

Table 11. Template task parameters

Parameter name	**Required**	**Values**	**Purpose**
templatefilename	Required	Any string	The name of the template file including the directory (if not the current directory)
fileoutputreference	Required	Any string	The fileoutput task reference parameter value used when opening a file

Note: In this example, the template file being used does not reside in the current directory, so a reference is used that could have been specified in the names file.

```
<task name="template">
 <param name="templatefilename">%TEMPLATE_LOCATION_1%bct_soap_wrapper.asbo.
 tpt</param>
 <param name="fileoutputreference">file1</param>
</task>
```

Notices

IBM may not offer the products, services, or features discussed in this document in all countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION ″AS IS″ WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

WebSphere Business Connection Lab Director
IBM RTP Laboratory
3039 Cornwallis Road
P.O. BOX 12195

Raleigh, NC 27709-2195
U.S.A

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples may include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

All statements regarding IBM’s future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM’s application programming interfaces.

Programming interface information

Programming interface information is intended to help you create application software using this program.

General-use programming interfaces allow you to write application software that obtain the services of this program’s tools.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

Warning: Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

Trademarks and service marks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM
alphaWorks
AIX
CrossWorlds
DB2
DB2 OLAP Server
DB2 Universal Database
DeveloperWorks
MQSeries
SecureWay
WebSphere

Lotus is a trademark of International Business Machines Corporation and Lotus Development Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

Other company, product or service names may be trademarks or service marks of others.