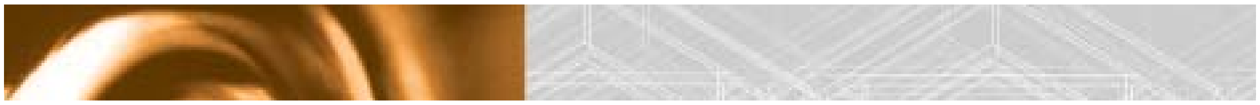
hp service composer



user's guide

Technology Preview-
Version 0.2, January 2002

invent

Contents

About this Guide

hp Service Composer Preview Edition

Thank you for downloading this preview edition of the new **hp Service Composer (HPSC)**!

hp Service Composer is part of a new suite of tools from Hewlett-Packard that will assist the developers in the daunting task of learning and applying Web Services.

Our continually growing tool suite consists of the following integrated tools:

- **hp Registry Composer**, which assists developers and system administrators in viewing, updating, and maintaining UDDI registries.
- **hp Service Composer**, which facilitates the creation and maintenance of new and existing Web Services.
- **hp RadPak** *(future product)*, which facilitates the packaging and deployment of Web Services to the hp Web Service's runtime.

Audience

This edition of the hp Service Composer is being made available in a preview release form to give the developer community an early access to this exciting new technology, and to solicit feedback and suggestions from a broad spectrum of Web Service developers[1]. Since this edition of hp Service Composer is an early release, it naturally suffers from certain deficiencies and limitations. We encourage you to read the *Readme* file that accompanies this release for the latest list of known issues and bugs.

Key Features

The hp Service Composer's key features supported by this preview release are the abilities to create, view, or edit the following:

1 Web Services is a new and evolving technology. Hewlett-Packard greatly values the feedback and suggestions of its developer community. Please refer to the Readme file accompanying this release for feedback channels for this product.

XML Schema Descriptions of Documents

The hp Service Composer has the ability to create, view, and edit XML schema descriptions of documents to be exchanged within a Web Service. The Schema Editor offers integrated graphical and hierarchical editors that allow you, the developer, to create document descriptions without requiring in-depth XSD knowledge or direct manipulation of XSD syntax. HPSC can import existing XSD[2] definitions and perform validation to simplify error identification and correction. Additionally, to maximize flexibility and utility, HPSC will also integrate with your favorite XML IDE if desired.

External Web Service Definitions

The hp Service Composer has the ability to create, view, and edit external Web Service definitions without requiring an in-depth WSDL knowledge or direct manipulation of the WSDL syntax.

The hp Service Composer provides complete support[3] for WSDL constructs and semantics, but allows the developer to manipulate these constructs using simple hierarchical displays, property editors, and wizards. Additionally, HPSC can be integrated with the hp Registry Composer to view and manipulate WSDL files stored as UDDI tModels. Imported WSDLs are validated so that syntactic and semantic errors can be quickly identified and corrected.

New Web Services

The hp Service Composer has the ability to create new Web Services from existing Java class methods and/or EJBs. By interacting with HPSC through a series of wizard pages, the developer can create “bottom-up” Web Service implementations, exposing some or all of a Java class or EJB’s public methods through the Web Service interfaces.

Client Proxies

The hp Service Composer can create client proxies that can be used to implement applications that access Web Services that conform to the external interface being viewed or edited by HPSC. Such client applications can be used either to test a Web Service being developed, or to utilize the functionality of a Web Service as part of a larger application package.

Skeleton Implementations of Web Services

The hp Service Composer can create skeleton implementations of Web Services that can be deployed to a Web Service’s runtime such as that provided by hp’s Web Service platform.

2 HPSC currently only supports XSD files that conform to the 2001 XML Schema specification.

3 Subject to certain limitations of this technology preview release, of course. Support is provided for most features described in the current W3C note for WSDL service descriptions bound to HTTP and SMTP transports.

hp Web Service Platform Deployment Descriptors

The hp Service Composer can generate hp Web Service platform deployment descriptors for a Web Service. These deployment descriptors, combined with generated WSDL and skeleton files, can then be packaged and deployed using a tool such as **RakPak** to create a fully functional Web Service implementation. The hp Registry Composer can then be used to register the newly created and deployed service with a developer, enterprise, or global UDDI registry.

Organization

This manual contains the following chapters:

- "User Interface", which describes the HPSC user interface in some detail. The user is referred to the trail maps that accompany this release for tutorial-level instruction on the use of the hp Service Composer tool.
- "Concepts and Terminology", which gives a brief description of the key terms and concepts integral to understanding Web Services and the hp Service Composer product. This chapter is provided mostly for the users who are new to the Web Services.

Documentation Conventions

The following conventions are used in this guide:

Table 1 Documentation Conventions

Convention	Description
Bold	Used to identify menu selections, toolbar selections, and section references.
Italic	In paragraph text, italic identifies the titles of documents that are being referenced. When used in conjunction with the Code text described below, italics identify a variable that should be replaced by the user with an actual value.
Bold Italic	Used to identify a term that is defined in the Glossary. Readers should reference the Glossary when they are unsure of the complete meaning of a term.
`Code`	Text that represents programming code.
CTRL+*X*	A combination of keystrokes that is used to complete a function. For example, CTRL+C indicates that the CTRL key should be held down while the C key is pressed.
Function | Function	A path to a function or dialog box within an interface. For example, "Select **File** | **Open**" indicates that you should select the **Open** function from the **File** menu.

Convention	Description
( ) and |	Parentheses enclose optional items in command syntax. The vertical bar separates syntax items in a list of choices. For example, any of the following four items can be entered in this syntax: `persistPolicy(Never|OnTimer|OnUpdate|NoMoreOftenThan)`
Note and **Caution**	A **Note** highlights important supplemental information. A **Caution** highlights procedures or information that is necessary to avoid damage to equipment, damage to software, loss of data, or invalid test results.

User Interface

The hp Service Composer tool follows a “toolbox” approach to user-interface design, allowing the developer to select and operate on any Web Service element in any order desired. The Web Service elements are items such as Types, Services, and Operations, which are contained within the current Web Service project.

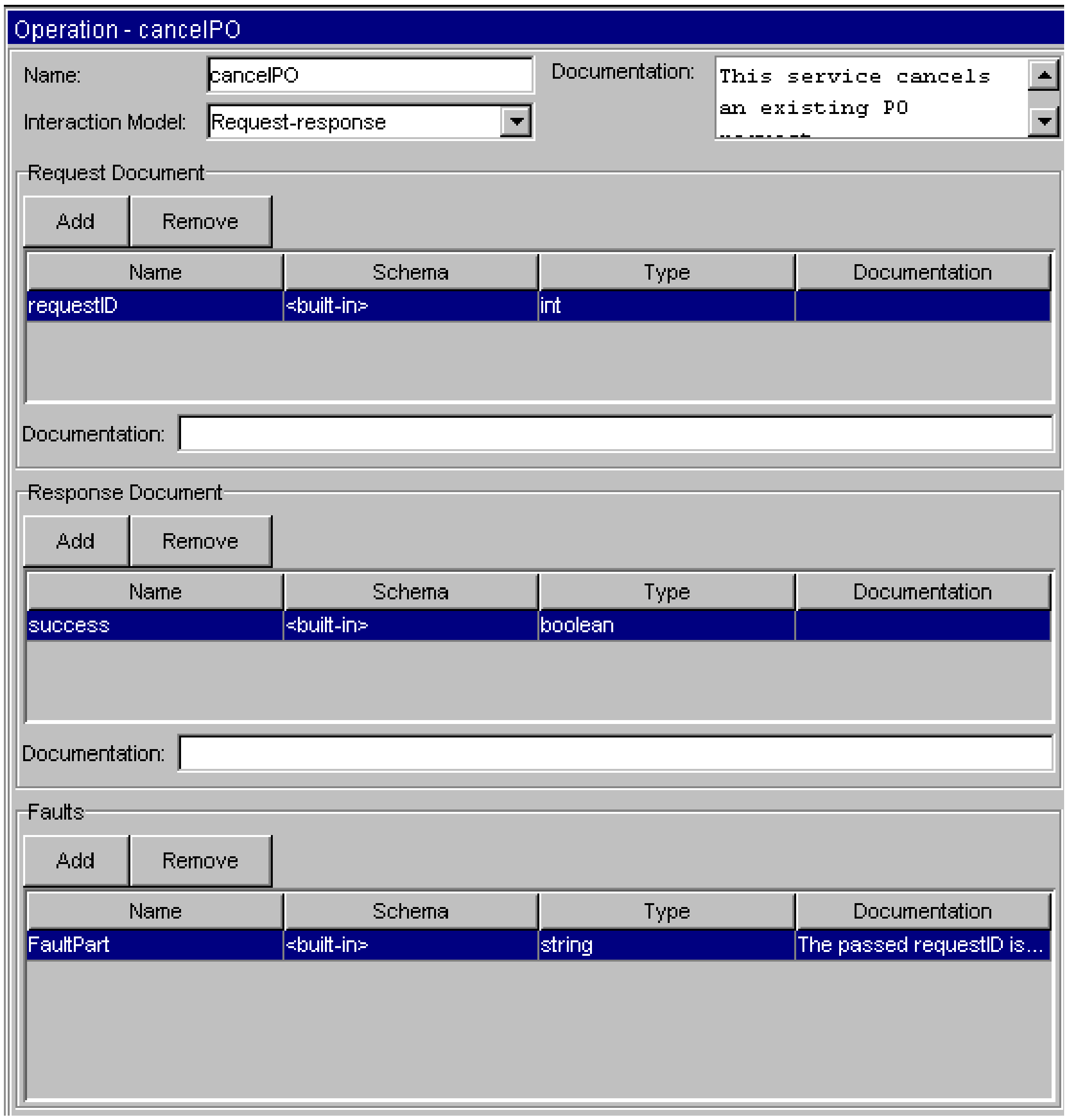
A project represents one or more services that make up a coherent set of functionality that the user is operating on. Projects are saved in files with the extensions **.scpj**. Projects may also be exported as WSDL files.

The Desktop

The hp Service Composer’s “desktop” is shown in Figure 1: "The Desktop". It consists of the following parts:

- The **Menu Bar** runs across the top of the HPSC desktop, and contains all the contextually relevant commands that can be invoked.
- The **Project Tree** takes up most of the left side of the HPSC desktop and displays a hierarchical view of all the service elements that make up the Web Service project. In the example below, the Project Tree pane is labeled “Project – demo_scpj”.
- The **Content Pane** takes up most of the right side of the HPSC desktop and displays an appropriate editor for the service element that is currently selected in the project tree. In the figure below, the Operation Editor is currently displayed in the Content Pane.
- The **Message Pane** lies horizontally near the bottom of the HPSC desktop and displays informational, warning, and error messages that occur during the user session. In the figure below, the Message Pane is labeled “Messages”.
- The **Status Pane** lines horizontally below the Message Pane, and displays status information and progress bars during long-running operations such as file reads and writes.

Each part is described in more detail in subsequent sections.



Operation - cancelPO

Name: cancelPO

Interaction Model: Request-response

Documentation: This service cancels an existing PO

Request Document

Name	Schema	Type	Documentation
requestID	<built-in>	int	

Documentation:

Response Document

Name	Schema	Type	Documentation
success	<built-in>	boolean	

Documentation:

Faults

Name	Schema	Type	Documentation
FaultPart	<built-in>	string	The passed requestID is...

Figure 1: The Desktop

The Menu Bar

The Menu Bar lies across the top of the HPSC desktop and contains all the contextually relevant commands that can be invoked by the developer. Contextually relevant means that the set of commands available via the menu bar may change depending on the state of the project and the Service Element that is selected.

The menu bar contains the following menus:

File

As shown in Figure 2: "The File Menu", the file menu contains the following commands.

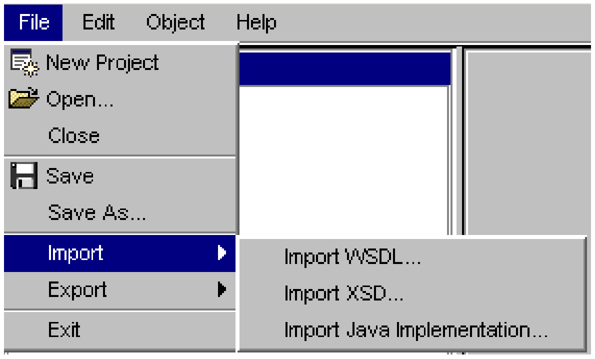


Figure 2: The File Menu

- **New Project:** create a new empty project.
- **Open…:** allows you to open an existing project (i.e., a file with a .scpj extension).
- **Close:** closes the current project.
- **Save:** saves the current project using the current pathname for the project. If the project has never been saved, the user is asked for a pathname.
- **Save As…:** allows you to save the current project under a different name.
- **Import:** contains commands that will build new projects from other files and sources.
  - — **Import WSDL…** allows you to create a new project from an existing WSDL file.
  - — **Import XSD…** allows you to create a new project from an existing XSD file.
  - — **Import Java Implementation…** allows you to create a new project from an existing java class file or EJB. This command will cause the Java Import Wizard to appear.
  - — **Export Project as WSDL...**: allows you to export the projevct as a WSDL file.

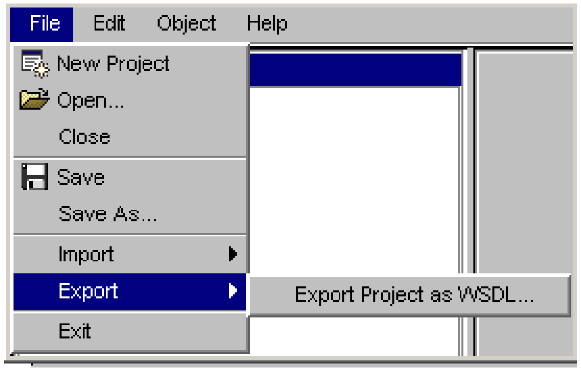


Figure 3: Export Project as WSDL

Exit: exits the HPSC application.

Edit

As shown in Figure 3: "The Edit Menu", the Edit menu contains the following commands:

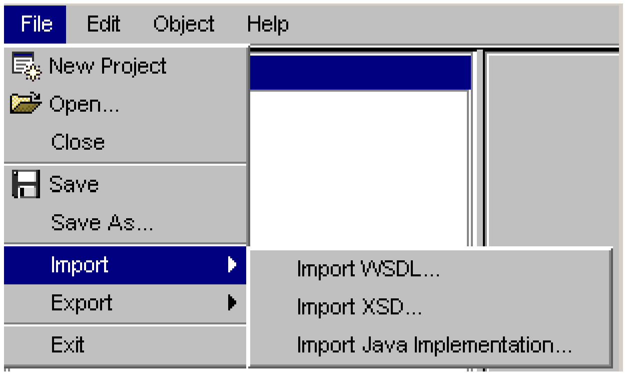
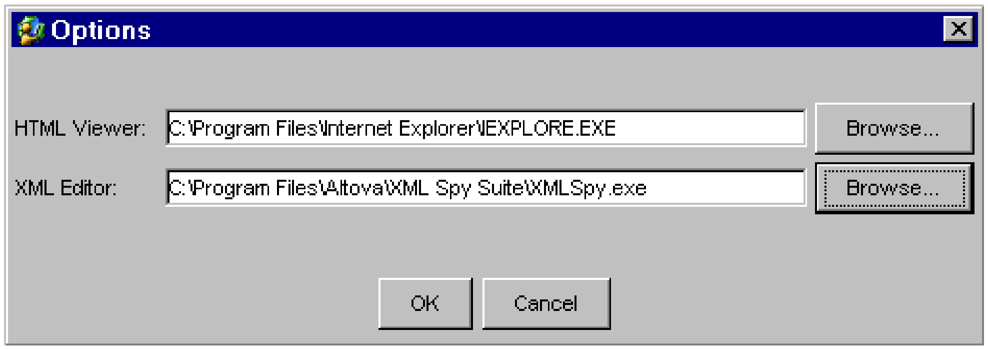


Figure 4: The Edit Menu

- **Options…** displays the Options dialog.



HTML Viewer: C:\Program Files\Internet Explorer\IEXPLORE.EXE

XML Editor: C:\Program Files\Altova\XML Spy Suite\XMLSpy.exe

Figure 5: The Options Dialog

The Options dialog permits the developer to change the following HPSC properties:

— **HTML Viewer** is the application that should be executed to view the HTML formatted documents. Examples of applicable viewers are Internet Explorer and Netscape. A text editor can also be used as an HTML viewer if desired.

— **XML Editor** is the application that should be used as the external XML schema editor. Currently, an external XML editor's use is entirely optional since HPSC has its own built-in schema editor. Examples of applicable applications include Tibco Turbo XML and XML Spy. A text editor can also be used as the XML schema editor if desired.

Object

The Object menu displays commands that can be applied to the currently selected Web Service Element. The menu displayed is always identical to the pop-up menu which can be seen by right-clicking on the Web Service Element which is currently selected in the Project Tree. For example, Figure 4: "The Object Menu" below shows the object menu for a Service object.

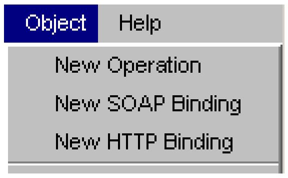


Figure 6: The Object Menu

These are the object menu commands for each class of the Web Service Elements:

- **Attribute**
  - — **Set Type** changes the type of the attribute to the selected XSD built-in type. Note that only the most common built-in types are currently available in the submenu. Use the schema editor's property dialog for a complete list of available simple types.
  - — **Optional** toggles the optional property of the attribute.
  - — **Delete** deletes the attribute from the project.
- **Complex Type**
  - — **New attribute** adds a new attribute to the complex type.
  - — **New element** adds a new element to the complex type.
  - — **Delete** deletes the complex type from the project.
- **Element**
  - — **Set Type** changes the type of the element to the selected XSD built-in type. Note that only the most common built-in types are currently available in the submenu. Use the schema editor's property dialog for a complete list of available simple and complex types.
  - — **Optional** toggles the optional property of the element. Optional elements have a min occurs setting of 0, while required elements have a min occurs setting of 1.
  - — **Bounded** toggles the bounded property of the element. Bounded elements have a **max occurs** setting of 1, while unbounded elements have an infinite max occurs value.
  - — **Delete** deletes the element from the project.
- **Group**
  - — **New element** adds a new element to the group.
- **Operation**
  - — **Delete** deletes the operation from the project.
- **Port**
  - — **Delete** deletes the port from the project.

- **Port Type**
  - **New Operation** adds a new operation to the port type.
  - **New SOAP Binding** adds a new SOAP service binding to the port type.
  - **New HTTP Binding** adds a new HTTP service binding to the port type.
  - **Export as WSDL…** allows you to export this port type as a WSDL file. The resultant WSDL will only contain the selected port type and any supporting service elements. This can be used to extract a portion of an overall project for reuse, etc.
  - **Delete** deletes the port type from the project.
- **Schema**
  - **Edit with external editor** launches the external XML editor specified in the Edit->Options dialog. The current schema is written out to a temporary file and the external editor is launched with the temporary file as an argument. You can modify the schema and save it as usual, i.e., overwriting the temp file. When the desktop is reactivated, HPSC will notice that the temporary file is modified and will reload the schema.
  - **Export as XSD schema…** allows you to save the schema out to a XSD file.
  - **New attribute** adds a new global attribute to the schema.
  - **New complex type** adds a new global complex type to the schema.
  - **New element** adds a new global element to the schema.
  - **New simple type** adds a new global simple type to the schema. The simple type submenu is used to specify the built-in type that the new simple type is to extend. Note that only the most common built-in types are currently listed.
  - **Delete** deletes the schema from the project.
- **Schema Folder**
  - **New Schema** creates a new empty schema in the project.
  - **Import Schema…** allows you to load an existing XSD file into the current project. A new schema element is created.
- **Service**
  - **New Port Type** creates a new port type interface in the project.
  - **Export as WSDL…** allows you to export the service as a WSDL file.
  - **Delete** deletes the service from the project.
- **Service Binding (HTTP/SOAP)**
  - **New Port** adds a new port to the service binding. This can be used to add a new service access point to a given service binding.
  - **Export as WSDL…** allows you to export this service binding as a WSDL file. The resultant WSDL file will only contain the selected service binding and any supporting service elements. This can be used to extract a portion of an overall project for reuse, etc.

- **Generate Implementation…** allows you to create client proxies and/or service skeletons for the service binding via the "Java Proxy/Skeleton Generation Wizard".
- **Deploy Service Binding…** creates a web service deployment file suitable for deploying the service described by the current service binding. This command is only available when implementation files are associated with the binding.
- **Delete** deletes the service binding from the project.

- **Service Folder**
  - **New Service** creates a new empty service in the project.
- **Simple Type**
  - **Delete** deletes the simple type from the project.

Help

The Help menu offers a number of resources for better understanding hp Service Composer and Web Services technology (see Figure 5: "The Help Menu").

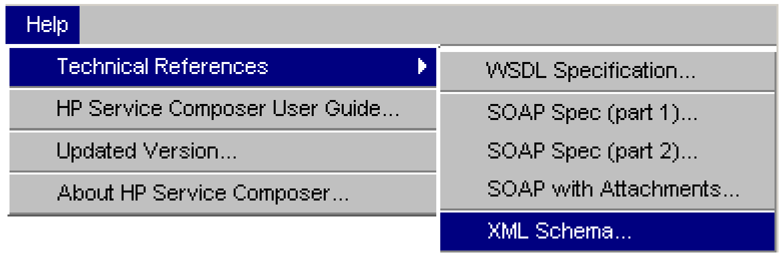


Figure 7: The Help Menu

The technical references listed and the HPSC User's guide all require that a valid HTML viewer be specified in the Edit->Options dialog.

The **About** menu item displays the HPSC splash screen. Click the mouse once to dismiss the About screen.

The Project Tree

The project tree displays all the model elements that make up the current web services project.

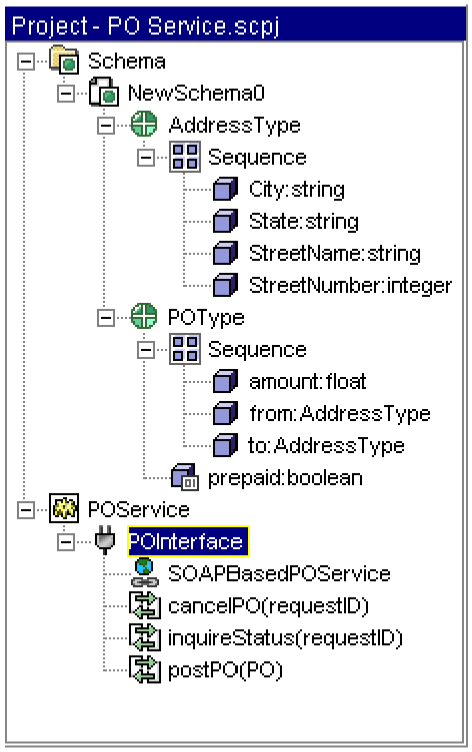


Figure 8: The Project Tree

The tree is organized in a simple hierarchy to show the containment relationship between the different elements. For example, **Sequences** contain zero or more **Elements** and/or **Attributes**; **Port types** contain zero or more **Operations** and/or **Service Bindings**; etc.

Table 1: "Project Tree Icons" shows the mapping between icons used in the project tree and the associated web service model element.

Table 1 Project Tree Icons

Icon	Model Element
▣	Attribute
⊕	Complex type
■	Element
⊞	Group (sequence)

Icon	Model Element
📄	Java Implementation File
⇄	Operation
🔌	Port
🔌	Port Type or Interface
📄	Schema
📁	Schema Folder
🌐	Service Binding
⚙	Service Definition
📁	Service Folder

When you click on a service model element in the project tree, that element becomes selected and the content pane updates to display an appropriate editor. For example, if you click on Operation the Project Tree will display the Operation Editor in the content area. Through the displayed editor, you can modify properties of the selected model element, create new model elements, etc.

In addition, the Object menu in the menu bar will display commands that can be applied to the selected item. The Object menu for a model element can also be accessed by right-clicking on that model element to expose the Object pop-up menu. See the "Object" section above for more details.

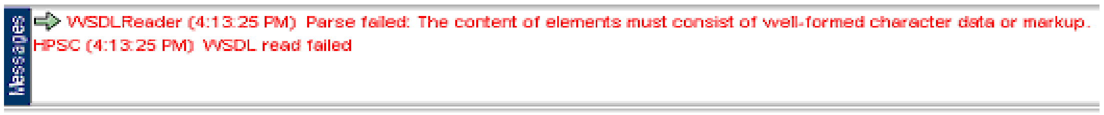
The name of any service model element can be edited via the project tree by triple-clicking on that element.

The Message Window

The Message Window displays information, warnings, and errors that occur during a HPSC session. If additional information is available about an error (such as the context of the error), the error message is prefixed with an arrow icon. Such messages are said to have an associated message context. Clicking on the message will cause the content pane to display the associate message context.

For example, the first line shown in Figure 7: "The Message Window" indicates that a read WSDL file contained errors that prevented a correct parse of the file. Clicking on that line will cause the WSDL file to be displayed in the content pane with the problem area highlighted.

Right clicking on the message area will display a pop-up menu containing a single command "clear all". This command will clear the message pane contents. The message pane will also be cleared automatically when a new project is created or loaded.



```
WSDLReader (4:13:25 PM)  Parse failed: The content of elements must consist of well-formed character data or markup.
HPSC (4:13:25 PM)  WSDL read failed
```

Figure 9: The Message Window

The Status Window

The Status window displays status and progress information during long-running activities such as file loading. The example shown in Figure 8: "The Status Window" shows that a WSDL file is a little more than halfway parsed. Clicking on the cancel button will terminate the current long-running activity.

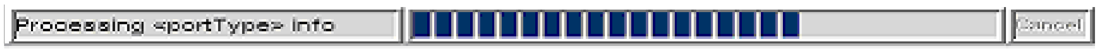
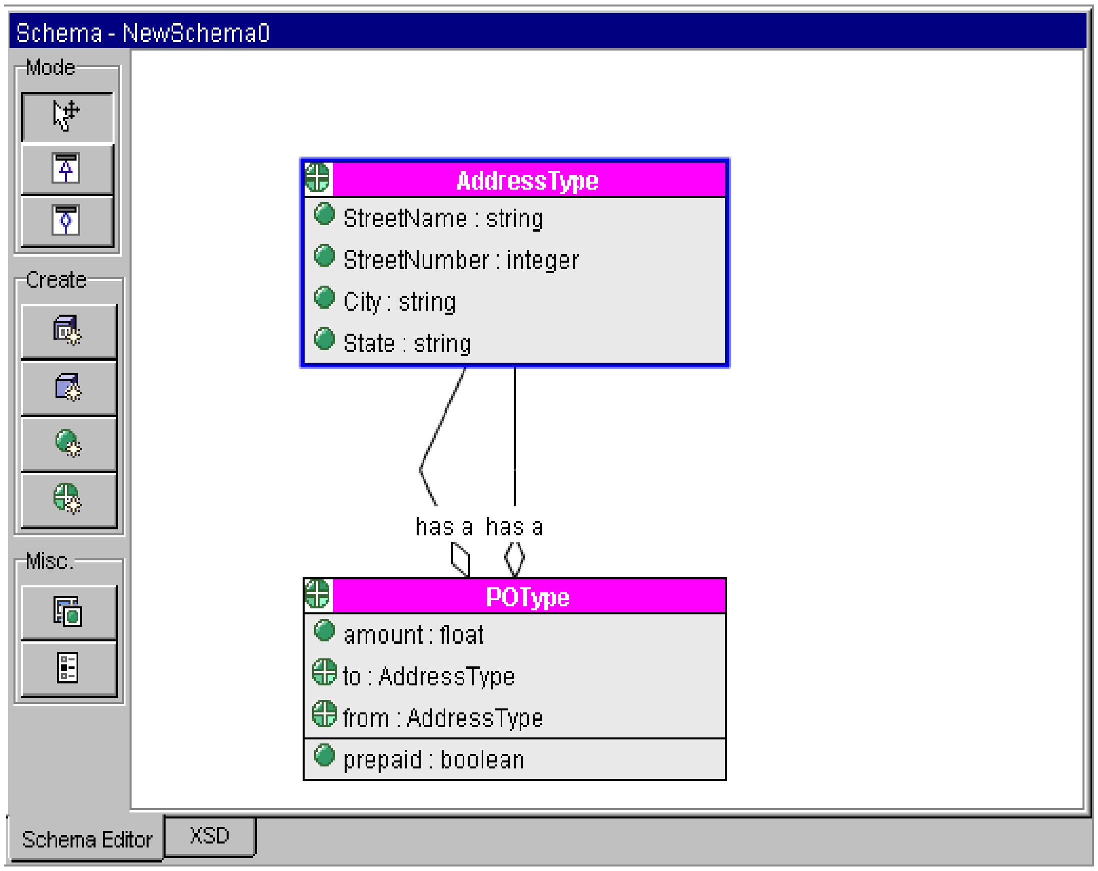


Figure 10: The Status Window

The Schema Editor

The Schema Editor is displayed when any schema element is selected in the project tree. It displays a UML-like, graphical view of a schema and its components. Using the vertical toolbar displayed on the left side of the schema editor, the developer can manipulate the graphical layout, create new schema components (attributes, elements, complex types, simple types), and connect components together into reference and type hierarchies (has-a and is-a). For example, in Figure 9: "The Schema Editor" below, the complex type POType is shown to contain two elements that have AddressType type. The "has-a" links between the two complex types makes this relationship explicit.



AddressType
- StreetName : string
- StreetNumber : integer
- City : string
- State : string

POType
- amount : float
- to : AddressType
- from : AddressType
- prepaid : boolean

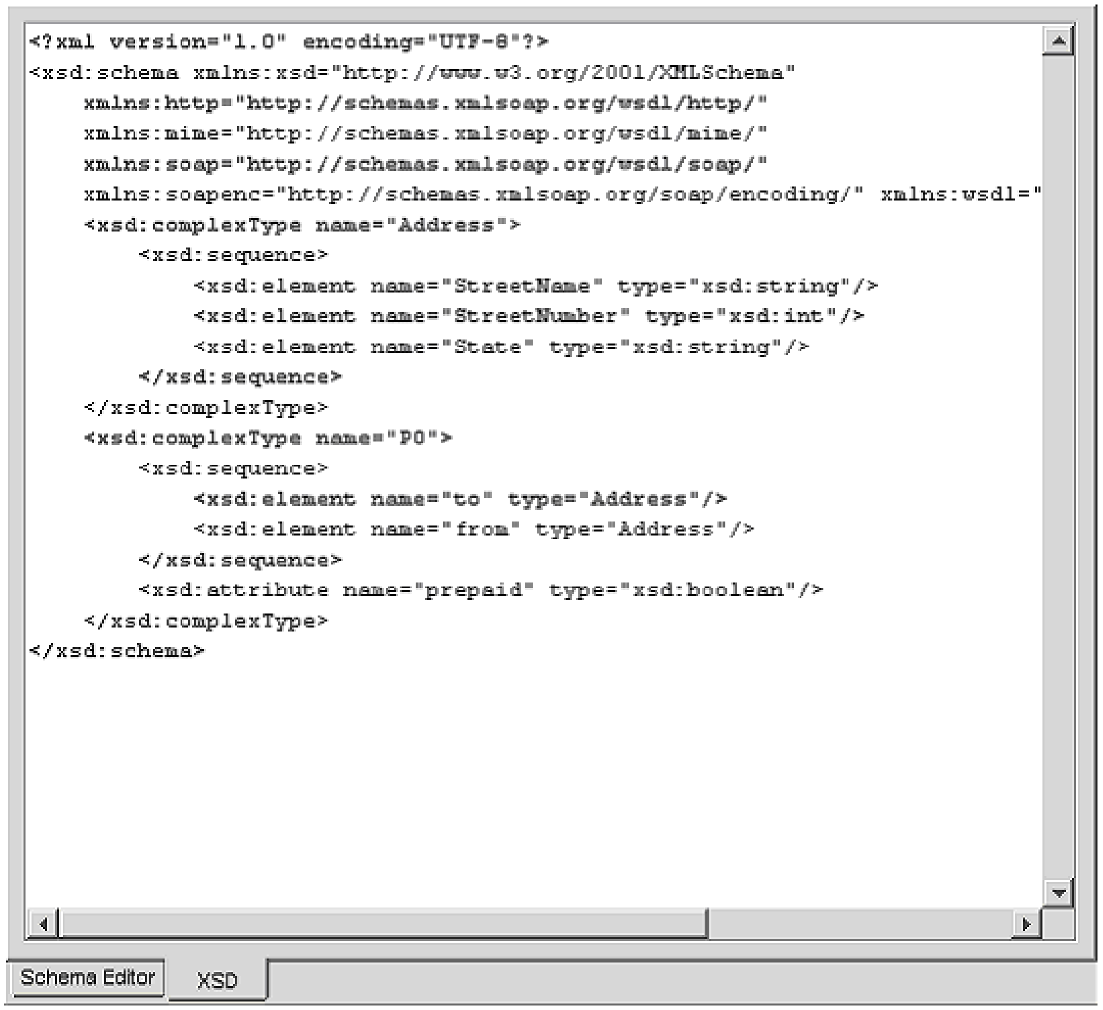
Figure 11: The Schema Editor

Table 2: "Toolbar Buttons" below describes the actions performed by each button in the toolbar. In addition, a property editor for a schema component can be exposed by double clicking on that component in the graphical view. Through the property editor, you can change the components name, change type information, and add/remove subcomponents.

Table 2 Toolbar Buttons

Icon	Operation
⊕	New complex type
●	New simple type
▣	New attribute
▣	New element
↖	Select the Move/Select mode
△	Connect with IS-A mode
◇	Connect with HAS-A mode
▣	schema_autolayout
☰	Edit properties

At any time during the viewing or editing of a schema, the associated XSD file can be prefixed by clicking on the XSD tab of the schema editor. See Figure 10: "An XSD Example" for an example of this XSD preview feature.



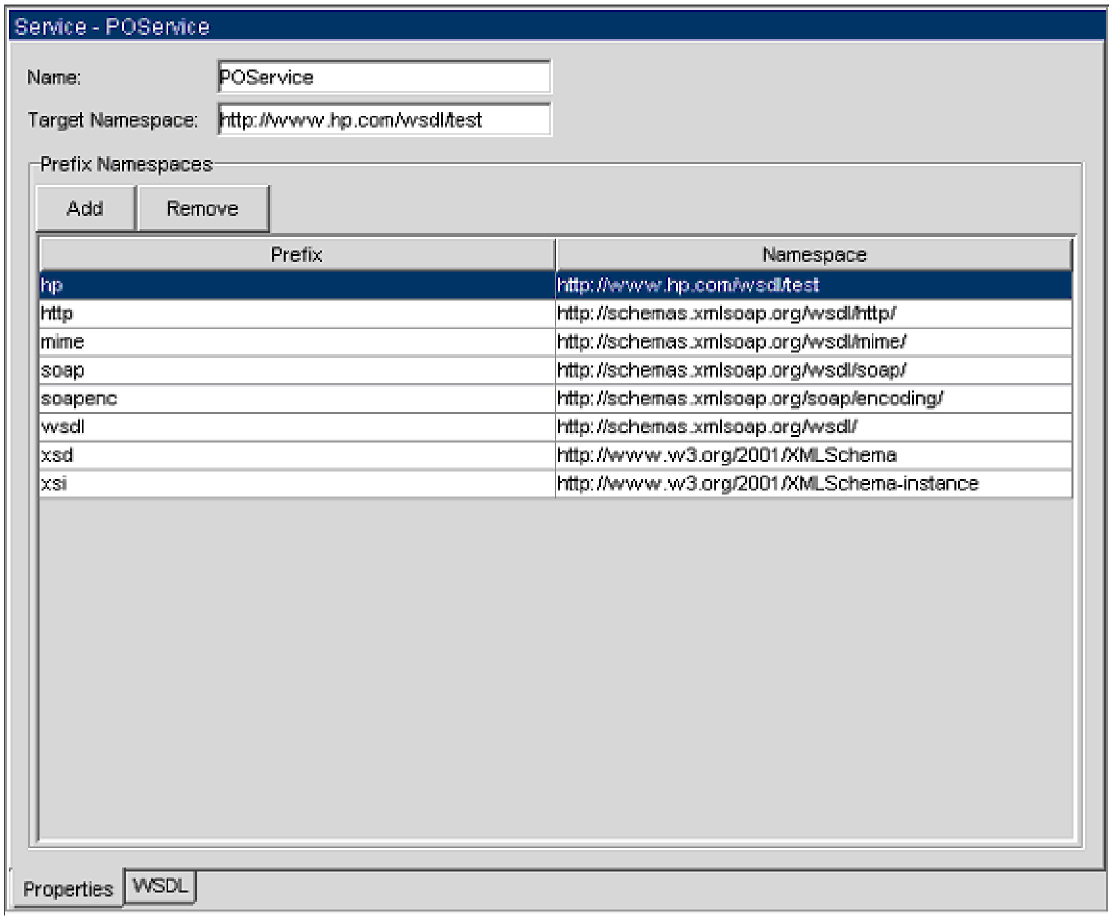
```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:wsdl="
    <xsd:complexType name="Address">
        <xsd:sequence>
            <xsd:element name="StreetName" type="xsd:string"/>
            <xsd:element name="StreetNumber" type="xsd:int"/>
            <xsd:element name="State" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="PO">
        <xsd:sequence>
            <xsd:element name="to" type="Address"/>
            <xsd:element name="from" type="Address"/>
        </xsd:sequence>
        <xsd:attribute name="prepaid" type="xsd:boolean"/>
    </xsd:complexType>
</xsd:schema>
```

Figure 12: An XSD Example

The Service Editor

When a Service is selected in the project tree, the service editor is displayed in the content pane. Through this editor, the developer can change the service name, assign/view a target prefix for model elements within that service, and designate/view prefixes associated with namespaces that are referenced within the Service definition.

In Figure 11: "The Service Editor" below, the POService is being edited, with service model elements such as port types and bindings being placed into the http://www.hp.com/test target namespace. In addition, the prefix "hp" is also associated with this target namespace. In new projects, HPSC automatically assigns default prefixes to namespaces that are commonly referenced within a WSDL definition.



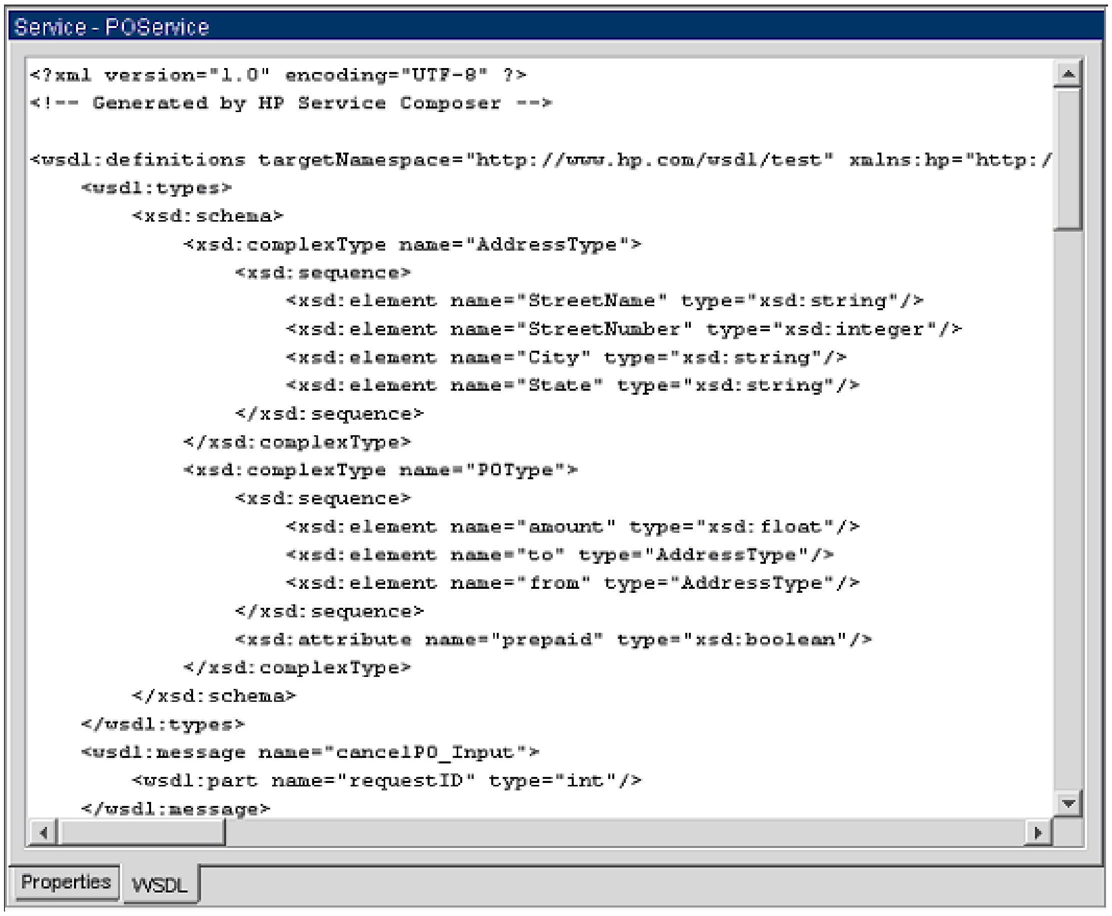
Name: POService

Target Namespace: http://www.hp.com/wsdl/test

Prefix	Namespace
hp	http://www.hp.com/wsdl/test
http	http://schemas.xmlsoap.org/wsdl/http/
mime	http://schemas.xmlsoap.org/wsdl/mime/
soap	http://schemas.xmlsoap.org/wsdl/soap/
soapenc	http://schemas.xmlsoap.org/soap/encoding/
wsdl	http://schemas.xmlsoap.org/wsdl/
xsd	http://www.w3.org/2001/XMLSchema
xsi	http://www.w3.org/2001/XMLSchema-instance

Figure 13: The Service Editor

At any time during the viewing or editing of a service definition, the associated WSDL file can be previewed by clicking on the WSDL tab of the service editor. Figure 12: "WSDL File Preview" shows an example of such as WSDL file view.



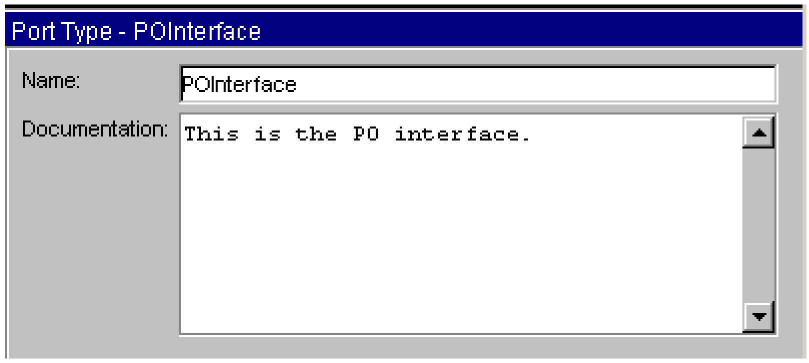
```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- Generated by HP Service Composer -->

<wsdl:definitions targetNamespace="http://www.hp.com/wsdl/test" xmlns:hp="http:/
    <wsdl:types>
        <xsd:schema>
            <xsd:complexType name="AddressType">
                <xsd:sequence>
                    <xsd:element name="StreetName" type="xsd:string"/>
                    <xsd:element name="StreetNumber" type="xsd:integer"/>
                    <xsd:element name="City" type="xsd:string"/>
                    <xsd:element name="State" type="xsd:string"/>
                </xsd:sequence>
            </xsd:complexType>
            <xsd:complexType name="POType">
                <xsd:sequence>
                    <xsd:element name="amount" type="xsd:float"/>
                    <xsd:element name="to" type="AddressType"/>
                    <xsd:element name="from" type="AddressType"/>
                </xsd:sequence>
                <xsd:attribute name="prepaid" type="xsd:boolean"/>
            </xsd:complexType>
        </xsd:schema>
    </wsdl:types>
    <wsdl:message name="cancelPO_Input">
        <wsdl:part name="requestID" type="int"/>
    </wsdl:message>
```

Figure 14: WSDL File Preview

The PortType Editor

When a Port Type is selected in the project tree, the port type editor is displayed in the content pane. This simple editor allows the developer to change/view the port type's name, and edit/ view any associated port type documentation string.



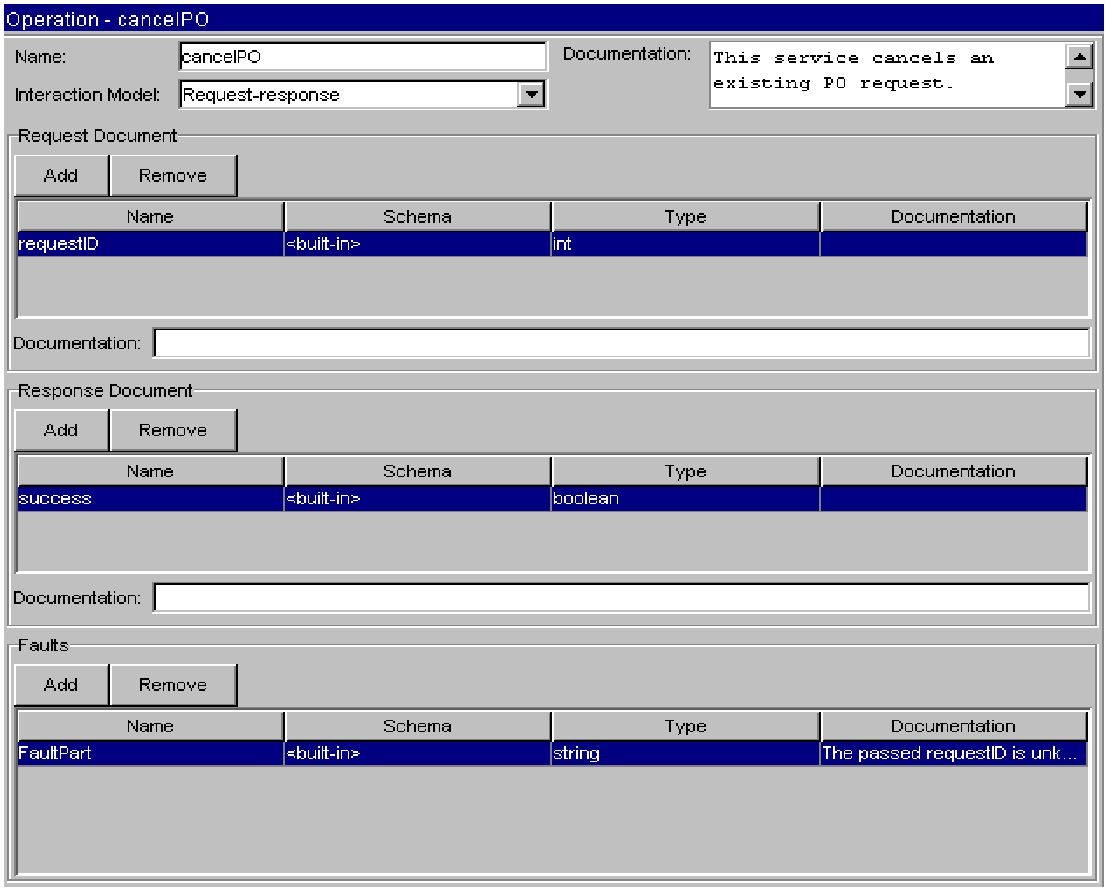
Name: POInterface

Documentation: This is the PO interface.

Figure 15: The PortType Editor

The Operation Editor

When an operation is selected in the Project Tree, the Operation Editor is displayed in the Content Pane.



Name: cancelPO
Interaction Model: Request-response
Documentation: This service cancels an existing PO request.

Request Document

Name	Schema	Type	Documentation
requestID	<built-in>	int	

Documentation:

Response Document

Name	Schema	Type	Documentation
success	<built-in>	boolean	

Documentation:

Faults

Name	Schema	Type	Documentation
FaultPart	<built-in>	string	The passed requestID is unk...

Figure 16: The Operation Editor

Properties of the Operation

The Operation Editor allows the developer to view and edit properties of the operation including:

- The operation’s **name.**
- The operation’s **interaction model**. This can be one of the following:
  - **Request-response**, in which the service receives and then responds to a request. This is the default interaction model and the most commonly used for synchronous services accessed across HTTP.
  - **One-way**, in which the service receives a request that it processes without responding. Note that there is no response document in this model.
  - **Solicit-response**, in which the service sends out a solicitation message to an end-point and expects a response in return.
  - **Notification**, in which the service sends out a notification message, but doesn’t expect any response from the end-point. Note that there is not request document in this model.

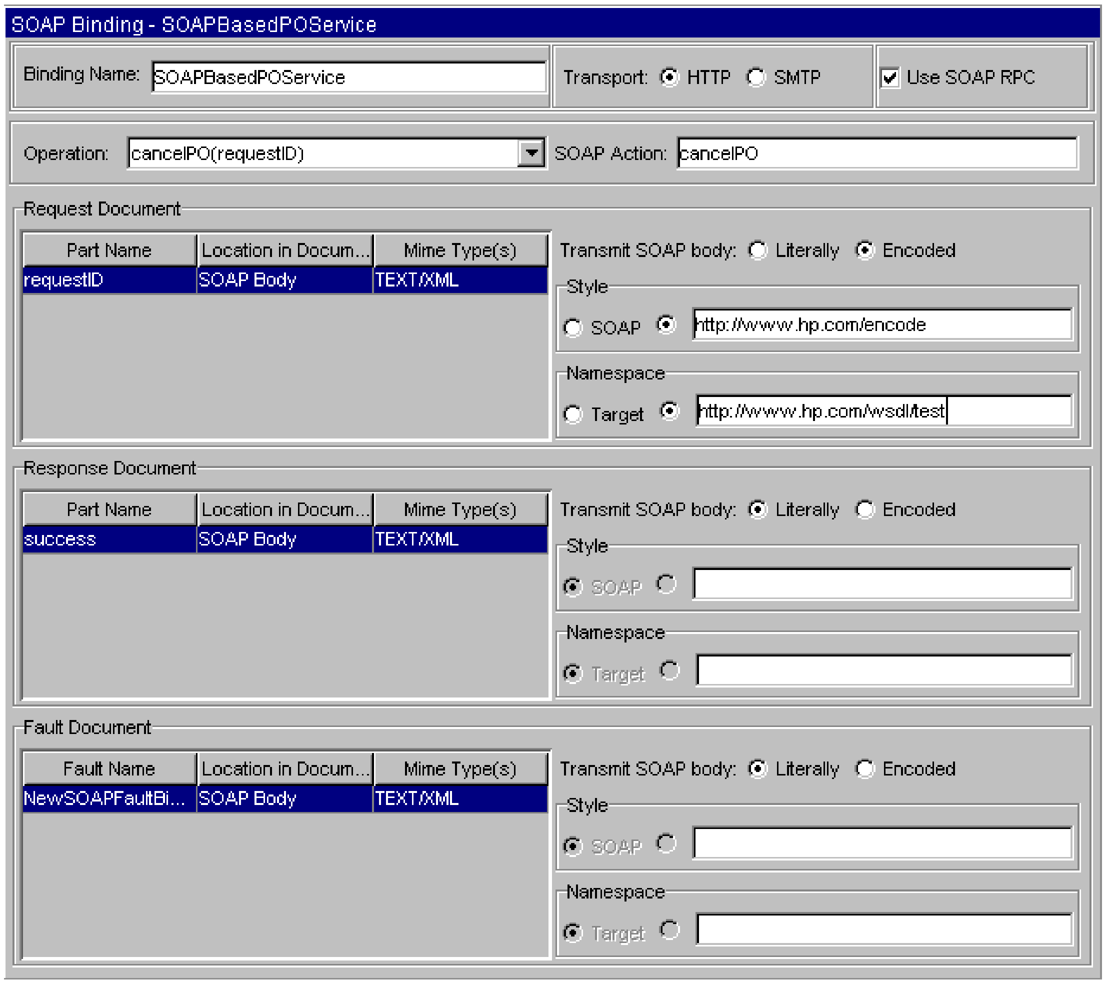
Documentation About the Operation

The following are the various documents that follow the operation of the Operation Editor:

- The **Request Document** is expected to be received by the service. Note that not all interaction models involve a request document. When applicable, however, the request document may contain one or more named *parts*, each part having an assigned type (simple or complex) and documentation string. Parts may then be bound to different Mime parts and may receive different encoding within a service binding. See the sections on "The HTTP Binding Editor" and "The SOAP Binding Editor" for details. Finally note that the request document as a whole may also have an associated documentation string.
- The **Response Document** is expected to be sent by the service. Note that not all interaction models involve a response document. When applicable, however, the response document may contain one or more named *parts*, each part having an assigned type (simple or complex) and documentation string. Parts may then be bound to different MIME parts and may receive different encoding within a service binding. See the sections on "The HTTP Binding Editor" and "The SOAP Binding Editor" for details. Finally, note that the response document as a whole may also have an associated documentation string.
- Zero or more **Fault Documents** that may be sent by the service if some application-level exception is encountered.

The SOAP Binding Editor

When a SOAP Binding is selected in the project tree, the SOAP Binding editor is displayed in the content area.



SOAP Binding - SOAPBasedPOService

Binding Name: SOAPBasedPOService
Transport: [✓ HTTP] SMTP
[✓ Use SOAP RPC]

Operation: cancelPO(requestID)
SOAP Action: cancelPO

Request Document

Part Name	Location in Docum...	Mime Type(s)
requestID	SOAP Body	TEXT/XML

Transmit SOAP body: Literally [✓ Encoded]
Style: SOAP [✓ http://www.hp.com/encode]
Namespace: Target [✓ http://www.hp.com/wsdl/test]

Response Document

Part Name	Location in Docum...	Mime Type(s)
success	SOAP Body	TEXT/XML

Transmit SOAP body: [✓ Literally] Encoded
Style: [✓ SOAP]
Namespace: [✓ Target]

Fault Document

Fault Name	Location in Docum...	Mime Type(s)
NewSOAPFaultBi...	SOAP Body	TEXT/XML

Transmit SOAP body: [✓ Literally] Encoded
Style: [✓ SOAP]
Namespace: [✓ Target]

Figure 17: The SOAP Binding Editor

Properties of the SOAP Service Binding

This editor allows the developer to view and edit properties of the SOAP Service Binding including:

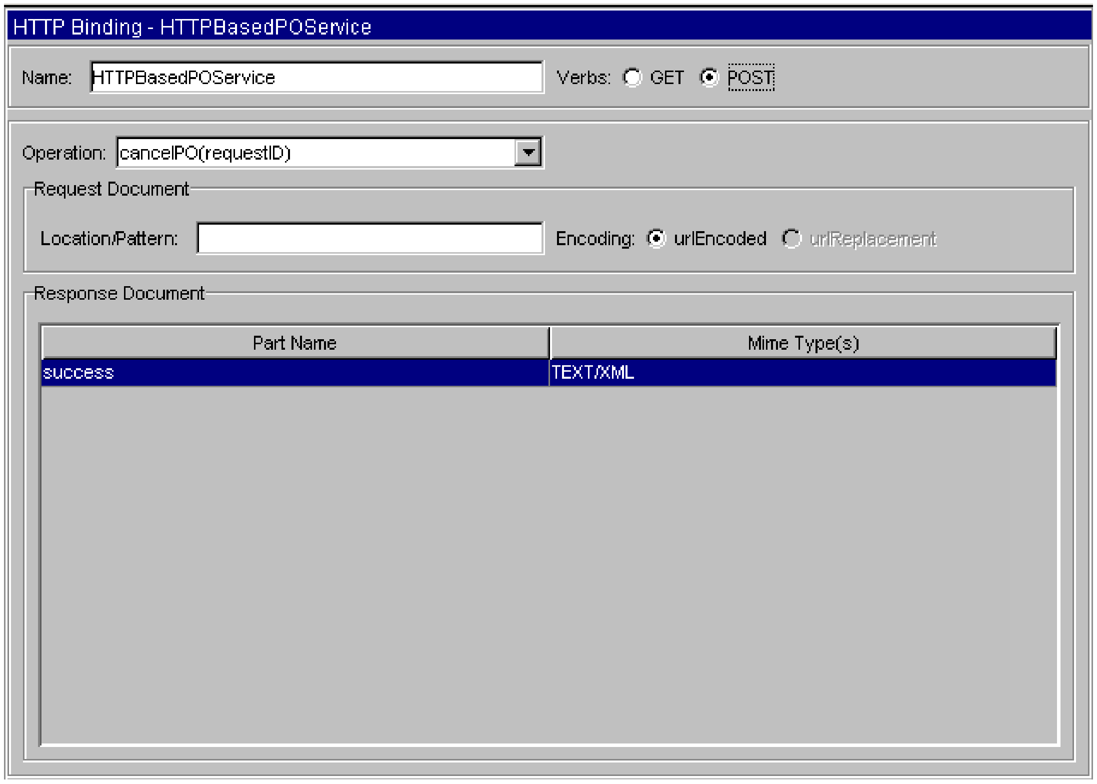
- The **Name** for the binding.
- The **Transport** to be used to access the service (either HTTP or SMTP).
- Whether **SOAP RPC** encoding should be used or not when communicating with the service. As discussed briefly in "Simple Object Access Protocol (SOAP)" in the Chapter "Concepts and Terminology", SOAP-RPC is an optional protocol used with SOAP for encoding remote procedural calls. Essentially, request documents that are meant to invoke an operation within a service that uses SOAP-RPC will be wrapped in an element named after the operation to be invoked. This wrapper can be used by the receiving SOAP server to invoke the appropriate back-end operation.

In addition, for each operation defined by the binding (or more specifically, by the port type that contains the binding), the following binding properties can be set:

- The **SOAP Action** that is expected to accompany service requests for that operation. This string is sent in an HTTP request using the SOAP-Action HTTP header extension. As discussed briefly in "SOAP Action" in Chapter "Concepts and Terminology", this string can be used to program firewalls to improve security.
- The location to be used to transport each **Request and Response Document Parts**. By default, each part will be located in the body of the SOAP envelope being used to transport the message. However, by using the binding editor, the developer can specify that one or more parts should instead be transmitted in one or more MIME parts. In this case, the resulting document will potentially be a MIME Multipart document. To provide alternate potential MIME types for a part, type each part into the MIME type(s) column, separating the types with a comma. For example, to indicate that the "myPicture" part may be transmitted as a "gif" or a "jpeg", you would type "image/gif, image/jpeg" as the MIME type for the "myPicture" part.
- The **Encoding** to be used (if any) for parts transmitted within the SOAP Body. Selecting the **Transmit SOAP Body: Encoded** option will require that the part contents in the SOAP body be encoded using the designated encoding style and namespace. Selecting the **Transmit SOAP Body: Literally** option will cause the part's data to be transported "literally", without additional encoding. When encoding is specified, the default encoding scheme is SOAP, and the default namespace is the same as the target namespace for the service definition. These defaults can be overridden as needed.
- The type of encoding to be used to transport any **Fault Documents**. At most one fault document will be sent in response to a service invocation, and Fault documents are always transported in the body of a SOAP message. However, the encoding style and namespace that will be used during transmission can be specified in the Fault Document section of this editor.

The HTTP Binding Editor

When an HTTP Binding is selected in the project tree, the HTTP Binding editor is displayed in the content area.



Name: HTTPBasedPOService Verbs: GET [✓ POST]

Operation: cancelPO(requestID)

Location/Pattern: Encoding: [✓ urlEncoded] urlReplacement

Part Name	Mime Type(s)
success	TEXT/XML

Figure 18: The HTTP Binding Editor

This editor allows you to view and edit properties of the HTTP Service Binding including:

- The **Name** for the binding.
- Whether the GET or POST **HTTP Verb** is expected to be used to request the service.

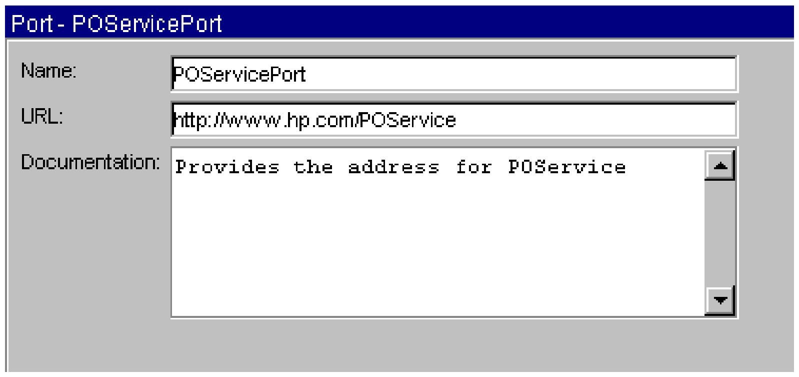
In addition, for each operation defined by the binding (or more specifically, by the port type that contains the binding), the following binding properties can be set:

- The **Encoding** style to be used when transmitting the request document parts. Accompanying this encoding style is a **Location/Pattern String,** which forms a relative address that is appended to an access point URL to invoke the service request. The format of the location/pattern string depends on the encoding style used:
  - — If **urlEncoded** is specified, then document parts are sent using standard URI encoding rules (partName0=partValue0&partName1=partValue1…).
    When used with HTTP GET, the encoded string is transmitted at the end of the URL. When used with HTTP POST, the encoded string is transmitted in the HTTP content (just as HTML form data would be). For example, a message with parts named A and B and having values of "foo" and 42 respectively would be transmitted during a HTTP GET by appending the string "A=foo&B=42" to the end of the relative URL indicated by the location string.

— If **urlReplacement** is specified, the document parts are sent in the request URI using a replacement algorithm on the indicated pattern string. Each message part to be transmitted should appear once in the pattern string using the form "(partName)". During transmission, each part pattern string will be replaced with actual part values. For example, in Figure 16: "The HTTP Binding Editor", a relative address of "cancelPO/ID4" is used for a requestID of 4.

The Port Editor

When a Port is selected in the project tree, the port editor is displayed in the content pane.


Port - POServicePort

Name: POServicePort

URL: http://www.hp.com/POService

Documentation: Provides the address for POService

Figure 19: The Port Editor

This simple editor allows you to view and edit properties of the port including:

- The **Name** for the port.
- The access point **URL** that can be used to access the port's services.
- A **Documentation** string describing the port.

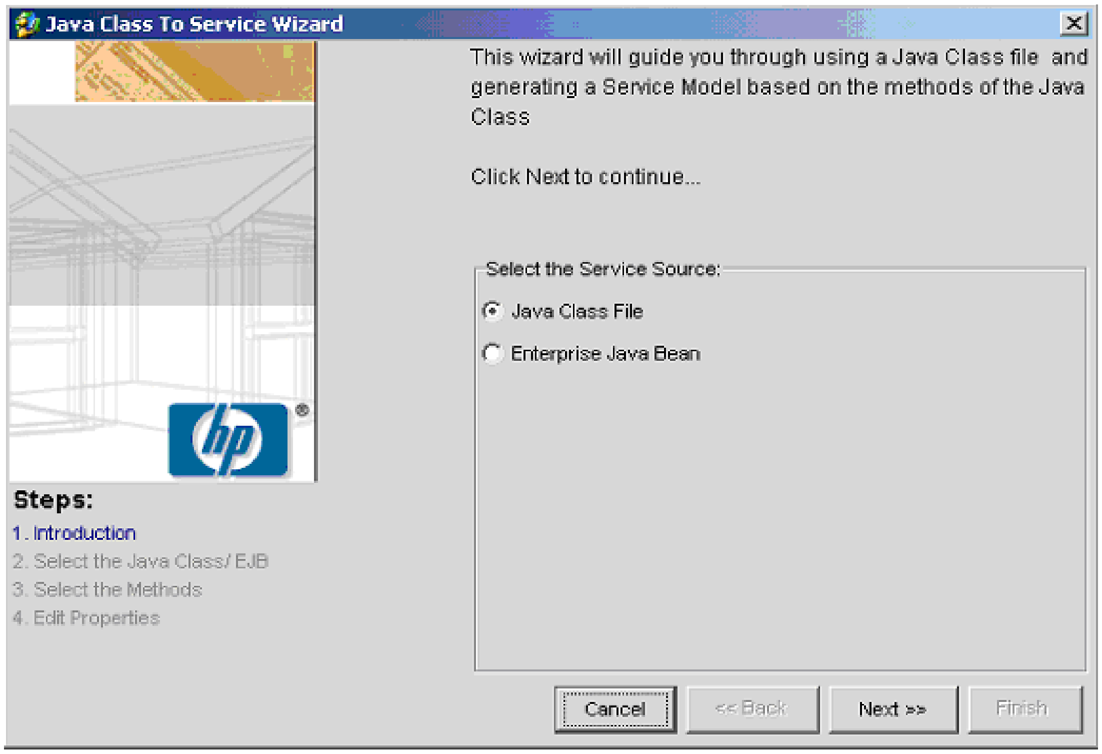
Java Import Wizard

The Java Import Wizard is exposed when the developer selects the **file->import->import java implementation…** menu item from the Menu bar. The wizard walks you through a simple four-step process that will result in the creation of a new Web Service project built on an existing Java class or EJB implementation. Through the dialog, a new Web Service model will be created for selected Java methods or Beans. The resultant model can then be deployed to a Web Services platform.

The following steps describe the dialog process:

Step 1: Select the Service Source

- Select “Java Class File” if the source to be read is a class file.
- Select “Enterprise Java Bean” if the source to be read is an EJB definition within a JAR file.
- Click on the **Next** button to continue.



This wizard will guide you through using a Java Class file and generating a Service Model based on the methods of the Java Class

Click Next to continue...

Select the Service Source:

[✓ Java Class File]

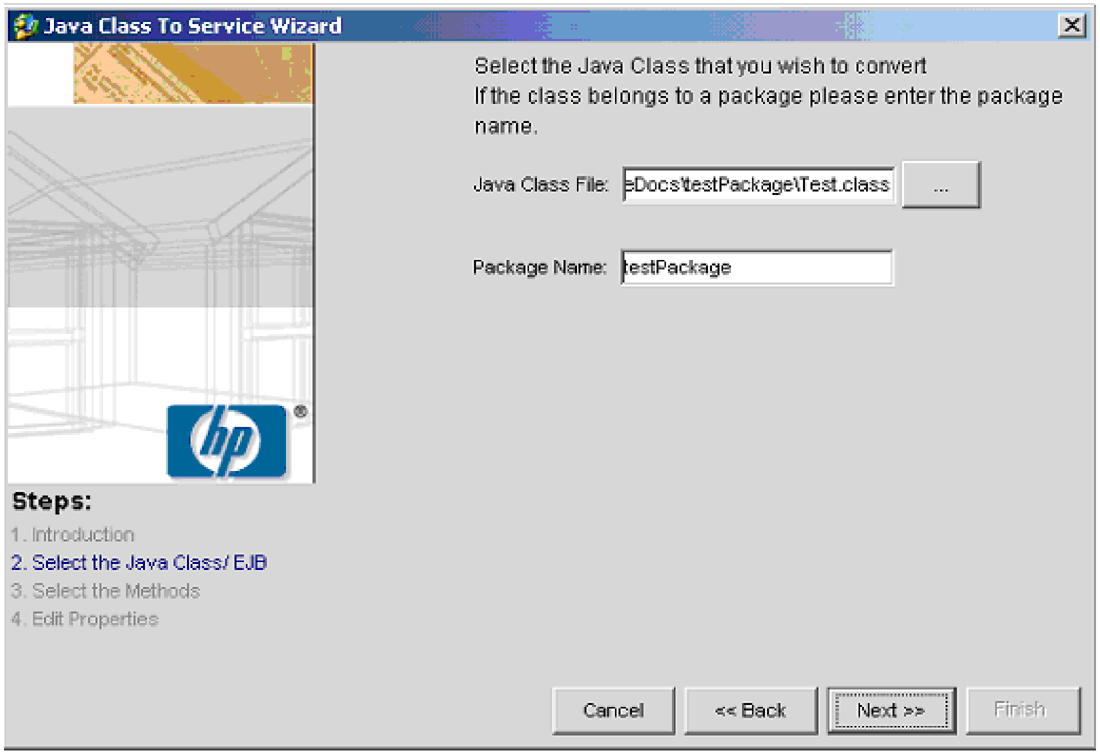
Enterprise Java Bean

Figure 20: Select the Service Source, Introduction

Step 2: Select the Class or Jar File to be Imported.

Figure 19: "Select the Java Class" shows the wizard screen displayed for importing a Java class implementation.

- Use this screen to enter the path name of a .class file to be read and the name of the package that contains the class. In this example, the Test.class file contains the class `testPackage.Test.`
- Click on the **Next** button to continue.



Select the Java Class that you wish to convert
If the class belongs to a package please enter the package name.

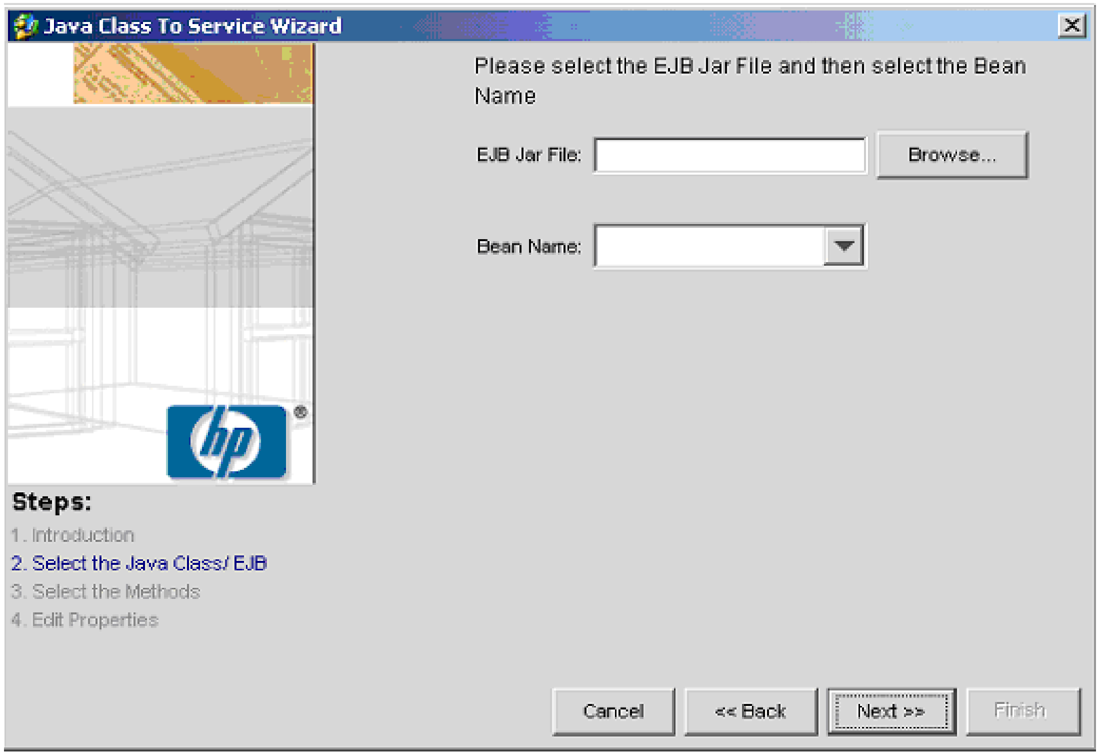
Java Class File: eDocs\testPackage\Test.class

Package Name: testPackage

Figure 21: Select the Java Class

Figure 20: "Select the EJB Jar File" shows the wizard screen for importing EJB implementations.

- Use this screen to select the jar file containing the target EJB. HPSC will parse this file and display a list of available EJBs.
- Select the target EJB from this list and click on the **Next** button.



Please select the EJB Jar File and then select the Bean Name

EJB Jar File:

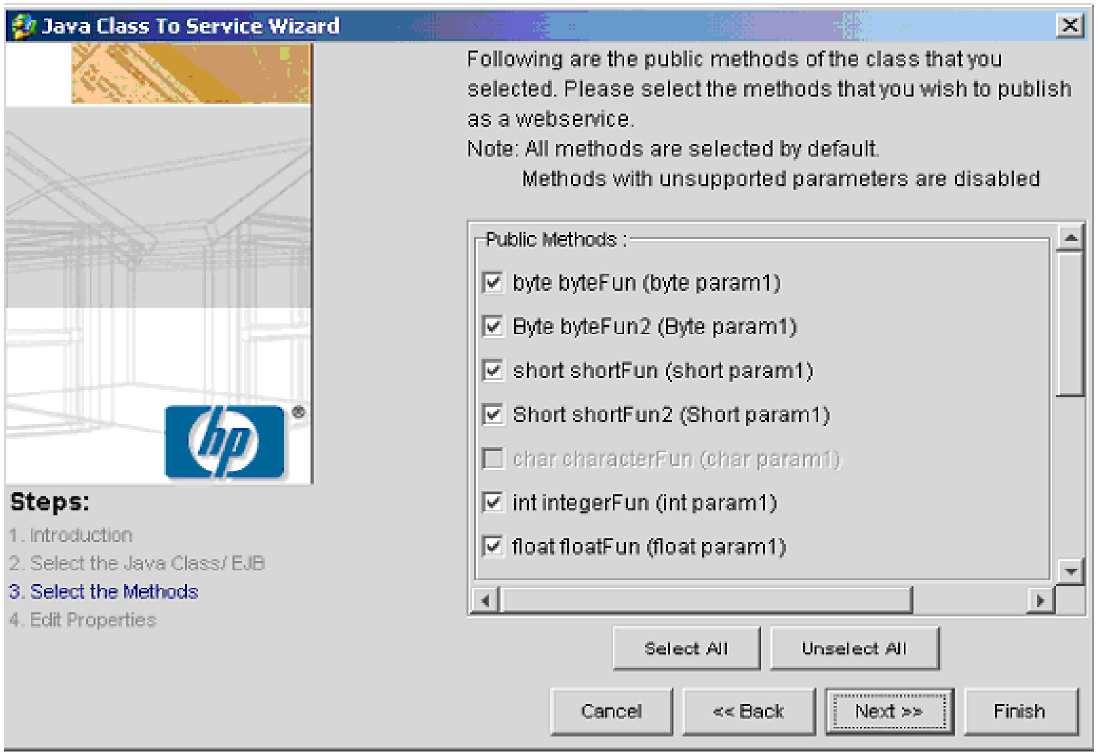
Bean Name:

Figure 22: Select the EJB Jar File

Step 3: Select the Methods or Beans to be Imported.

The wizard will now examine the Java class file or jar file and extract names of methods that can be imported. Select the methods that should be imported and then click the "Next" button (see Figure 21: "Select the Methods" for an example).

If you want to use default values for the generated model elements, you can instead click on "Finish" at this point.



Following are the public methods of the class that you selected. Please select the methods that you wish to publish as a webservice.
Note: All methods are selected by default.
Methods with unsupported parameters are disabled

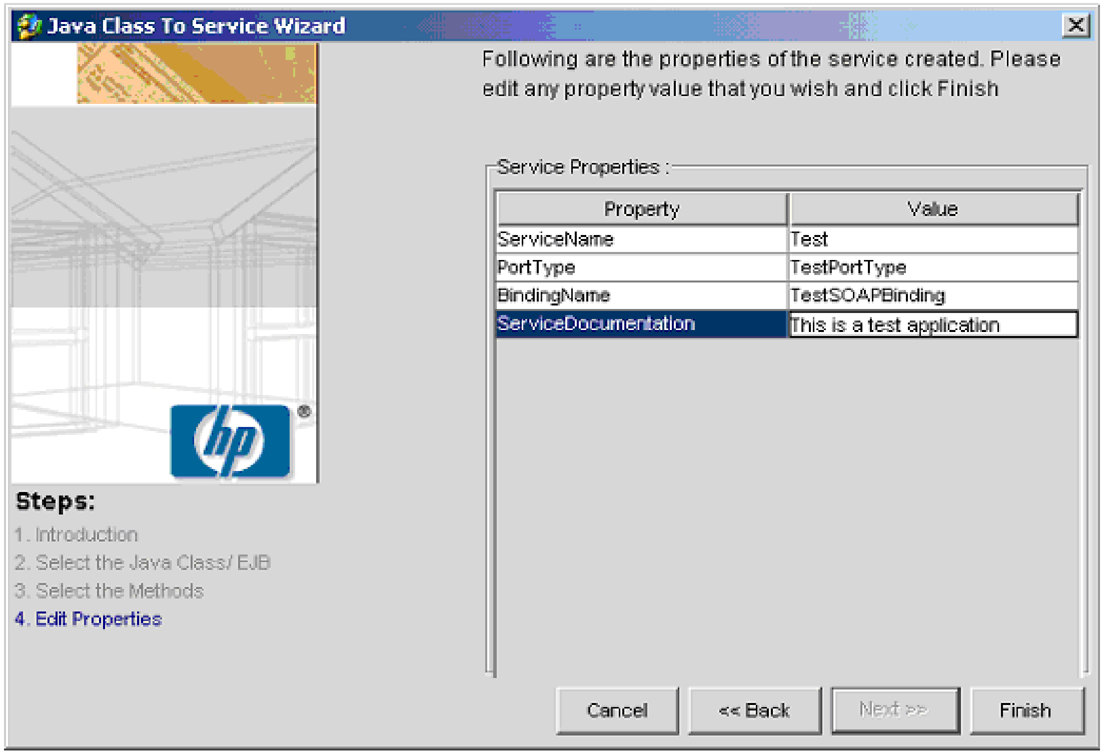
Public Methods :

- [✓] byte byteFun (byte param1)
- [✓] Byte byteFun2 (Byte param1)
- [✓] short shortFun (short param1)
- [✓] Short shortFun2 (Short param1)
- [ ] char characterFun (char param1)
- [✓] int integerFun (int param1)
- [✓] float floatFun (float param1)

Figure 23: Select the Methods

Step 4: (Optional) Modify Model Properties

This last step can be used to modify the default properties of the model elements that are created by the wizard. Finally, click "Finish" to actually generate the new project. When the wizard completes, the new project will be loaded into the HPSC project tree.



Following are the properties of the service created. Please edit any property value that you wish and click Finish

Service Properties :

Property	Value
ServiceName	Test
PortType	TestPortType
BindingName	TestSOAPBinding
ServiceDocumentation	This is a test application

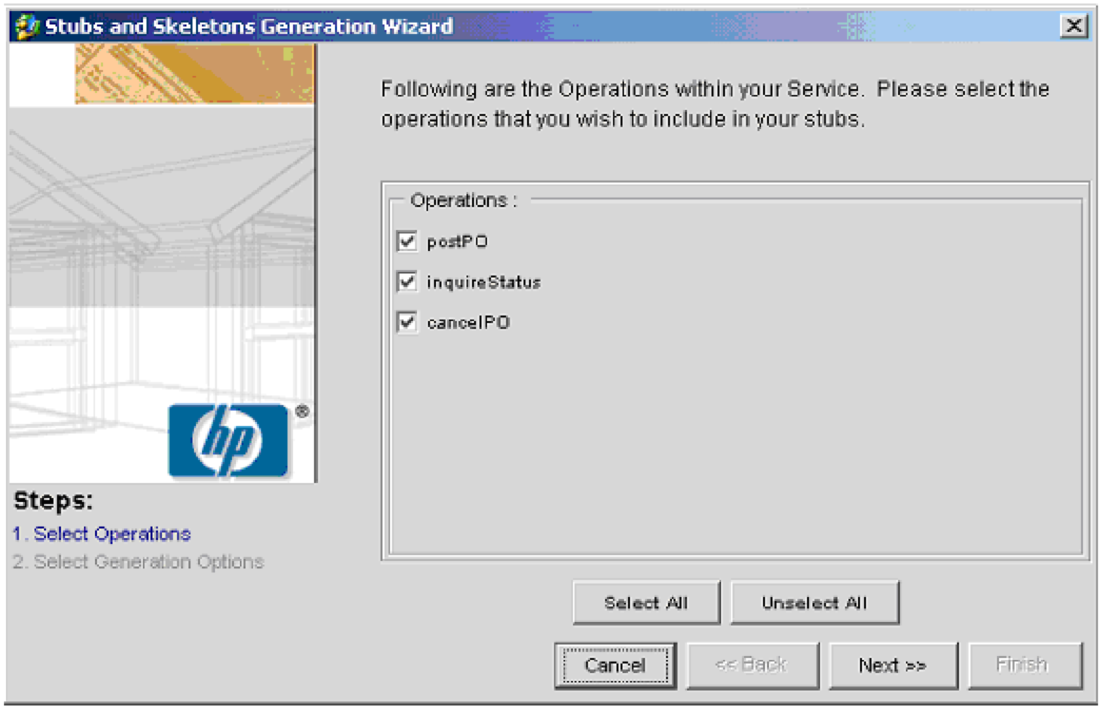
Figure 24: Modify Model Properties

Java Proxy/Skeleton Generation Wizard

To run the Java proxy/skeleton generation wizard, select a Service Binding and choose the **Object->Generate Implementation…** menu item. This command is also available on the Service Binding popup menu. The wizard will assist you in the process of creating client proxy and/or server skeleton code for the service being modeled. A client proxy is a class that implements the interface of the web service by transparently making web service calls to a selected end-point. A server skeleton is a framework implementation of a service that only requires the addition of business logic to form a complete service implementation. The following paragraphs describe the steps involved in the wizard process:

Step 1: Select the Operations to be Included in the Implementation

By default, all operations of the binding are selected. If desired, you can deselect operations that are not of interest. Click the "Next" button to move to the next and final pane.



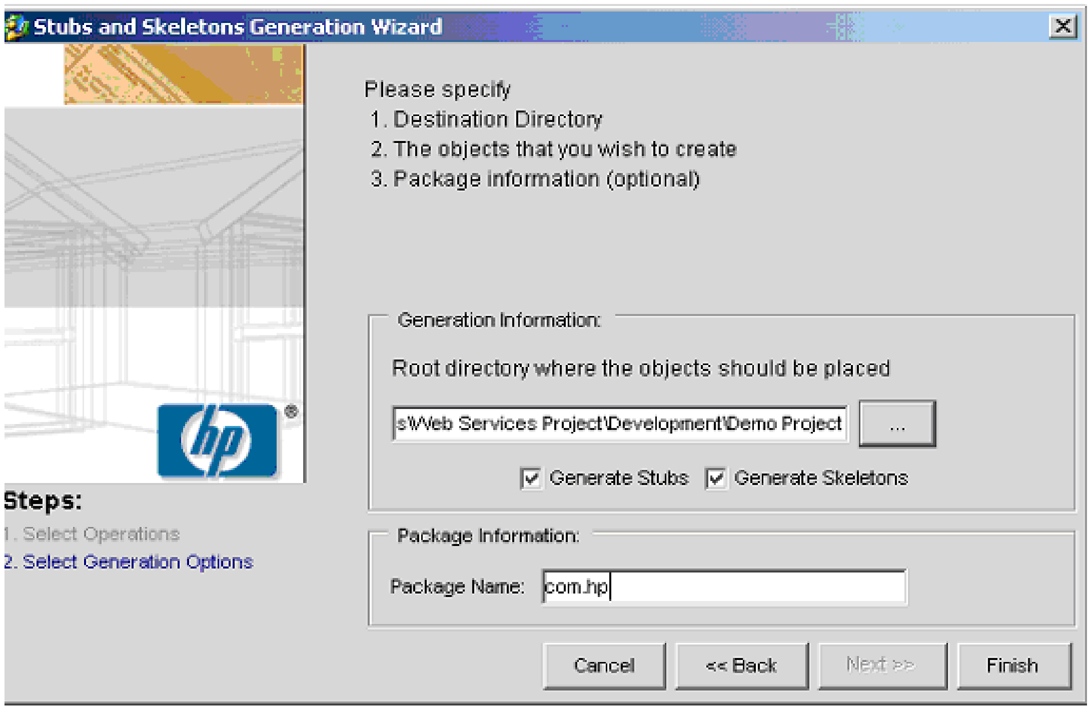
Following are the Operations within your Service. Please select the operations that you wish to include in your stubs.

- [✓] postPO
- [✓] inquireStatus
- [✓] cancelPO

Figure 25: Select the Operations to be Included in the Implementation

Step 2: Select Implementation Generation Options

Specify the directory and package name for the generated code. Use the check boxes to select whether client proxy stubs, server skeletons, or both should be generated. Finally click "Finish" to cause the wizard to invoke the code generator.



Please specify
1. Destination Directory
2. The objects that you wish to create
3. Package information (optional)

Root directory where the objects should be placed

s\Web Services Project\Development\Demo Project

[✓] Generate Stubs [✓] Generate Skeletons

Package Name: com.hp

Figure 26: Select Implementation Generation Options

Concepts and Terminology

This section gives a brief description of the key terms and concepts that are integral to an understanding of Web Services and the hp Service Composer. This section is not meant to be a tutorial of Web Service technology. Please refer to applicable white papers such as those published on hp's DSPP site for more details and tutorials.

Extensible Markup Language (XML)

XML is a widely used standard for defining markup languages that can be used to express application-specific data. It defines a simple and regular grammar that facilitates the communication, processing, and storage of application data. XML uses textual encoding (usually UTF-8) which allows it to be used with text-based transports like HTTP and simplifies application debugging. The structure of a particular class of XML documents is generally described through the XML Schema Language using a special XML dialect called XSD.

The following are some of the key terms that should be understood to take advantage of XML and XSD. Note that the following descriptions, while not technically precise, do convey the general meaning of the terms.

Element

An element is the basic unit for organizing content using XML. Elements can contain attributes, other elements, character data, or some combination of the three to express content. Elements can also be "empty", containing no information beyond the mere presence or absence of the element itself. In an XML document, elements are delimited by start and end tags containing the name of the element. For example, the following element has a name of "Address" and contains one sub-element named "Street" that contains a string:

```
<Address>
              <Street>Main Street</Street>
</Address>
```

Attribute

An attribute is a name-value pair within an element. Attributes can only "hold" simple atomic values such as strings, integers, etc, although in fact attribute values are always expressed as strings. The following element has one attribute called **Color** which contains a string value:

```
<Dog color="blue">…
```

Namespace

To prevent collisions between element names in one document with identical names in another, elements within XML documents can be associated with namespaces. The assignment of a namespace to a name is done by mapping “prefixes” to namespaces, and then using those prefixes when referring to the symbol. By convention, namespaces are named using URI’s. In the following example, the elements abc:flavor and xyz:flavor are of the same type because abc and xyz map to the same namespace. However pdq:flavor is in a different namespace and thus refers to a different type of “flavor”.

```
<doc>
              xmlns:abc=”http://food”
              xmlns:xyz=”http://food”
              xmlns:pdq=”http://physics”>
              <abc:flavor>strawberry</abc:flavor>
              <xyz:flavor>chocolate</xyz:flavor>
              <pdq:flavor>charm</pdq:flavor>
</doc>
```

As suggested by this example, namespaces also support the assignment of a shared understanding of “semantics” to document elements. For example, the element “flavor” in the namespace “http://food” might always refer the flavor of a food item. In contrast, the element “flavor” in the namespace “http://physics” might refer to a class of sub-atomic particles. The structure and semantics of each element may be quite different even though the element name is the same.

Simple Type

Attributes and elements can contain simple text strings as their content. How that text string is interpreted is generally specified in an XML schema which defines the “type” of that content. The XML schema specification also defines “built-in” types such as *String* and *Integer*. User-defined types within a schema (XSD) document can extend those built-in types using a number of mechanisms such as type restriction and enumeration. These mechanisms can be used to define new simple types such as “Zipcode”, “State”, and “SSN”. Note that simple types are always *atomic*, i.e., they don’t contain subdividable parts.

Complex Type

Unlike Attributes, Elements can contain content with named parts such as attributes and other Elements. Such Elements are said to have “Complex Type”. User defined Complex Types are used to define the structure of complex documents like “Purchase Order”, “Address”, and “Receipt”.

Simple Object Access Protocol (SOAP)

SOAP is a standardized protocol for XML messaging. It defines a packaging scheme that makes explicit the separation of messaging information (i.e., header information which is processed by the SOAP messaging layer) and the application-level content to be communicated (i.e., the “body” of the message). It also describes approaches to encoding and serializing computer language constructs (i.e., procedures and data types such as strings and arrays).

However, the use of SOAP RPC and SOAP type encoding is not a prerequisite for the use of SOAP messaging. SOAP servers (i.e., SOAP 1.1 end-points) implement the SOAP messaging layer, processing SOAP headers, throwing SOAP transport errors if necessary, encoding/decoding content, and dispatching that content to back-end processing applications. Dispatch is based on entries in the SOAP server's configuration files and some server-specific criteria, such as the content root node or the access point URL. Because SOAP uses XML and a shared server architecture, it is useful for communicating across firewalls. SOAP can also be used with MIME to exchange multi-part and non-textual information (like gifs and encrypted data). It is important to understand that basic SOAP is a fairly simple protocol that doesn't dictate constructs for things like transationality, sessionality, security, etc. However, SOAP provides extension mechanisms for describing and communicating such concepts

The following are some of the key terms that should be understood to take advantage of SOAP:

SOAP Action

As part of a measure to extend the security of SOAP servers when used using the HTTP transport, SOAP defines a HTTP header extension called SOAPAction. Before dispatching a SOAP message to a back-end processor, the SOAP server is required to check if an appropriate SOAPAction for that back-end has been specified and throw an error if this is not the case. This allows system administrators to confidently configure their firewalls using this HTTP header without having to extend the firewall to understand the SOAP protocol. To support this, HPSC and WSDL allow the developer to specify a SOAPAction string to be associated with each operation within a SOAP binding. The use of SOAPAction is optional and is only enforced by a SOAP server if specified in the service definition.

SOAP Header

SOAP messages can optionally contain zero or more headers that communicate information that should be processed by the SOAP server itself. Header information can also form a context which can be accessed by SOAP applications just as HTTP headers form a context which can be accessed by servlets and CGI scripts. Headers can be marked as being mandatory or optional for correct processing of the message. If a SOAP server receives a mandatory header (marked as "must-understand") that it doesn't understand, it must return an error back to the requestor. This mechanism allows applications to ensure that they are talking to a SOAP-server that has been extended to understand transactionality, for example.

SOAP Body

Application data is communicated in the SOAP body as an XML fragment. SOAP messages that use MIME can also put information in different MIME parts, and refer to those parts from within the SOAP body.

SOAP Encoding

SOAP encoding describes a processes for serializing common programming language data types such as strings, structures, and arrays as XML fragments. The use of SOAP encoding is optional, and is contrasted by the use of a "literal" style where the "on-the-wire" format of data to be exchanged is described explicitly by a particular XML schema.

SOAP RPC

SOAP RPC is a way of explicitly encoding and serializing a remote procedure call using SOAP. Again, the use of SOAP RPC is optional depending on the capabilities and requirements of the SOAP server.

Web Service Description Language (WSDL)

WSDL is an XML language that is commonly used to describe the public interface of a Web Service. It provides a machine-readable description of the document formats, public interfaces, operations, technology bindings, and service access points that can be used to access and interact with a Web Service. Note that WSDL does not describe or expose any details of the underlying implementation of the Web Service. This allows service implementers complete flexibility in the selection of their back-end implementations, and facilitates the "plug-and-play" characteristic of Web Service based architectures.

WSDL defines Web Services using the following key constructs. Although HPSC uses a syntax neutral internal model for manipulating web service descriptions, it currently uses WSDL constructs and terms for building web service descriptions.[1]

Types

Types, generally described using XML Schema (using the XSD language), define the structure of document that will be exchanged during a Web Service interaction. These document descriptions can be used literally within a Web Service definition, or they can be mapped via some selected encoding scheme to a particular "on-the-wire" format. In general, Complex Types defined within a schema are streamed literally while simple types might be encoded using SOAP encoding. HPSC supports the creation of document type descriptions using the Schema Editor (see "The Schema Editor").

Messages

Messages allow Web Service interactions to contain multiple named "parts", where each part is of some simple or complex type defined by some type system. These parts can then be transported in different ways (e.g., different MIME parts) as specified in a particular service binding. In HPSC, messages are defined as part of the signature of an operation within a Port Type. Operation signatures are manipulated using the Operation Editor (see "The Operation Editor"[2]

1 It is important to note, however, that HPSC does not attempt to precisely follow WSDL structure in its user interface. HPSC will depart from WSDL when necessary to improve clarity, generality, and user-interface integrity. It is the goal of HPSC, however, to provide complete support for importing and manipulating WSDL files, even those not generated by HPSC.

2 WSDL permits message definitions to be shared among multiple operations. HPSC does not directly expose the Message construct and consequently does not explicitly support Message definition sharing. In this release, each operation will reference its own unique Message definition. HPSC does, however, support the importing of WSDL's that use this construct.

Port Types

Port Types define the abstract interfaces that a concrete Web Service may implement. Such an interface consists of zero or more **Operations** that represent potential interactions between end-points (i.e., an exchange of one or more documents between a service client and a service provider). It is important to note that a Port Type Operation does not necessarily have to map one-to-one to a particular back-end method. Port Types are abstract and do not expose in any way the back-end implementation details. (Because of this confusion, it might have been better if a term like "Interaction" was used instead within the WSDL specification.)

Bindings

Bindings map abstract interfaces (a.k.a Port Types) to concrete serialization and transport mechanisms that can be used to access a real web service that implements that interface. Bindings specify how message parts (or equivalently, operation parameters) are marshaled (i.e., serialized or encoded for on-the-wire communication), and how the different parts are packaged for transport. Bindings represent an agreement between end-points on how interaction semantics (data types, relational data, sessionality, and Quality of Service constraints[3]) are to be preserved and communicated. HPSC currently supports the two standard binding constructs described by WSDL via the the HTTP Binding Editor (see "The HTTP Binding Editor") and the SOAP Binding Editor (see "The SOAP Binding Editor").

HTTP bindings are always accessed using the HTTP transport using either a URL encoding scheme (via HTTP-GET) or standard HTTP-POST form encoding. SOAP bindings can be accessed via a number of different transports, although support is currently only provided for HTTP-POST and SMTP.

Services

Services are collections of related **Ports** that represent concrete end-points (e.g., a URL) for accessing the functionality implemented by a Web Service. Ports represent concrete implementations of a particular Binding, and consequently expect that service clients that interact with the Port will conform to the encoding and packaging constraints described by that binding.

Universal Description, Discovery, and Integration (UDDI)

In order for a prospective clients to find a service, the service must be registered or advertised in some well know location. UDDI is a standard SOAP-based interface for accessing registries that can contain business and technical service information including references to WSDL documents. When a service is deployed and an access point (URL) assigned, a developer or administrator can register the service within a UDDI registry so that the access point and the associated service information is available to client applications. The hp Registry Composer can be used to view and manipulate UDDI registries such as the global federated registry being run by Hewlett-Packard, IBM, and others.

3 Quality of Service issues such as transactionality security, non-repudiation, and transport reliability, as well as issues of sessionality are not currently supported by standard WSDL extensions except by a generic SOAP header capability. However, these issues when addressed will form part of the binding description of a service interface.