HP Service Manager

For the Supported Windows® and UNIX® operating systems

Software Version: 9.32

Web Services Guide

Document Release Date: August 2013

Software Release Date: August 2013

Legal Notices

Warranty

The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notice

Trademark Notices

Adobe™ is a trademark of Adobe Systems Incorporated.

Java is a registered trademark of Oracle and/or its affiliates.

Microsoft® and Windows® are U.S. registered trademarks of Microsoft Corporation.

Oracle® is a registered US trademark of Oracle Corporation, Redwood City, California.

UNIX® is a registered trademark of The Open Group.

For a complete list of open source and third party acknowledgements, visit the HP Software Support Online web site and search for the product manual called *HP Service Manager Open Source and Third Party License Agreements*.

Documentation Updates

The title page of this document contains the following identifying information:

- Software Version number, which indicates the software version.
- Document Release Date, which changes each time the document is updated.
- Software Release Date, which indicates the release date of this version of the software.

To check for recent updates or to verify that you are using the most recent edition of a document, go to:

http://h20230.www2.hp.com/selfsolve/manuals

This site requires that you register for an HP Passport and log on. To register for an HP Passport ID, go to:

http://h20229.www2.hp.com/passport-registration.html

Or click the **New users - please register** link on the HP Passport log on page.

You will also receive updated or new editions if you subscribe to the appropriate product support service. Contact your HP sales representative for details.

The following table indicates changes made to this document since the first released edition.

Document Changes

Document Date	Changes
July 2011	Initial edition.
April 2012	Updated "fastInfoset" instances in "Interpreting the http.log" on page 158 to "FastInfoset".
July 2012	Updated topic "The response" on page 85.
July 2013	Updated topics for RESTful API.

Support

Visit the HP Software Support Online web site at:

http://www.hp.com/go/hpsoftwaresupport

This web site provides contact information and details about the products, services, and support that HP Software offers.

HP Software online support provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued support customer, you can benefit by using the support web site to:

- Search for knowledge documents of interest
- Submit and track support cases and enhancement requests
- Download software patches
- Manage support contracts
- Look up HP support contacts
- Review information about available services
- Enter into discussions with other software customers
- Research and register for software training

Most of the support areas require that you register as an HP Passport user and sign in. Many also require a support contract. To register for an HP Passport ID, go to:

http://h20229.www2.hp.com/passport-registration.html

To find more information about access levels, go to:

http://h20230.www2.hp.com/new_access_levels.jsp

Contents

Chapter 1

Service Manager Web Services

Service Manager Web Services provide the ability to communicate and integrate with applications in an open and efficient manner. Web Services provide the ability to use a third-party application inside Service Manager, manipulate Service Manager data inside your custom application, or transfer data among separate Service Manager systems.

Purpose

This document provides guidance for users who wish to publish or consume Web Services using Service Manager. It includes examples that can be used as templates.

Web Services and their clients can be written in any programming language and for any platform. Service Manager Web Services ships with examples using both the Java™ and Visual C++® programming languages.

What is a Web Service?

The formal definition (according to www.w3c.org) is that a Web service is a software application identified by a Uniform Resource Identifier (URI), whose interfaces and binding are capable of being defined, described, and discovered by XML artifacts and supports direct interactions with other software applications using XML-based messages via Internet-based protocols.

A Web service is a software system designed to support interoperable application to application interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with Web Services in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

There are two major classes of Web Services:

- REST-compliant Web Services: The primary purpose of the service is to manipulate XML representations of Web resources using a uniform set of "stateless" operations.
- Arbitrary Web Services: The service may expose an arbitrary set of operations.

Both these Web Services use URIs to identify resources and use Web protocols (such as HTTP and SOAP) and XML data formats for messaging.

Understanding the Service Manager Web Services

Every Web service published by Service Manager is a document-literal service. The documents which are used for the requests and replies are derived from the dbdict definition of a single Service Manager file and published via the fields section of the extaccess record.

Each field in the Service Manager data model must be understood in the context of the business logic for the application that defines the data. Before approaching any Web Services consumption project, it is important to understand the data model that is implemented within the Service Manager instance you are targeting. Because Service Manager allows you to add new fields, change the validation of fields or make fields mandatory, every Service Manager implementation will have a slightly different data model and business logic, and each difference has to be reflected in the published Web Service to ensure successful processing.

Web service definitions are maintained in the Web Service Configuration Utility. In this utility you can see how file names such as probsummary are aliased to Incident, how fields within files can be exposed for purposes of Web Services; and how they are aliased to more appropriate names. Finally, the Web Service Configuration Utility is where XML schema data types such as dateTime can be applied to individual fields. The default type is string, but Service Manager fields can be mapped to various XML schema types if needed.

Web Services basics

The basic Web Services architecture includes the following:

- Publishing - Publishing a Web Service means enabling a Web Service user (consumer) to locate the service description and instructing the consumer how they should interact with the Web Service.
- Consuming (client) - A client is software that is able to use (consume) a Web Service. A service consumer issues one or more queries to the directory to locate a service and determine how to communicate with that service.
- Service - A collection of EndPoints that provides the servicing of the consumers request.
- EndPoint (port) - An EndPoint indicates a specific location for accessing a Web Service using a specific protocol and data format.

Adding or changing Web Services

1. Modify an existing extaccess record if the Web Service connecting through it needs to get additional information or add a new extaccess record if you want to expose a table to a new Web Service and do not want to interfere with existing Web Service applications.

   Note: Writing expressions in extaccess: The extaccess tool uses the same file variables as the Document Engine. For example, a file variable that holds the current record is $L.file, and the copy of the record before modifications is $L.file.save.

2. For SOAP, rebuild or build the Service Object by re-running WSDL2JS if Service Manager is the consumer.

3. Modify the calling applications if actions, field names, or object name have changed, and a calling application refers to them.

Introduction to Web Services in Service Manager

The published out-of-box ITIL®-based processes for Web Services are:

- Service Desk
- Incident Management
- Problem Management
- Knowledge Management
- Configuration Management
- Change Management
- Service Catalog
- Service Level Management

Note: Table-based Web services are still available in Service Manager when needed. Each Web Service application can have a "different view" of the defined services, but the underneath logical flow is still controlled by same Service Manager applications. To avoid validation failure, make sure all the required fields are always exposed.

To publish a Service Manager Web Service, you create one `extaccess` record per table that you want to publish in that service. Each Web Service application can have a different view of the defined services but the underneath logical flow is still controlled by same Service Manager applications. To avoid validation failure, make sure all the required fields are always exposed on all `extaccess` records for the table. To add or modify an extaccess record:

Click **Tailoring > Web Services > Web Service Configuration**.

For SOAP, you may need the `allowwsdlretrieval` parameter in the `sm.ini` to be able to view Service Manager WSDL.

Important: Changes to a specific `extaccess` record affect any client that is currently consuming the Web Service created by that record. If you modify this configuration, make sure to test all other applications that consume the same Web Service and address possible issues immediately. To avoid issues stemming from different applications using the same Web Service, create a unique `extaccess` record for each Web Service application, so that each application has a unique Web Service to consume. A single table can be represented in multiple `extaccess` records.

Web Services and Service Manager

A Web Service enables one application to access the functionality of another application using SOAP operations (XML-based transactions) or RESTful operations, regardless of differences in their operating system platform, application language, or tool set. HP Service Manager supports two types of Web Services features:

- Connecting to and consuming external Web Services
- Publishing Service Manager fields and methods as Web Services

For SOAP, the Service Manager server now offers out-of-box functionality to connect to and consume external Web Services. When you connect to an external Web service, Service Manager retrieves the Web Service Description Language (WSDL) for the service. You can then write custom JavaScript to use JavaScript functions generated by Web Services and send and receive messages to the remote Web Services. For example, you might query external Web Services to:

- Validate an email address or a phone number when updating a contact record.
- Automatically fill in the time zone of a contact in a Service Desk interaction based on the location given.
- Automatically perform a search for solutions using the brief description of the Service Desk interaction.

The out-of-box Service Manager includes a bundle of published tables, fields, and display actions collectively known as the Service Manager Web Services. The Service Manager Web Services includes Web Services for all the applications and uses an ITIL-compliant naming convention to refer to the Web Service object. The use of ITIL-compliant service and object names allows Web Services developers to create custom Web Services without needing to be familiar with the Service Manager database layer. To consume Service Manager tables, fields, and display actions, you must grant an operator the SOAP API or RESTful API capability word.

You can use the Service Manager Web Services to integrate applications and automate transactions. For example, you might want to publish a Web Service that enables another application or process to:

- Automatically open, update, escalate, resolve, or close Service Manager incidents.
- Automatically add or update a configuration item.

In addition to the tables, fields, and display actions available though the Service Manager Web Services, you can customize the Web Services available from Service Manager by adding, changing, or removing your own tables, fields, and display actions. When you customize the Web Services, Service Manager creates a new version of the Service Manager Web Services. Afterwards, any custom Web Services clients you create access this new version of the Web Service.

Web Services naming conventions for SOAP

The request and response names use the literal strings of the Action Name and Object Name defined in the extaccess record. The name of the Request and Response methods within Service Manager's Web Services are constructed by combining the Action Name with the Object Name and Request or Response.

Note: These names are case sensitive.

For example, the method to add a new incident to the system is:

Action Name	Object Name	Request	Response
Create	Incident	CreateIncidentRequest	CreateIncidentRespons e

If your Object Name for the Incident object starts with a lower case “i” (incident) the request is CreateincidentRequest and the response is CreateincidentResponse.

Web Services security considerations

The Service Manager server requires that each Web Service request provide a valid operator name and password combination. These must be supplied in a standard HTTP Basic Authorization header. The Web Service toolkits universally support this authentication mechanism. Use SSL if you are concerned about the possibility of someone using a network monitoring tool to discover passwords. Basic Authorization by itself does not encrypt the password; it simply encodes it using Base 64.

Note: Only ASCII operator names are supported in Service Manager Web Service integrations. When Service Manager is handling an incoming Web Service request, the authorization string is decoded by BASE64Decoder. Service Manager uses the decoded string value to construct a UTF-8 string that is used in the RTE. However, the authorization string is in the header and Service Manager does not know the charset or encoding of the underling string value, which is BASE64 encoded. Therefore, if the underlying string value is not UTF-8, Web Service clients will fail to connect to Service Manager. In Service Manager, when fetching an operator from the database, no matter what collation the database uses, the operator finally will get a UTF-8 operator value. However, even if users put the same value in the authorization header, the operator name may differ because of the charset/encoding issue.

In addition to having a valid login, the operator must have the SOAP API or RESTful API capability word to access the Web Services. If the Web Service request does not contain valid authorization information, then the server sends a response message containing “`401 (Unauthorized).`” If the request is valid, then the server sends a response message containing the results of your Web Services operation. The response message contains only the information the operator is allowed to see. The security settings of the user's profile, Mandanten security settings, and conditions defined in the Document Engine are maintained by all Web Services.

Valid URLs for Service Manager

The Service Manager publishes two different URLs for SOAP:

`http://<server>:<port>/SM/7`

This URL contains similar functionality as `sc62server/PWS`, except that it uses MTOM attachments.

`http://<server>:<port>/sc62server/PWS`

This URL provides complete functionality and despite the name `sc62server`, it is a fully implemented Service Manager7 Web Services interface using MIME attachments.

The Service Manager also publishes one URLs for REST:

```
http://<server>:<port>/SM/9/rest
```

Service Manager Web Services URLs

HP Service Manager support the Web Services at both URLs for SOAP API. If you already use the SC62 server , continue to use it. If you are starting to create a new Web service, use the SM/7 server. You can continue to use the methods, which are still applicable other than the following.

- Any new objects added to Service Manager 9.32, such as the new required fields in Incident Management, will not be available to existing Web Services.
- If you have an SOA broker application, BPEL orchestration engine, or Web Services middleware application cofigured between the deployed SOAP client application and ServiceCenter or Service Manager application. If so, the orchestration scenario or middleware can be modified to work as a mediator between the old and the new version of the IncidentManagement WSDL.
- MIME – If you use the legacy Web Services URL, then the server uses MIME to encode attachments.
- MTOM/XOP – If you use the Service Manager Web Services URL, then the server uses MTOM/XOP to encode attachments.

HP Service Manager also supports one URL for RESTful API.

Configure the Web Service field definitions

Use the Web Service Configuration Utility to define the fields that will be passed from HP Service Manager to the Web Service. The Service Manager fields are taken directly from the database dictionary.

Field	Description
Service Name	The name of the Web Service you want to use to publish theService Manager table. You can reuse the same Web Service name to publish multiple tables. Since this name becomes part of a URL, the name must consist of alphanumeric characters valid for URLs. The name cannot consist of URL reserved characters such as spaces, slashes, or colons.
Released	You should consider any web service with the Released option selected as the supported version of the Web Service in Service Manager. While it is possible to clear the Released option and edit or delete the Web Service, HP recommends that you assign the service a different name and work on that copy of the web service instead. When the Released option is selected, the external access definition remains read-only.
Name	The name of the Service Manager table that will be published as a Web Service.
Deprecated	Web Services marked as deprecated are not supported.

Field	Description
Object Name	The name you want to use to identify the Service Manager table in the Web service. Since this name becomes part of the WSDL, the name must consist of alphanumeric characters valid for XML. The name cannot consist of XML reserved characters such as brackets (<) and (>), colons (:), or quotation marks (").

Allowed Actions tab field definitions

Use this tab to enter the HP Service Manager Document Engine display actions you want to globally enable for this table.

Field	Description
Allowed Actions	Click to see the list of allowable display actions for the Service Manager table you have selected for the Web Service.
Action Names	The name used to identify the display action in the Web service as an operation. Since this name becomes part of the WSDL, the name must consist of alphanumeric characters valid for XML. The name cannot consist of XML reserved characters such as brackets (<) and (>), colons (:), or quotation marks (").
Action Type	The type for each of the Document Engine display actions that are defined for this table. Click the drop-down icon to see a list of valid type values. • Create only actions will only create new records. • Update only actions will only update existing records. • Merge actions will update the record if it exists and create it if it does not exist. • Application Pass Through actions will perform custom actions defined in External Access Actions.
Custom Action To...	Create a custom action for the Service Manager table you have selected for the Web Service.

Expressions tab field definitions

Use this tab to enter system language expressions that run before the display action that runs as part of the Web Service.

Field	Description
Expressions	Call a custom action created in External Access Actions.

Fields tab definitions

Use this tab to set the fields, captions, and field types.

Field	Description
Field	The HP Service Manager field name that is published by the Web Services Configuration Utility.
Caption	The name that Service Manager displays for the associated Field in the Web Service.
Type	The data type that the Web Services API will convert field data to for Web Services access.

RESTful tab field definitions

Use this tab to enter the RESTful API related configurations.

Field	Description
RESTful Enabled	If it is selected, RESTful API is available for this service. If it is not selected, RESTful API is unavailable for this service.
Attachment Enabled	If it is selected, attachment is supported by RESTFful API for this service. If it is not selected, attachment is not supported by RESTful API for this service.
Resource Collection Name	This is the name of the Resource Collection. For example, you may specify the group of incidents from the probsummary table as "incidents".
Resource Name	This is the name of the individual Resource. For example, you may specify that any individual incident from the probsummary table be referred to as an "Incident".
Unique Keys	This field specifies one or more fields that will function as a unique identifier for a Resource from the Resource Collection. For example, {ID} in single resource query URI, http://<server>:<port>/SM/9/rest/incidents/{ID}
Max Records Returned in Query	This is the max number of records returned in every single query when the number of records is huge. By default, the value is 1000.

Field	Description
Query Authorization	This is the query privilege for this service.
Resource Collection Action	This field represents the default action for resource collection.
Resource Actions	This field specifies the action to take when an individual resource is part of a POST, PUT, DELETE command. These actions are specified in the **Allowed Actions** tab of the **External Access Definition**.

Chapter 2

Publishing Service Manager data using WS API

To publish Service Manager data via Web Services, use the Web Service Configuration tool to expose files and methods to add, update, or delete Service Manager records. The consumer of this data can be a custom C# or Java program or an interface program such as Connect-It as well as another Service Manager system.

To expose a set of Service Manager tables as a Web Service, click **Tailoring > Web Services > External Access Actions** and create or update the related extaccess record for each of the tables.

Things to consider prior to publishing data

Before publishing Service Manager data via a Web Service, there are several things to consider. When investigated thoroughly, each of the following items will serve to improve the organization and performance of the Web Services.

Publishing Service Manager applications as Web Services

You can publish HP Service Manager applications as Web Services and create new integration points between the Service Manager server and external applications.

You can customize the Web Services that Service Manager publishes by adding or removing tables, fields, and display options, from the list of objects available to the Web Services. In addition, you can create alias names for each of these options that only appear in the Web Services but HP recommends that you do not do this. You can also specify the XML schema data type you want the Service Manager server to use when publishing data to a SOAP Web service.

For your custom Web Services client to access Service Manager Web Services, it must present a valid operator record name and password with each request. Furthermore, the operator must have the SOAP API or RESTful API capability word as part of his or her security profile.

When to use Web Services

Web Services enable user-driven integrations with any application that supports Web Services.

- Web Services can be used for any table and external applications that support the technology.
- Web Services can be used to view data from an external source or copy data from one system to another.

Can I use the out-of-box Web Services?

The ITIL-standard Web Services provided with Service Manager should be used whenever possible. They have been tested extensively and are well documented, which makes them easier to use. If you are interested in using one of these Web Services, HP recommends that do not modify the out-of-box extaccess records. Instead, always create your own copy if you need to add actions or fields. If the changes are unique, create a copy of the extaccess record(s) involved first and name the service differently; for example, IncidentManagementForPortal rather than just IncidentManagement, and make your changes against the new set of extaccess records.

What items do I need to expose?

Only expose required fields and fields that are necessary for the actions exposed. Expose only those actions that are required for the consumer to perform their duty. All actions that are exposed need to be able to run in background without user interaction.

Though an entire table and even an entire system can be exposed via Web Services, doing so would affect performance and confuse users. Only the data that is needed by a client should be exposed. This prevents excess traffic and decreases the amount of storage that your client may need to use.

Publish a Document Engine display action in the Web Services API

You must have the SysAdmin capability words to use this procedure.

The Service Manager Web Services API allows you to publish any Document Engine display action as part of a Web Service.

1. Log on to Service Manager as a System Administrator.
2. Click **Tailoring > Web Services > Web Service Configuration**. Service Manager displays the External Access Definition form.
3. In the Name field, type the name of the Service Manager table or join file whose display actions you want to publish.
4. Click **Search**. The External Access Definition record for the table opens.
5. Double-click the applicable object name entry. The External Access Definition form opens.
6. Click an empty cell from the Allowed Actions array and select the Document Engine display action you want to publish from the list.

   Note: If a join file is chosen, the allowed actions for the join file are controlled by the initialization expressions in the `ext.init` Process record (click **Tailoring** > **Document Engine** > **Processes**).

Note: You must first have created the necessary Document Engine records, states, objects, and display actions, for a custom display action to appear in this list.

7. In the Action Names field next to the allowed action, type the name you want Service Manager to display for the action in the Web Services API.

   Note: The name you type for this field becomes the alias name for the display action and becomes part of the Web Service WSDL. For example, if you type Create for the add action of the Incident object, then the WSDL operation becomes CreateIncident and the WSDL messages are CreateIncidentRequest and CreateIncidentResponse.

   Caution: Since this name becomes part of the WSDL, the name must consist of alphanumeric characters valid for XML. The name cannot include XML-reserved characters such as brackets (< & >), colons (:), or quotation marks (" & ").

8. In the Action Type field, select the conditions where this action will be valid.
   - To limit the action to new records, select the Create only type.
   - To limit the action to existing records, select the Update only type.
   - To make the action available for both new and existing records, select the Merge type.

9. Click **Save**.

Publish a Service Manager field in the Web Services API

You must have the SysAdmin capability words to use this procedure.

1. Log in to Service Manager as a System Administrator.
2. Click **Tailoring > Web Services > Web Service Configuration**. Service Manager displays the External Access Definition form.
3. In the Service Name field, select the name of the Service Manager table or join file in which you want to rename fields.

   Note: If a join file is chosen, the Fields tab lists all of the fields for all of the files in the join file.

4. Click **Search**. The web services record for that table opens.
5. Click the Fields tab.
6. In the Field field, type the name of field for which you want to create an alias.

   Note: To specify a compound field type such as an array of structure or an array of characters, you must use a special syntax.

7. In the Caption field, type the name (alias) you want the field to have in the Web Services API.

Caution: Since this name becomes part of the WSDL the name must consist of alphanumeric characters valid for XML. The name cannot consist of XML reserved characters such as brackets (< & >), colons (:), or quotation marks (“ & ”).

8. In the Type column, select a data type override, if any, you want the field to have in the Web Services API.

   Note: If you leave the Type field blank, Service Manager uses the default mapping to determine the data type. Any data type you select overrides the default mapping. The default is StringType.

9. Click **Save**.

What data types should I use for SOAP?

HP Service Manager has a more lenient data typing policy than the XML schema data typing policy used for Web Services. Certain field types in Service Manager can correspond to multiple data types in the XML schema data type policy. For example, the Service Manager data type decimal could be a decimal, a floating number, or an integer in the XML schema data type policy.

In addition, the actual formatting of data varies between Service Manager and XML schema data types. This is especially true of Service Manager date/time fields that use a different order than XML schema dates. Because some Web Services may require changes to field data format, you can now define the XML Schema data type you want Service Manager to convert the field's data to when you publish the field as part of a web service.

For outbound data, the Service Manager server automatically converts Service Manager data to the format you select in the data policy record for the Service Manager field. For inbound data, the Service Manager server automatically converts the XML schema data to the Service Manager field's listed data type format.

The services, objects, and fields published in the Service Manager Web Services API already have the proper XML schema data mappings listed in the Web Services definition (extaccess record). If the extaccess record does not list a data type mapping, then the Web Services API treats the field data as a string.

The following table lists the available SOAP API data types and their Service Manager equivalents.

SOAP API Data Type	Service Manager Data Type
Base64Type	used for binary data
BooleanType	Boolean
ByteType	Decimal
DateTimeType	Date/Time
DateType	Date/Time

SOAP API Data Type	Service Manager Data Type
TimeType	Date/Time
DurationType	Date/Time
DecimalType	Decimal
DoubleType	Decimal
IntType	Decimal
LongType	Decimal
ShortType	Decimal
FloatType	Decimal
StringType	Text

Caution: Always map Service Manager date/time fields to the XML schema dateTime or to one of the related XML schema date or time types. Otherwise these fields will cause errors when you consume the service.

Important: When integrating with Service Manager, array data should be broken into multi elements by separator "\r." This is because Service Manager uses "\r" as the separator between array elements. When a string that contains "\r" is retrieved from the Service Manager system, it is decoded as an array with multi elements separated by "\r". For this reason, when integrating other applications (for example, UCMDB) with Service Manager through web services, array data should be broken into multi elements by separator "\r" before the data is encoded and sent to the Service Manage system. For example, if an array contains elements "aabb" and "ccdd", it should be sent to Service Manager as the following:

```
<ns:Comments type=\"Array\">
<ns:Comments mandatory=\"\" readonly=\"\">aabb</ns:Comments>
<ns:Comments mandatory=\"\" readonly=\"\">ccdd</ns:Comments>
</ns:Comments>
```

You can define the data type you want Service Manager to convert field data to when publishing it as a Web Service. These data types are consistent with XML schema data types.

1. Click **Tailoring > Database Manager**.
2. In the Table field, type **extaccess** and click **Search**. The External Access Definition record opens.
3. In the Name field, select the name of the Service Manager table whose exposed field you want to define datatypes for.
4. Click **Search**. The External Access Definition record for the table opens.

5. On the Fields tab, find the field that you want to define the data type for.
6. In the Type column for that field, either type the data type or select a data type from the predefined list in the drop-down list.

   Note: The data type you select for this field becomes an XML schema data type in the web services WSDL.

   Important: You must also specify a Field name in API value when you set a data type value. Data type validation depends upon the existence of an alias name.
7. Click **Save**.

What methods do I need?

By default, any operation that is a part of the Document Engine for a table can be made available in the table's Web service. If you need additional methods, add them to the Document Engine first so that Service Manager has a process to follow when performing them. If you have methods in the Document Engine that you do not want exposed, delete them from the allowed actions array in the extaccess table.

Note: All actions performed from Web Services have to run without user interaction in Service Manager. It is not possible to prompt the user for more information when that user is a Web Services consumer.

Managing records with Web Services requests

An implementer can send a Web Services request to HP Service Manager that will create a new record, update an existing record, or merge two records. These actions are defined by selecting a value in the Action Type field on the Allowed Actions tab of the extaccess record. The following is a description of the expected behavior for each of the values in the drop-down list.

Create only

The server uses Create Semantics to initialize the file variable, fill it with the data from the Web Services request, and pass it to the se.external.action RAD application.

Update only

The server uses Update Semantics to select the matching record before calling the se.external.action RAD application. The server returns an error if it does not find a matching record.

Merge

The server attempts to select the record. If it finds the record, it changes the action to Update and calls the se.external.action RAD application. If the server fails to find the record, it changes the action to Create and calls the se.external.action RAD application. If either the Update or Create action is missing, the se.external.action returns a 70 – invalid action error message.

If there is no value specified in the Action Type field, the server uses Update Semantics. The only exception is when the Action Name specified is Create, in which case the server uses Create Semantics.

Are there any security considerations?

After you have exposed data via Web Services, any client consuming the Web Service you are publishing has access to that data. If there are certain fields that you want to restrict from specific clients, create a different Web Service with those fields removed and have these clients consume that data.

What are released Web Services?

The Web Services delivered out-of-box with Service Manager are read-only and marked with the released option in the external access definition form. You should consider any Web Service with the released option selected as the supported version of the Web Service in Service Manager. While it is possible to clear the released option and edit or delete the Web Service, HP recommends that you instead work on a copy of the Web Service that you give it a different name. While the released option is selected the external access definition remains read-only.

Enable SSL encryption for published Web Services

If you want external Web Services clients to use an SSL connection with the Service Manager server, you must provide them with the CA certificate for the Service Manager server. If you purchased a server certificate, copy the CA certificate from the CA certificate keystore provided with your purchased certificate. If you generated your own server certificate by using a self-signed private CA certificate, copy the CA certificate from your private CA certificate keystore instead.

Note: HP recommends you do not use the Service Manager sample server CA certificate because the sample certificate uses a common name (CN) for the server which will not match your actual server name. The best practice is to purchase or create a valid certificate for the Service Manager server in order to establish an SSL-encrypted connection with external web service clients.

1. Copy the keystore that contains the CA certificate that signed your server's certificate and send it to the systems running the external Web Services clients. Out-of-box, Service Manager uses a sample CA certificates keystore as part of the Web tier.

   Note: HP recommends using a CA certificate that you created or purchased instead of the default Service Manager CA certificate.

2. Import the CA certificate of the Service Manager system into the CA certificate keystore of the external Web Services client. You may use a tool like keytool to import the Service Manager CA certificate.

3. Configure the external Web Services client to use the updated CA certificate keystore. Follow the instructions for your Web Services client to set the path to the CA certificate keystore.

4. Update the endpoint URL that the external Web Services client uses to include the HTTPS protocol.

   For example, `https://myserver.mydomain.com:13443/SM/7/ws` for SOAP and `https://myserver.mydomain.com:13443/SM/9/rest` for RESTful.

   Follow the instructions for your Web Service client to update the endpoint URL.

   Note: The endpoint URL must use the Service Manager server's common name (CN) as defined in the server certificate. For example, if the server certificate uses the name myserver.mydomain.com, then the endpoint URL must also use the name myserver.mydomain.com.

   Note: If you want external Web Services clients to download the Service Manager Web Services WSDL, point them to a URL using the following format:
   `https://myserver.mydomain.com:13443/SM/7/<Service Name>.wsdl`

Example: Publishing request processes for integration

In this example, we prepare the data from the `ocmq` file to integrate to Project and Portfolio Management (PPM) via Web Services. We will assume that as part of a project, a new employee needs to be hired and the hiring process (approvals and workflow) will be done within the Service Manager Request Management Module.

Since Request Management is not published as a Web Service, we will need to start by creating some customized Display options and Processes. Once these work in the Windows client, we will include them in the newly-created extaccess record.

Since the goal is to publish just the possibility to start the new hire process, a lot of the required information will be hard-coded in the request creation, to minimize overhead.

Create the display option

To create the display option:

1. Log in to Service Manageras a System Administrator.

2. Click `Tailoring > Database Manager` and open the **displayoption** table.

3. Search for an available display option for the rmq.main.display display screen in the range from 200 – 2000.

4. Create a new display option with the following values:

Field	Value
Screen ID	rmq.main.display
Action	CreateNewHire
Unique ID	rmq.main.display_CreateNewHire
GUI option	500
Text Option	500
Default Label	Create New Hire Request
Bank	3
Condition	true

5. Add the new record.

Set up the Request Management category

For the background processing to work correctly, the category has to have `Assign Number Before Commit` selected, so the flag is set to true.

Note: For this example, set the hr category flag to true.

Field	Value
Name	hr
Description	Human Resources
Availability	true
Assign Number Before Commit?	Select this option
Phases - Phase Name	**Condition**
Initial Quote	true
Quote Approval	true
Working	true
Customer follow-up	true

Create the new Process

The new `rmq.open.newhire` Process will first prepare the `ocmcowork` record, and then open the new quote with a single line item for the New Employee bundle.

The new Process will need the following Initial JavaScript:

```
system.vars.$L_work=new SCFile("ocmcowork");
```

On the RAD tab, enter the following information:

Expressions before 1st RAD:

```
$L.part.no={100};$L.quantities={1};$L.item.quantity=1
```

1st RAD:

```
-      svcCat.build.work.file.sub      -       Condition: true

-      names           -        $L.part.no
-      record          -        $L.work
-      numbers         -        $L.quantities
-      number1         -        $L.item.quantity
```

Expressions before 2nd RAD:

```
if (filename($L.file)="ocmq") then ($fileq=$L.file)
```

2nd RAD:

```
-      rmq.open        -       Condition: true
-      file            -       $L.file
-      second.file     -       $L.object
-      text            -       $L.exit
-      boolean1        -       true
-      cond.input      -       false
-      record          -       $L.work
```

Set up the State record

The rmq.view State record has to link the new display option to the new Process record. Add the following line:

```
- CreateNewHire - rmq.open.newhire - true -false
```

Update the format control record

Disable the `ocml.bld.smry` subroutine call on the ocmq format control record that runs on display.

Set up the extaccess record

In the default Service Manager system, Request Management tables are not published as a Web Service.

Note: Enter all fields that need to be exposed, and then create field captions for those fields. The field captions cannot be XML-reserved characters and cannot contain spaces. Use of mixed case is supported.

To create a new Web Service, do the following:

1. Log in to Service Manageras a System Administrator.
2. Click **Tailoring > Web Services > Web Service Configuration**.
3. To publish Request Management Quotes, create a new record with the following information:

Field	Value
Service Name	RequestManagement **Note**: Type the name of the web service that you want to use to publish this table may be comprised of multiple Service Manager tables. The name you type in this field becomes the alias name for the service and it becomes part of the web service URL. For example, when you type **RequestManagement** as the service name, then the WSDL you publish will be called RequestManagement.wsdl. The name cannot contain URL-reserved characters, such as spaces, slashes, or colons.
Name	ocmq

Field	Value
Object Name	Quote **Note**: Type the name you want to use to identify the table. This name becomes the alias name for the table, and then becomes part of the web service WSDL. For example, when you type **Quote** as the object name, then the SOAP operations for this table include Quote as part of the WSDL element, such as UpdateQuote, CreateQuote, and DeleteQuote. The name cannot consist of XML-reserved characters, such as brackets (< and >), colons (:), or quotation marks (" and '). Never use "CamelCase" (mixed case) notation in the Object name, as this creates an incorrect or missing filename when calling the web service via Service Manager. As a workaround, you can use a tool that lets you modify the XML to include the filename in the SOAP body request. However, Service Manager and some other tools do not allow modifications.
Allowed Actions	The Allowed Actions have to match the action field in the Display Option, and in the display action field in the State record. Only options that have a true condition will be available through the web service interface. Operator privileges will be checked to ensure security.
1st entry	save
2nd entry	GenNewHire
Action Names	Type the name you want to use in the Web Services application program interface (API) to identify the Document Engine display actions for this table. The name you type for this field becomes the alias name for the display action, and then becomes part of the web service WSDL. The only action that can be used to add a record to a Service Manager table is Create. Updating actions can be named to fit the action. Foe example, if you type Create for the add action of the Quote object, then the WSDL operation becomes CreateQuote and the WSDL message is CreateQuoteRequest. The name cannot consisit of XML-reserved characters, such as brackets(< and >), colons (:), or quotation marks (" and ').
1st entry	Update
2nd entry	Create
Action Type	
1st entry	Merge
2nd entry	Create only
Expressions tab	`if null(number in $L.file) then ($L.mode="add")`

4. On the Fields tab, type the following:

Field	Caption	Type
priority	Priority	StringType
requested.for	Requestor	StringType
requestor.dept	RequestingDepartment	StringType
reason	Reason	StringType
location	Location	StringType
hire.type	HireType	StringType
requested.date	StartDate	DateTimeType
requestor.fname	NewEmployeeFirstName	StringType
requestor.lname	NewEmployeeLastName	StringType
category	Category	StringType
current.phase	Phase	StringType
number	Number	StringType

5. If you want to use RESTful API, on the **RESTful** tab, type the following:

Field	Value
RESTful Enabled	Checked
Attachment Enabled	Checked if you want to use attachment later.
Resource Collection Name	quotes
Resource Name	Quote
Unique Keys	number
Max Records Returned in Query	1000
Query Authorization	true
Resource Collection Actions: POST	Create
Resource Actions:POST	Update
Resource Actions: PUT	Update

When your Web service is set up, it is ready to be consumed by a custom client. Windows and Web clients are unaffected by changes you make to the extaccess table. The operator's application profile is used to determine which tables the user can access, and which actions the user can perform.

List: Web Services available in the Service Manager Web Services API

The Service Manager Web Services includes ITIL-compliant Web Services. The following table lists some of those web services. To see all the Web Services that are ITIL-compliant, use Web Service Configuration in Tailoring (**Tailoring > Web Services > Web Service Configuration**) and then do a true search. This will list all of the out-of-box services.

Note: This is the out-of-box list for SOAP.

Web Service	URL to access WSDL	Service Manager objects (tables) published
Change Management	ChangeManagement.wsdl	Change (cm3r), ChangeTask (cm3t)
Configuration Management	ConfigurationManagement.wsdl	Company (company), Contact (contacts), Department (dept), Device (device), DeviceParent (deviceparent), Computer (joincomputer), DisplayDevice (joindisplaydevice), Furnishing (joinfurnishings), HandHeldDevice (joinhandhelds), MainFrame (joinmainframe), NetworkDevice (joinnetworkcomponents), OfficeElectronic (joinofficeelectronics), SoftwareLicense (joinsoftwarelicense), StorageDevice (joinstorage), TelecommunicationDevice (jointelecom), Location (location), Model (model), InstalledSoftware (pcsoftware), Vendor (vendor)
Incident Management	IncidentManagement.wsdl	Incident (probsummary)

Web Service	URL to access WSDL	Service Manager objects (tables) published
Problem Management	ProblemManagement.wsdl	Problem (rootcause)
Service Desk	ServiceDesk.wsdl	Call (incidents)
Service Level Management	ServiceLevelManagement.wsdl	ServiceEntry (serviceent), SLA (sla), ActiveSLA (slaactive), AssignedSLA (slaassigned), SLA Control (slacontrol), MonthlySLA (slamonthly), MonthlySLALag (slamonthlylag), SLAResponse (slaresponse)

For RESTful, you can find the out-of-box list from the Service Document, `http://<server>:<port>/SM/9/rest`.

Field names in the extaccess record

Implementers can change the field name and data type of a Service Manager field when they publish the field as part of a Web Service. To change the field name and data type of a Service Manager field, the implementer must specify the Service Manager field in the extaccess record using one of the formats listed in the following table.

Type of Service Manager field	Format required to specify field	Example field listing fromthe Web Services API
All primitive fields	field.name	initial.impact
array **field**	 field.name	 misc.array1
structure **field 1** **field 2** **field 3**	 structure.name,field.name.1 structure.name,field.name.2 structure.name,field.name.3	 header,agreement.id header,approval.status header,assigned.to

Type of Service Manager field	Format required to specify field	Example field listing fromthe Web Services API
array structure **field 1** **field 2** **field 3**	 array.name[field.name.1] array.name[field.name.2] array.name[field.name.3]	 affected.ci [ci.assign.group] affected.ci [ci.device.name] affected.ci[ci.device.type]
structure1 structure2 **field 1** **field 2** **field 3**	 structure.name.1,structure.name.2,field.name.1 structure.name.1,structure.name.2,field.name.2 structure.name.1,structure.name.2,field.name.3	 <*no example available*>
structure 1 array structure 2 **field 1** **field 2** **field 3**	 structure.name.1,array.name[field.name.1] structure.name.1,array.name[field.name.2] structure.name.1,array.name[field.name.3]	 <*no example available*>

Create dedicated Web Services listeners

An HP Service Manager system configured for vertical or horizontal scaling uses a Load Balancer to redirect client connection requests to an available Service Manager process. A system that also has many Web Services may need a Load Balancer for multiple nodes. Service Manager's Web Services do not support http redirect, and will fail to clean up the resources on the Service Manager loadBalancer process, if the loadBalancer port is used as the endpoint URL. For this reason, HP recommends creating one or more Service Manager processes dedicated to Web Services requests. You can then configure any external Web service clients to connect directly to the dedicated Service Manager processes. If your system needs a load balancer, use a hardware load balancer to balance between a set of servlets with the debugnode parameter.

1. Log in to the host running Service Manager with an administrator account.

2. Stop the Service Manager server.
   Note: It is not necessary to stop and start the Service Manager server to add a new port. You can add the line to the `sm.cfg` file while the system is running and start that same port from a command prompt manually.

3. Open the `sm.cfg` file, and create a dedicated Service Manager process to listen for Web Services requests using the `-debugnode` parameter. For example, the following entries create a dedicated process listening on ports 13085 and 13445.

   ```
   sm -httpPort:13080 -loadbalancer
   sm -httpPort:13081 -httpsPort:13443
   sm -httpPort:13083 -httpsPort:13444
   sm -httpPort:13085 -httpsPort:13445 -debugnode
   ```

   Note: The `debugnode` parameter tells the Service Manager Load Balancer not to forward any client connection requests to this Service Manager process. Only clients that directly connect to the process can access it.

4. Restart the Service Manager server.

5. Configure any external web service clients to connect directly to the Service Manager processes running in debugnode. For example, set the endpoint URL to `http://<fully qualified host name>:13085/SM/7/<Service Name>` for normal connections and set the URL to `https://<fully qualified host name>:13445/SM/7/<Service Name>` for SSL-encrypted connections.

Data conversion between Service Manager and SOAP Web Services

HP Service Manager has a more lenient data typing policy than the XML schema data typing policy used for Web Services. Certain field types in Service Manager can correspond to multiple data types in the XML schema data type policy. For example, the Service Manager data type decimal could be a decimal, a floating number, or an integer in the XML schema data type policy.

In addition, the actual formatting of data varies between Service Manager and XML schema data types. This is especially true of Service Manager date/time fields that use a different order than XML schema dates. Because some Web Services may require changes to field data format, you can now define the XML Schema data type you want Service Manager to convert the field's data to when you publish the field as part of a web service.

For outbound data, the Service Manager server automatically converts Service Manager data to the format you select in the data policy record for the Service Manager field. For inbound data, the Service Manager server automatically converts the XML schema data to the Service Manager field's listed data type format.

For example, the Service Manager Web Services API publishes the Service Manager field closed.time as ClosedTime in the IncidentManagement service. The Web Services API converts

the outbound Service Manager data into the appropriate ISO 8601 date format for XML schema. When the Web Service responds, the Web Service API converts the ISO-formatted date back into a Service Manager date format. Here is an example: the conversion between `1994-11-05T08:15:30-05:00` and November 5, 1994, 8:15:30 am, US Eastern Standard Time by the Web Services API.

The services, objects, and fields published in the Service Manager Web Services API already have the proper XML schema data mappings listed in the Web Services definition (extaccess record). If the extaccess record does not list a data type mapping, then the Web Services API treats the field data as a string field. Typically, you only need to add or change a Web Services API data type mapping to publish custom fields you have added to Service Manager as Web Services objects.

Warning: Changing the Web Services API data type mappings for existing fields in the Service Manager Web Services API may result in data mismatch errors.

Example: Publishing the Terminate Change functionality via Web Services

In the default Service Manager system, the Terminate Change functionality is not published via Web Services.

To publish the Terminate Change functionality, follow the steps described in this section.

Create a display option

Add a Display Option record to eliminate the prompt for a closure code and closing comments.

Field	Value
Screen ID	cm.view.display
Modifies Record	Leave blank
Action	terminatebg
Unique ID	cm.view.display_terminatebg
GUI option	6
Balloon Help (If Option < 200)	Terminate Change
Text Option	6
Default Label	Terminate Background

Field	Value
Bank	1
Text Alternative	Leave blank
Condition	`evaluate($L.tableAccess.close) and open in $L.file=true and nullsub($G.ess, false)=false and (category in $L.file="Release Management" and ($phasepntr=3 or $phasepntr=2 or $phasepntr=1))`
User Condition	`$G.bg=true`
RAD tab	
PreRad Expressions subtab	`$terminate.release=true`

Create a new process

Enter the following values in the Process Definition record to create the `terminate.release.bg` Process record.

Field	Value
Process Name	terminate.release.bg
Run in Window?	Select this option
RAD tab	
Expressions evaluated before RAD call	
`$L.file.vars={$L.category, $L.phase, $L.fc, $L.fc.master}` `if (index(current.phase in $L.file, phases in $L.category)=lng(denull(phases in $L.category))) then ($L.last=true) else ($L.last=false)`	
RAD Application	sla.confirm.outage
Condition	$L.last and enable in$G.sla.environment
Parameter Names	file
Parameter Values	$L.file
Expressions evaluated before RAD call	

Field	Value
$L.terminated.parent.name=number in $L.file;$terminate.ok=true;$terminate.release=true	
RAD Application	cm3.close.child.tasks
Condition	true
Parameter Names	name
Parameter Values	$L.terminated.parent.name
Expressions evaluated beforeRAD call	
$phasepnt=7;current.phase in $L.file="Verification" status in $file="terminated"	
RAD Application	cm.close
Condition	true
Parameter Names	record second.file boolean.1 prompt
Parameter Values	$L.file $L.object $L.bg $L.exit

Set up a State record

Add the following entries to the cm.view State record.

- Display Action: terminatebg
- Process Name: terminate.release.bg
- Condition: true
- Save First: (leave blank)

Set up an extaccess record

1. Update the extaccess record to expose this function. Select the extaccess record with Name = cm3r. Type **TerminateChange** in **Object Name** and click **Add**.
2. Add the following entries to the extaccess record:
   - Allowed Actions: terminatebg
   - Actions: Terminate
   - Action Type(Leave Blank)
3. If you want to use RESTful API, on RESTful tab, type the following:

Field	Value
RESTful Enabled	Checked
Attachment Enabled	Checked if you want to use attachment later.
Resource Collection Name	terminatechanges
Resource Name	TerminateChange
Unique Keys	header,number
Max Records Returned in Query	1,000
Query Authorization	lioption("Change Management") and (index("SysAdmin", $lo.ucapex)>0 or index("ChMAdmin", $lo.ucapex)>0 or index("change request", $lo.ucapex)>0)
Resource Collection Actions: POST	Create
Resource Actions:POST	Update
Resource Actions:PUT	Update

Execute a request via SOAP Web Services

Execute the following request via SOAP Web Services.

Note: The change number has to be a change of the Release Management category. ClosingComments and ClosureCode are required fields for terminating a Release Management change.

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:pws="http://servicecenter.peregrine.com/PWS"
xmlns:com="http://servicecenter.peregrine.com/PWS/Common">
   <soapenv:Header/>
   <soapenv:Body>
      <pws:TerminateChangeRequest attachmentInfo="?" attachmentData="?"
ignoreEmptyElements="true">
         <pws:model query="">
            <pws:keys query="">
               <!--Optional:-->
               <pws:ChangeNumber type="String" mandatory="?"
readonly="?">C10027</pws:ChangeNumber>
            </pws:keys>
            <pws:instance query="" uniquequery="?" recordid="?">
               <pws:header type="Structure">
                  <!--Optional:-->
                  <pws:ChangeNumber type="String" mandatory="?"
readonly="?"></pws:ChangeNumber>
                  <!--Optional:-->
                  <pws:Category type="String" mandatory="?"
readonly="?"></pws:Category>
                  <!--Optional:-->
                  <pws:Status type="String" mandatory="?"
readonly="?">terminated</pws:Status>
                  <!--Optional:-->
                  <pws:ApprovalStatus type="String" mandatory="?"
readonly="?"></pws:ApprovalStatus>
                  <!--Optional:-->
                  <pws:RequestedBy type="String" mandatory="?"
readonly="?"></pws:RequestedBy>
                  <!--Optional:-->
                  <pws:AssignedTo type="String" mandatory="?"
readonly="?"></pws:AssignedTo>
                  <!--Optional:-->
                  <pws:Coordinator type="String" mandatory="?"
readonly="?"></pws:Coordinator>
                  <!--Optional:-->
                  <pws:CoordinatorPhone type="String" mandatory="?"
readonly="?"></pws:CoordinatorPhone>
```

```
                <!--Optional:-->
                <pws:PlannedStartDate type="DateTime" mandatory="?"
readonly="?"></pws:PlannedStartDate>
                <!--Optional:-->
                <pws:PlannedEndDate type="DateTime" mandatory="?"
readonly="?"></pws:PlannedEndDate>
                <!--Optional:-->
                <pws:Reason type="String" mandatory="?"
readonly="?"></pws:Reason>
                <!--Optional:-->
                <pws:CurrentPhase type="String" mandatory="?"
readonly="?"></pws:CurrentPhase>
                <!--Optional:-->
                <pws:RiskAssessment type="String" mandatory="?"
readonly="?"></pws:RiskAssessment>
                <!--Optional:-->
                <pws:Priority type="String" mandatory="?"
readonly="?"></pws:Priority>
                <!--Optional:-->
                <pws:DateEntered type="DateTime" mandatory="?"
readonly="?"></pws:DateEntered>
                <!--Optional:-->
                <pws:Open type="Boolean" mandatory="?"
readonly="?"></pws:Open>
                <!--Optional:-->
                <pws:BackoutDuration type="Duration" mandatory="?"
readonly="?"></pws:BackoutDuration>
                <!--Optional:-->
                <pws:CloseTime type="DateTime" mandatory="?"
readonly="?"></pws:CloseTime>
                <!--Optional:-->
                <pws:ForeignID type="String" mandatory="?"
readonly="?"></pws:ForeignID>
                <!--Optional:-->
                <pws:RFCType2 type="String" mandatory="?"
readonly="?"></pws:RFCType2>
                <!--Optional:-->
                <pws:Company type="String" mandatory="?"
readonly="?"></pws:Company>
                <!--Optional:-->
                <pws:BriefDescription type="String" mandatory="?"
readonly="?"></pws:BriefDescription>
                <!--Optional:-->
                <pws:Subcategory type="String" mandatory="?"
readonly="?"></pws:Subcategory>
                <!--Optional:-->
                <pws:SLAAgreementID type="Int" mandatory="?"
readonly="?"></pws:SLAAgreementID>
             </pws:header>
```

```
                <pws:description.structure type="Structure">
                   <!--Optional:-->
                   <pws:Description type="Array">
                      <!--Zero or more repetitions:-->
                      <pws:Description type="String" mandatory="?"
readonly="?"></pws:Description>
                   </pws:Description>
                   <!--Optional:-->
                   <pws:Justification type="Array">
                      <!--Zero or more repetitions:-->
                      <pws:Justification type="String" mandatory="?"
readonly="?"></pws:Justification>
                   </pws:Justification>
                   <!--Optional:-->
                   <pws:BackoutMethod type="Array">
                      <!--Zero or more repetitions:-->
                      <pws:BackoutMethod type="String" mandatory="?"
readonly="?"></pws:BackoutMethod>
                   </pws:BackoutMethod>
                </pws:description.structure>
                <pws:middle type="Structure">
                   <!--Optional:-->
                   <pws:ConfigurationItem type="String" mandatory="?"
readonly="?"></pws:ConfigurationItem>
                   <!--Optional:-->
                   <pws:Location type="String" mandatory="?"
readonly="?"></pws:Location>
                   <!--Optional:-->
                   <pws:Misc1 type="String" mandatory="?"
readonly="?"></pws:Misc1>
                   <!--Optional:-->
                   <pws:Misc2 type="String" mandatory="?"
readonly="?">pass</pws:Misc2>
                   <!--Optional:-->
                   <pws:Misc3 type="String" mandatory="?"
readonly="?"></pws:Misc3>
                   <!--Optional:-->
                   <pws:Misc4 type="String" mandatory="?"
readonly="?"></pws:Misc4>
                   <!--Optional:-->
                   <pws:Misc5 type="String" mandatory="?"
readonly="?"></pws:Misc5>
                   <!--Optional:-->
                   <pws:Misc6 type="String" mandatory="?"
readonly="?"></pws:Misc6>
                   <!--Optional:-->
                   <pws:Misc7 type="String" mandatory="?"
readonly="?"></pws:Misc7>
                   <!--Optional:-->
```

```
                  <pws:Misc8 type="String" mandatory="?"
readonly="?"></pws:Misc8>
                  <!--Optional:-->
                  <pws:Misc9 type="String" mandatory="?"
readonly="?"></pws:Misc9>
                  <!--Optional:-->
                  <pws:Misc10 type="String" mandatory="?"
readonly="?"></pws:Misc10>
                  <!--Optional:-->
                  <pws:OutageStart type="DateTime" mandatory="?"
readonly="?"></pws:OutageStart>
                  <!--Optional:-->
                  <pws:OutageEnd type="DateTime" mandatory="?"
readonly="?"></pws:OutageEnd>
                  <!--Optional:-->
                  <pws:ScheduledOutageStart type="DateTime" mandatory="?"
readonly="?"></pws:ScheduledOutageStart>
                  <!--Optional:-->
                  <pws:ScheduledOutageEnd type="DateTime" mandatory="?"
readonly="?"></pws:ScheduledOutageEnd>
                  <!--Optional:-->
                  <pws:ActualOutageStart type="DateTime" mandatory="?"
readonly="?"></pws:ActualOutageStart>
                  <!--Optional:-->
                  <pws:ActualOutageEnd type="DateTime" mandatory="?"
readonly="?"></pws:ActualOutageEnd>
                  <!--Optional:-->
                  <pws:MiscArray1 type="Array">
                     <!--Zero or more repetitions:-->
                     <pws:MiscArray1 type="String" mandatory="?"
readonly="?"></pws:MiscArray1>
                  </pws:MiscArray1>
                  <!--Optional:-->
                  <pws:MiscArray2 type="Array">
                     <!--Zero or more repetitions:-->
                     <pws:MiscArray2 type="String" mandatory="?"
readonly="?"></pws:MiscArray2>
                  </pws:MiscArray2>
                  <!--Optional:-->
                  <pws:MiscArray3 type="Array">
                     <!--Zero or more repetitions:-->
                     <pws:MiscArray3 type="String" mandatory="?"
readonly="?">test passed</pws:MiscArray3>
                  </pws:MiscArray3>
                  <!--Optional:-->
                  <pws:Assets type="Array">
                     <!--Zero or more repetitions:-->
                     <pws:Assets type="String" mandatory="?"
readonly="?"></pws:Assets>
```

```
                  </pws:Assets>
                  <!--Optional:-->
                  <pws:EstimateDescription type="String" mandatory="?"
readonly="?"></pws:EstimateDescription>
                  <!--Optional:-->
                  <pws:EstimatePrice type="String" mandatory="?"
readonly="?"></pws:EstimatePrice>
                  <!--Optional:-->
                  <pws:ActualCost type="String" mandatory="?"
readonly="?"></pws:ActualCost>
                  <!--Optional:-->
                  <pws:ActualPrice type="String" mandatory="?"
readonly="?"></pws:ActualPrice>
               </pws:middle>
               <pws:close type="Structure">
                  <!--Optional:-->
                  <pws:CompletionCode type="Decimal" mandatory="?"
readonly="?">1</pws:CompletionCode>
                  <!--Optional:-->
                  <pws:ClosingComments type="Array">
                     <!--Zero or more repetitions:-->
                     <pws:ClosingComments type="String" mandatory="?"
readonly="?">Terminating Change</pws:ClosingComments>
                  </pws:ClosingComments>
               </pws:close>
               <!--Optional:-->
               <pws:Urgency type="String" mandatory="?"
readonly="?"></pws:Urgency>
               <!--Optional:-->
               <pws:InitialAssessment type="String" mandatory="?"
readonly="?"></pws:InitialAssessment>
               <!--Optional:-->
               <pws:attachments>
                  <!--Zero or more repetitions:-->
                  <com:attachment href="?" contentId="?" action=""
name="?" type="?" len="?" charset="?" attachmentType="?"/>
               </pws:attachments>
            </pws:instance>
            <!--Optional:-->
            <pws:messages>
               <!--1 or more repetitions:-->
               <com:message type="String" mandatory="?" readonly="?"
severity="?" module="?"></com:message>
            </pws:messages>
         </pws:model>
      </pws:TerminateChangeRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

Response to a request via SOAP Web Services

The response to a request via SOAP Web Services is as follows:

```
<SOAP-ENV:Envelope xmlns:SOAP
-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
      <TerminateChangeResponse message="Success" returnCode="0"
schemaRevisionDate="2008-05-21" schemaRevisionLevel="5" status="SUCCESS"
xsi:schemaLocation="http://servicecenter.peregrine.com/PWS http://<sm
server>.americas.hpqcorp.net:13701/sc62server/ws/Change.xsd"
xmlns="http://servicecenter.peregrine.com/PWS"
xmlns:cmn="http://servicecenter.peregrine.com/PWS/Common"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
         <model>
            <keys>
               <ChangeNumber type="String">C10027</ChangeNumber>
            </keys>
            <instance recordid="C10027 - test"
uniquequery="header,number=&amp;quot;C10027&amp;quot;">
               <header type="Structure">
                  <ChangeNumber type="String">C10027</ChangeNumber>
                  <Category type="String">Release Management</Category>
                  <Status type="String">terminated</Status>
                  <ApprovalStatus type="String">approved</ApprovalStatus>
                  <RequestedBy type="String">ALSTON, LOU</RequestedBy>
                  <Coordinator type="String">CM 3</Coordinator>
                  <Reason type="String">problem</Reason>
                  <CurrentPhase type="String">Verification</CurrentPhase>
                  <Priority type="String">1</Priority>
                  <DateEntered type="DateTime">2008-05-
27T16:34:26+00:00</DateEntered>
                  <Open type="Boolean">false</Open>
                  <BackoutDuration
type="Duration">P0DT0H0M0S</BackoutDuration>
                  <CloseTime type="DateTime">2008-05-
27T16:34:26+00:00</CloseTime>
                  <Company type="String">advantage</Company>
                  <BriefDescription type="String">test</BriefDescription>
               </header>
               <description.structure type="Structure">
                  <Description type="Array">
                     <Description type="String">test</Description>
                  </Description>
               </description.structure>
               <middle type="Structure">
                  <Location type="String">North America</Location>
```

```
                  <Misc2 type="String">pass</Misc2>
                  <MiscArray3 type="Array">
                     <MiscArray3 type="String">test passed</MiscArray3>
                  </MiscArray3>
               </middle>
               <close type="Structure">
                  <CompletionCode type="Decimal">1</CompletionCode>
                  <ClosingComments type="Array">
                     <ClosingComments type="String">Terminating
Change</ClosingComments>
                  </ClosingComments>
               </close>
               <Urgency type="String">1</Urgency>
               <InitialAssessment type="String">1</InitialAssessment>
            </instance>
         </model>
         <messages>
            <cmn:message type="String">Audit Record successfully recorded
and added.</cmn:message>
            <cmn:message type="String">Change C10027 Phase Verification
Closed by System Administrator.</cmn:message>
         </messages>
      </TerminateChangeResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>END
```

Execute a request via RESTful Web Services

Execute the following request via RESTful Web Services.

Note: The change number must be a change of the Release Management category. **ClosingComments** and **ClosureCode** are required fields for terminating a Release Management change.

POST an action terminate to a change which is in **Evaluation & Change Closure** phase. The json data resembles:

```
{ "TerminateChange":    {
      "close":{
         "ClosingComments":"closureComments tested",
         "ClosureCode":"1"
      }
   }
}
```

Response to a request via RESTful Web Services

The response to a request via RESTful Web Services resembles:

```
{
   "Messages": [],
   "ReturnCode": 0,
   "TerminateChange":    {
      "Impact": "4",
      "RequestedEndDate": "2007-10-23T21:06:00+00:00",
      "Service": "Applications",
      "Urgency": "1",
      "close":       {
         "ClosingComments": ["closureComments tested"],
         "ClosureCode": 1
      },
      "description.structure": {"Description": ["Check and clean system on virus
ses."]},
      "header":       {
         "ApprovalStatus": "approved",
         "AssignmentGroup": "Application",
         "BackoutDuration": "P0DT0H0M0S",
         "Category": "Maintenance",
         "ChangeCoordinator": "Change.Coordinator",
         "ChangeID": "C10019",
         "Company": "advantage",
         "DateEntered": "2013-07-02T09:16:29+00:00",
         "InitiatedBy": "BERRY, ELLIS",
         "Open": true,
         "Phase": "Evaluation & Change Closure",
         "PlannedEnd": "2008-10-19T18:00:00+00:00",
         "PlannedStart": "2008-10-12T18:00:00+00:00",
         "Priority": "2",
         "RiskAssessment": "5",
         "Status": "initial",
         "Subcategory": "Maintenance",
         "Title": "Multiple virusses"
      },
      "middle":       {
         "Assets": ["Norton Anti-Virus"],
         "ConfigurationItem": "Norton Anti-Virus",
         "Location": "North America"
      }
   }
}
```


Publish a table as a Web service

You must have the SysAdmin capability words to use this procedure.

1. Login to Service Manager as a System Administrator.
2. Click **Tailoring > Database Manager**.
3. In the Table field, type **extaccess**.
4. Click **Search**.
5. In the Name field, select the name of the Service Manager table or join file you want to publish as a web service.

   Important: Type the name of the table as it is defined in the database dictionary.

   Note: Only valid Service Manager table names appear in the list. This list includes the names of tables that do not physically reside in the database, but are defined in memory at run time based on join definitions and relationship information in joindef and erddef records respectively.
6. In the Service Name field, type the name of the Web service you want to use to publish this table. You can reuse the same web service name to publish multiple tables, as long the combination of Service Name and Object Name is unique.

   Important: Since this name becomes part of a URL, the name must consist of alphanumeric characters that are valid for URLs. The name cannot consist of URL reserved characters such as spaces, slashes, or colons.

   Note: The name you type in this field becomes the alias name for service and becomes part of the Web service URL. For example, if you type IncidentManagement for the service name, then SOAP applications must include IncidentManagement.wsdl in the URL to access this service.
7. In the Object Name field, type the name you want to use to identify this table.

   Note: The name is unique and cannot be used by other Web Services definitions.

   Note: The name you type in this field becomes the alias name for the table and becomes part of the Web service WSDL. For example, if you type Incident for the object name, then the SOAP operations for this table include Incident as part of the WSDL element (such as RetrieveIncident, CreateIncident, and ResolveIncident).

   Important: Since this name becomes part of the WSDL, the name must consist of alphanumeric characters valid for XML. The name cannot consist of XML reserved characters such as brackets (< & >), colons (:), or quotation marks (" & ’).
8. In the Allowed Actions array, select the Service Manager Document Engine display actions you want to globally enable for this table.

   Note: Each table has its own set of display actions allowed as defined in the Service Manager Document Engine. Enabling or disabling the display actions from this field only determines

whether the display action is available through the Web Services API. Service Manager still validates the operator credentials supplied with each Web service request to ensure that the operator has sufficient privileges to perform the display action. Click the array field to see a list of allowable display actions for the table you select.

Note: If a join file is chosen, the allowed actions for the join file come from the primary table of the join.

9. In the Action Names field, type the name you want to use in the Web Services API to identify the Document Engine display actions for this table.

   Note: The name you type for this field becomes the alias name for the display action and becomes part of the Web service WSDL. For example, if you type Create for the add action of the Incident object, then the WSDL operation becomes CreateIncident and the WSDL messages are CreateIncidentRequest and CreateIncidentResponse.

   Important: Since this name becomes part of the WSDL, the name must consist of alphanumeric characters valid for XML. The name cannot consist of XML reserved characters such as brackets (< & >), colons (:), or quotation marks (" & ').

10. If you want to use RESTful API, you need to configure RESTful tab field too. Type the name of Resource Collection and Resource you want to use, and set the Unique Key and default actions.

11. Click **Add**.

Users can now access this Service Manager table from a custom or third-party Web Service client and use the actions you have enabled.

Expose a table with more than one Web service

User role: System Administrator

An implementer can define multiple Web service definition records with different names for a given table or join file, and have different fields and actions exposed for each.

1. Click **Tailoring > Web Services > Web Service Configuration Utility**.

2. In the Table field, type **extaccess**, and then click **Search**. The External Access Definition form opens.

3. In the Name field, select or type the name of the table or join file for which you want to create a copy of the extaccess record, and then click **Search**. The record opens.

4. Change the Service Name to the name of the web service you want to use to publish the Service Manager table.

Important Note: The combination of Service Name and Object Name must be unique to this record. The combination cannot exist anywhere else in the system.

5. Change the Object Name to the name you want to use to identify the Service Manager table in the Web Services API.
6. On the Fields tab, change the fields that are exposed and modify the Caption and Type information, if necessary.

   Note: If a join file is chosen, the Fields tab lists all the fields for all the files in that join file.
7. On the Allowed Actions tab, change the actions, if necessary.
8. On the Expressions tab, add expressions, if necessary.
9. On the RESTful tab, add RESTful API configurations, if necessary.
10. Click **Add**.

The new extaccess record is added to the system. When you view the exposed Object (for example, WSDL for SOAP) for both web services, they should display with the applicable actions and fields, as defined in each extaccess record.

Remove a Document Engine display action from a Web service

You must have the SysAdmin capability word to use this procedure.

The Service Manager Web Services Configuration Utility allows you to remove any Document Engine display action you published as part of a Web service.

1. Click **Tailoring > Web Services > Web Service Configuration**.
2. In Service Name, type the name of the service.
3. In the Name field, type the name of the Service Manager table whose display actions you want to remove.
4. From the Allowed Actions array, select the Document Engine display action you want to remove from the list.
5. Clear the Allowed Actions field with the Backspace key.
6. Click **Save**.

Remove a Service Manager field from a Web service

You must have the SysAdmin capability words to use this procedure.

1. Click **Tailoring > Web Services > Web Service Configuration Utility**.
2. In the Table field, type **extaccess** and click **Search**. The External Access Definition record opens.
3. In the Name field, select the name of the Service Manager table in which you want to remove fields.
4. Click **Search**. The Web Services record for that table opens.
5. On the Fields tab, find the fields you want to remove and make the value that is currently there NULL.
6. In the Caption column, make the value NULL for the field you want to remove.
7. In the Type column, make the value NULL for the field you want to remove.
8. Click **Save**.

Sample client for SOAP Web Services SM7 URL

The HP Service Manager server includes a sample Web Services client application for the `http://servername:port_number/SM/7/service_name.wsdl`. The sample application was created for Apache™ Axis2 (version 2.1.4). If you have Axis 2.1.4 and Apache™ Ant installed, you can review and update the source code of the sample application as well as generate updated proxy code to test the Service Manager Web Services functionality. The Apache Axis2 sample is written in Java. The sample client application is included with the server installation in the following folder:

```
<Service Manager server installation folder>

\webservices\sample\sm7webservice
```

The sample includes the source code for the client applications as well as support files for the Web Services development environment. The Apache Axis2 jar files are included and they are located under the "lib" folder. A set of the batch files that you can use to run each class are located under the "bin" folder and you can run each class from the Windows command prompt after you have compiled the sample Java. You can use the sample application as an example of how to create your own custom Web Services client applications.

Note: All the sample applications use a command line interface. To see the usage information for the command line interface, change to "bin" folder, type **xxxSample** where xxxSample is the batch file name of the sample application.

The Apache Axis2 sample client application assumes that you have a Service Manager server instance running from the local host. If this is not the case, you can change the server host name and port number using the sample's command line interface.

Each of the sample folders includes a readme file that contains valuable information about using the sample application found in each of the sample folder.

The sample client application contains examples of how to send the MTOM attachments to the Service Manager server.

Configuration Management sample

The sample client applications contain the following classes for Configuration Management. Refer to the sample application source code for comments on the usage of each class.

Field	Description
ConfigurationManagementServiceUtility	• Provides the CreateService method to initialize an object for the service. • Provides the InitServiceAuthentication method to send the host name, communications port, operator name, and operator password with each SOAP request.
CreateContactSample	Creates a contact record with the supplied parameters.
DeleteContactSample	Deletes the contact record listed in the supplied parameters.
RetrieveContactSample	Retrieves a single contact record matching the supplied parameters.
UpdateContactSample	Updates a contact record with the supplied parameters.

Incident Management sample

The sample client applications contain the following classes for Incident Management. Refer to the sample application source code for comments on the usage of each class.

Class	Description
CloseIncidentSample	Closes an incident record with the supplied parameters.
CreateIncidentSample	Creates an incident record with the supplied parameters.

Class	Description
IncidentManagementServiceUtility	• Provides the CreateService method to initialize an object for the service. • Provides the InitServiceAuthentication method to send the host name, communications port, operator name, and operator password with each SOAP request.
ResolveIncidentSample	Resolves an incident record matching the supplied parameters.
RetrieveIncidentListSample	Retrieves multiple incident records matching the supplied parameters.
RetrieveIncidentSample	Retrieves a single incident record matching the supplied parameters.
UpdateIncidentSample	Updates an incident record with the supplied parameters.

Command line arguments for the Axis2 sample application

The Axis2 sample application runs from the command prompt using Java. After you have compiled the Axis2 sample into an executable class files, you can perform configuration and incident management tasks with the following arguments.

Note: To see the usage information for the Axis2 sample application, type: **ClassName** where ClassName is the name of a sample application class.

Configuration Management

The following commands invoke Configuration Management functionality. These examples assume you are using the batch files provided with the Axis2 sample application to automatically set the class path and call the proper executable class.

Operation	Command-line example
Create contact	CreateContactSample -name "FALCON, MERLINE2" -fullname "MERLINE2 FALCON"
Delete contact	DeleteContactSample DeleteContactSample -name "FALCON, MERLINE2"
Retrieve contact	RetrieveContactSample RetrieveContactSample -name "FALCON, MERLINE2"
Update Contact	UpdateContactSample UpdateContactSample -name "FALCON, MERLINE2" -email "fmerline2@hp.com"

Incident Management

The following commands invoke Incident Management functionality. These examples assume you are using the batch files provided with the Axis2 sample application to automatically set the class path and call the proper executable class.

Operation	Command-line example
Close incident	CloseIncidentSample -incidentId IM10001 -closeCode "User Closer" -resolution "Problem disappeared"
Create incident	CreateIncidentSample -briefDescription "Java sample brief description" -category incident -incidentDescription "This is a description" -severity 1 -subCategory hardware -productType "missing or stolen" -initialImpact 1 -service Applications -primaryAssignmentGroup Networks
Create incident with attachment (s)	CreateIncidentSample -briefDescription "Java sample brief description" -category incident -incidentDescription "This is a description" -severity 1 -subCategory hardware -productType "missing or stolen" -initialImpact 1 -service Applications -primaryAssignmentGroup Network -attachment 101.jpg:README.txt
Resolve incident	ResolveIncidentSample -incidentId IM10006 -resolution "Problem disappeared"
Retrieve incident list	RetrieveIncidentListSample -incidentId IM10001:IM10002
Retrieve incident	RetrieveIncidentSample -incidentId IM1001
Update incident	UpdateIncidentSample -incidentId IM10006 -journalUpdates "User provided more information"

The CreateIncicentSample and UpdateIncidentSample classes can send MTOM attachments to Service Manager server. The command line argument is -attachment file_01:file_02. You can send more than one attachment to Service Manager server. Be sure to place the attachments in the `<SM_installation_directory>\webservices\sample\sm7webservices\Axis2Sample\bin\resources` directory.

Add an external access action to the Web Services

You must have the SysAdmin capability words to use this procedure.

1. Click **Tailoring > Web Services > External Access Actions**. Service Manager displays the External Access Actions form.

2. In External Action ID, type a unique ID name.

3. In RAD/ScriptLibrary.function, type the name of the RAD or JavaScript function you want to make available as a custom action in the Web Services API.

   Note: To specify a script from the Script Library, use the following format:
   <script name>.<function name>
   For example, Approval.buildAllStatus.

4. In Type, select RAD to if your custom action is a RAD function or select JavaScript if your custom action is a JavaScript.

5. In Description, type the name you want custom action to have.

   Note: Service Manager displays the name you type here as the Custom Action to Perform in the External Access Defintion form.

   The type you select determines what Parameters array Service Manager displays. If you select RAD, Service Manager displays an array with Parameter Names and Parameter Values fields. If you select JavaScript, Service Manager displays an array with only the Parameter Values field.

6. Type any required input parameters of the RAD function or JavaScript in the parameters array.

   RAD functions require values in both the Parameter Names and Parameter Values fields. Each RAD function has its own list of required RAD parameters names. RAD parameter values are typically system variables such as $L.file or$L.exit. You can type RAD function parameters in any order.

   JavaScript parameters only require the Parameter Values field, but require you to type them in the same order as the JavaScript function expects them. For example, the buildAllStatus function of the Approval script expects the following parameters in the following order:
   a. record

   b. fApprovalDef

   c. keepRoleOld

   d. keepRoleNew

   e. tokens

   f. tokenToDescription

7. Click **Add** to create your custom Web Services action.

Chapter 3

SOAP API

This chapter introduces the SOAP API used in Service Manager.

Web Services Description Language (WSDL)

The W3C describes WSDL in the W3C Note 15 March 2001 as "*WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate, however, the only bindings described in this document describe how to use WSDL in conjunction with SOAP 1.1, HTTP GET/POST, and MIME*." In other words, the WSDL defines a URL endpoint that publishes objects and methods usable against the publishing application. These objects and methods can then be used to communicate to that application.

Basic operations in WSDL files

Each Web Service that HP Service Manager publishes has a set of valid operations that an administrator can enable or disable for custom Web Services clients. The list of valid Web Service operations comes from two sources:

- The Document Engine display actions defined for each Service Manager table
- The common operations available to all Web Services

The `<Operation Name>` is the alias name of the Service Manager display option as defined in the Web Services Configuration Utility. The `<Object Name>` is the alias name of the Service Manager table as published in the Web Service. Use a Request message to send SOAP operations to the Service Manager server. The Service Manager server uses a Response message to send its reply to the SOAP operation.

You can see the list of available Document Engine display actions for each table in the extaccess table. The Service Manager server converts each published display action into a separate `<operation>` element in the Web Services Definition Language (WSDL).

For example, the Resolve operation for the Incident object translates to the ResolveIncidentRequest SOAP message. The Service Manager server replies with a ResolveIncidentResponse SOAP message. Any custom Web Services client you create must be able to generate these SOAP message requests and understand the SOAP message response.

In addition to application-specific display actions, there are common operations available to all Service Manager Web Services objects. Just as with display options, the Service Manager server

converts each common operation into a separate `<operation>` element in the Web Services Definition Language (WSDL). The following common messages are always available.

- **Retrieve<Object>Request** – retrieves a single record detail matching the value of the `<keys>` element or query attribute, for example an Incident record.
- **Retrieve<Object>KeysList** – retrieves a list of keys matching the value of query attribute.
- **Retrieve<Object>List** – retrieves a list of objects matching the value of query attribute.

The following common messages but are not always available.

- **Update<Object>Request** – updates a single record matching the value of the `<keys>` element or query attribute with the new values defined in the `<instance>` element
- **Delete<Object>Request** * – deletes a single record matching the value of the `<keys>` element
- **Create<Object>Request** – adds a single record with the values defined in the `<instance>` element

* The IncidentManagement Web Service does not offer the delete operation. To retrieve a single incident record, you can use the RetrieveIncident operation.

For more information about Web Services and WSDL, see the W3C Web site.

Service Manager WSDL files

You can view the Web Services Description Language (WSDL) for any Service Manager Web Service by navigating to one of the following URLs:

Version	URL	Supports
Backwards compatibility for HP ServiceCenter 6.2 servlet mode	http://<*servername*>:<*port_number*>/sc62server/PWS/<*service_name*>.wsdl	MIME attachments
Service Manager	http://<*servername*>:<*port_number*>/SM/7/<*service_name*>.wsdl	MTOM attachments

For example, type `http://myserver:13080/SM/7/IncidentManagement.wsdl` to view the Incident Management service WSDL from myserver.

The server also responds to requests with `?WSDL` as the file extension. For example, `http://myserver:13080/SM/7/IncidentManagement?wsdl`

The Service Manager server automatically generates a WSDL whenever it receives an HTTP *get* request for WSDL. Service Manager WSDLs use XML Schema definitions to describe literal Web Services. Service Manager is able to serve two different versions of the WSDL for a given service:

- HP ServiceCenter 6.2 WSDL files for backwards compatibility. The API described in these WSDL files is deprecated. See the HP ServiceCenter 6.2 documentation for more information.
- Service Manager WSDL . New applications should use this WSDL.

Note: To avoid receiving the "Invalid XML schema: Element <xs:import> is not allowed at this location under element <xs:schema>" error when viewing any multiple object WSDL (for example, ConfigurationManagement.wsdl), disable the validation in the SOAP tool you are using before loading the WSDL and creating a Web Service request.

The XML document which describes a particular Service Manager record (such as a Change or Incident) is wrapped in an outer document called a "model". The model is nothing more than a container for separating the actual data (the “instance” part) from the "keys" part, which is metadata about the fields that make up the primary key of the object.

Types of Web Services in Service Manager

The types of Web Services supported by Service Manager are as follows:

- Service Manager 7.x URL supporting the W3C Message Transmission Optimization Mechanism (MTOM) attachments, which is a method of efficiently sending binary data to and from Web Services. MTOM is usually used with XML-binary Optimized Packaging (XOP).

  ```
  http://<SM Server>:<SM port>/SM/7/<service name>.wsdl
  ```

Note: AXIS2 supports MTOM.

- Service Manager 7.x URL supporting Multipurpose Internet Mail Extensions (MIME), which is an Internet standard that extends the format of email to support MIME attachments. MIME's use has grown beyond describing the content of email to describing content type in general, including for the Web.

  ```
  http://<SM Server>:<SM port>/sc62server/PWS/<service name>.wsdl
  ```

Which URL to use depends on the consumer side. You also need to consider whether it supports MTOM or MIME. For example, Microsoft applications tend to support MIME.

WSDL document structure

A WSDL document is simply a **set of definitions**. There is a **definitions** element at the root and definitions inside. A WSDL document defines **services** as collections of network endpoints or **ports**. In a WSDL document, the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings. This allows the reuse of abstract definitions: **messages**, which are abstract descriptions of the data being exchanged, and **port types** which are abstract collections of **operations**. The concrete protocol and data format specifications for a particular port type constitutes a reusable **binding**. A port is defined by associating a network address with a reusable binding. A collection of ports defines a service.

A WSDL document uses the following elements in the definition of network services:

- **Types**– a container for data type definitions using some type system (such as XSD).
- **Message**– an abstract, typed definition of the data being communicated.
- **Operation**– an abstract description of an action supported by the service.
- **Port Type**–an abstract set of operations supported by one or more endpoints.
- **Binding**– a concrete protocol and data format specification for a particular port type.
- **Port**– a single endpoint defined as a combination of a binding and a network address.
- **Service**– a collection of related endpoints.

XML header

The XML header specifies the XML version number, and optionally the character encodings, as part of a grammar document's XML declaration on the first line of the document.

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

Namespace definitions

XML namespaces are used for providing uniquely named elements and attributes in an XML document

The following section of the Service Manager IncidentManagement wsdl shows the namespace definitions.

```
- <definitions targetNamespace="http://servicecenter.peregrine.com/PWS"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:cmn="http://servicecenter.peregrine.com/PWS/Common"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:ns="http://servicecenter.peregrine.com/PWS"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.xmlsoap.org/wsdl/
  http://schemas.xmlsoap.org/wsdl/">
```

Operation section

The following section of the example Service Manager IncidentManagement wsdl shows the operation section used to define each individual action supported by the service.

```
- <operation name="RetrieveIncident">
  <documentation />
  <input message="ns:RetrieveIncidentRequest" />
  <output message="ns:RetrieveIncidentResponse" />
```

```
  </operation>
  </portType>
```

Messages section

The following section of the example Service Manager IncidentManagement wsdl shows the messages section used to define the data being communicated.

```
- <message name="RetrieveIncidentRequest">
  <part element="ns:RetrieveIncidentRequest"
   name="RetrieveIncidentRequest" />
  </message>
- <message name="RetrieveIncidentResponse">
  <part element="ns:RetrieveIncidentResponse"
   name="RetrieveIncidentResponse" />
  </message>
  </message>
```

Types section

The following section of the example Service Manager IncidentManagement wsdl shows the definition of the data, including data types, that is being communicated between the consumer and Service Manager.

```
- <types>
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  attributeFormDefault="unqualified" elementFormDefault="qualified"
  targetNamespace="http://servicecenter.peregrine.com/PWS"
  version="2007-04-14 Rev 1"
xmlns="http://servicecenter.peregrine.com/PWS"
xmlns:cmn="http://servicecenter.peregrine.com/PWS/Common">
  <xs:import namespace="http://servicecenter.peregrine.com/PWS/Common"
  schemaLocation="http://server:13080/sc62server/PWS/Common.xsd" />
- <xs:complexType name="IncidentKeysType">
- <xs:sequence>
  <xs:element minOccurs="0" name="IncidentID" nillable="true"
  type="cmn:StringType" />
  </xs:sequence>
  <xs:attribute name="query" type="xs:string" use="optional" />
  </xs:complexType>
- <xs:complexType name="IncidentInstanceType">
- <xs:sequence>
  <xs:element minOccurs="0" name="IncidentID" nillable="true"
   type="cmn:StringType" />
- <xs:element minOccurs="0" name="IncidentDescription">
- <xs:complexType>
- <xs:complexContent>
- <xs:extension base="cmn:ArrayType">
- <xs:sequence>
```

```
  <xs:element maxOccurs="unbounded" minOccurs="0"
   name="IncidentDescription" type="cmn:StringType" />
  </xs:sequence>
  </xs:extension>
  </xs:complexContent>
  </xs:complexType>
  </xs:element>
- <xs:complexType name="IncidentModelType">
- <xs:sequence>
  <xs:element name="keys" type="IncidentKeysType" />
  <xs:element name="instance" type="IncidentInstanceType" />
  <xs:element minOccurs="0" name="messages" type="cmn:MessagesType" />
  </xs:sequence>
  <xs:attribute name="query" type="xs:string" use="optional" />
  </xs:complexType>
- <xs:element name="RetrieveIncidentRequest">
- <xs:complexType>
- <xs:sequence>
  <xs:element name="model" type="IncidentModelType" />
  </xs:sequence>
  <xs:attribute name="attachmentInfo" type="xs:boolean" use="optional" />
  <xs:attribute name="attachmentData" type="xs:boolean" use="optional" />
  <xs:attribute default="true" name="ignoreEmptyElements" type="xs:boolean" use=
"optional" />
  </xs:complexType>
  </xs:element>
- <xs:element name="RetrieveIncidentResponse">
- <xs:complexType>
- <xs:sequence>
  <xs:element name="model" type="IncidentModelType" />
  <xs:element minOccurs="0" name="messages" type="cmn:MessagesType" />
  </xs:sequence>
  <xs:attribute name="status" type="cmn:StatusType" use="required" />
  <xs:attribute name="message" type="xs:string" use="required" />
  <xs:attribute name="schemaRevisionDate" type="xs:date" use="required" />
  <xs:attribute name="schemaRevisionLevel" type="xs:int" use="required" />
  <xs:attribute name="returnCode" type="xs:decimal" use="optional" />
  <xs:attribute name="query" type="xs:string" use="optional" />
  </xs:complexType>
  </xs:element>
  </types>
```

Nillable attribute

The nillable attribute specifies whether or not an explicit NULL value can be assigned to the element. True enables an instance of the element to have the NULL attribute set to true. The NULL attribute is defined as part of the XML Schema namespace for instances. The default is false. This attribute is optional.

The nillable attribute is analogous to the SQL concept of NULL and is useful for dealing with the ambiguity that may otherwise surround an empty XML element value. With SQL there is a difference between a NULL value and a column containing a varchar of length zero. Similarly, in an XML schema there is a difference between an XML element containing no text value and one which is explicitly marked with xsi:nil=”true”.

Unless the XML schema indicates that an XML element is nillable, you cannot specify the nillable attribute for the element.

The following sample code with the nillable attribute can be found in the schema definition section:

```
<xs:element minOccurs="0" name="IncidentID" nillable="true"
type="cmn:StringType" />
```

Port type

The port defines the connection point to a Web Service. The follow section of the example Service Manager IncidentManagement WSDL shows the port type section, which includes the set of operations allowed by the endpoint.

```
- <portType name="IncidentManagement">
```

Binding section

The following section of the example Service Manager IncidentManagement WSDL shows the binding section used to define a protocol and defined data formats for a particular port type.

```
- <binding name="IncidentManagement" type="ns:IncidentManagement">
  <soap:binding style="document"
   transport="http://schemas.xmlsoap.org/soap/http" />
- <operation name="RetrieveIncident">
  <soap:operation soapAction="Retrieve" style="document" />
- <input>
  <soap:body use="literal" />
  </input>
- <output>
  <soap:body use="literal" />
  </output>
  </operation>
- <operation name="RetrieveIncidentKeysList">
  <soap:operation soapAction="RetrieveKeysList" style="document" />
- <input>
  <soap:body use="literal" />
  </input>
- <output>
  <soap:body use="literal" />
  </output>
  </operation>
  </binding>
```

Service section

The Service section describes one or more concrete endpoints where the functionality of the service can be found.

The follow section of the example Service Manager IncidentManagement WSDL shows the service section, which is a collection of endpoints (In this example, just IncidentManagement).

```
- <service name="IncidentManagement">
```

Port section

The following section of the example Service Manager IncidentManagement WSDL shows the port section, which is a single endpoint defined as a combination of a binding and a network address.

```
- <port binding="ns:IncidentManagement" name="IncidentManagement">
  <soap:address location="http://server:13080/sc62server/ws" />
  </port>
  </service>
  </definitions>
```

Change example to use the cookie

To change the Keep-Alive example to use the cookie you can modify the following methods.

The method createService ( ) in IncidentManagementServiceUtility.java file:

```
public static IncidentManagementStub createService(Map arguments)
throws Exception
  {
    String host = (String) arguments.get(ARGUMENT_HOST);
    String port = (String) arguments.get(ARGUMENT_PORT);
    String address = "http://" + host + ":" + port + "/SM/7/ws";
    IncidentManagementStub stub = new IncidentManagementStub(address);

    stub._getServiceClient().getOptions().setManageSession(true);
    stub._getServiceClient().getOptions().setProperty
       (HTTPConstants.REUSE_HTTP_CLIENT,true);
    // set connection: close
    //Header hdr = new Header(HTTPConstants.HEADER_CONNECTION,
        HTTPConstants.HEADER_CONNECTION_CLOSE);
    //ArrayList<Header> headers = new ArrayList<Header>();
    //headers.add(hdr);
    //stub._getServiceClient().getOptions().setProperty
       (HTTPConstants.HTTP_HEADERS, headers);
    stub._getServiceClient().getOptions().setProperty
        (Constants.Configuration.ENABLE_MTOM,
        Constants.VALUE_TRUE);
```

```
    ServiceUtility.initServiceAuthentication(stub, arguments);

    return stub;
  }
```

The method createIncidents() in CreateIncidentSample.java file:

```
public void createIncidents() throws Exception, IOException
  {
    /* Open a port to the Incident Management Web Service */
    IncidentManagementStub stub =
       IncidentManagementServiceUtility.createService(arguments);
    int totalIM = 10;

    /* Create details about the new incident */
    for (int i = 1; i <= totalIM; i++)
    {
      if (i == totalIM)
      {
        // close the connection if this is the last request
        Header hdr = new Header(HTTPConstants.HEADER_CONNECTION,
           HTTPConstants.HEADER_CONNECTION_CLOSE);
        ArrayList<Header> headers = new ArrayList<Header>();
        headers.add(hdr);
        stub._getServiceClient().getOptions().setProperty
           (HTTPConstants.HTTP_HEADERS, headers);
      }

      createIncident(stub);
    }

    return;
  }
```

The client is responsible for echoing back this value in a Cookie header in all subsequent POST requests. If the client fails to do this, the servlet container will quickly run out of sessions.

If a client request causes any Service Manager Server error or exception then this session will be terminated by the Service Manager Server. Once this happens the current JSESSIONID becomes invalid and a new JSESSIONID will be returned on the following client request. The SOAP client should echo back the new JSESSIONID for the subsequent requests to avoid the user login/logout overhead and dangling sessions saturation.

Verify the WSDL to JS output

Generated JavaScript must end with

```
// Ensure that material in lib.SOAP is available

lib.SOAP.init();

/// End ----------------
```

All defined types and operations must be represented by a function such as

```
this.SOAPOperations[ "UpdateIncident" ] = new soap_Operation( "UpdateIncident",
"Update", "document", "UpdateIncidentRequest", "UpdateIncidentResponse" );

function UpdateIncidentRequest( )
```

Or -

```
this.ProductType= new StringType();

•functionStringType( val)
```

If any of these definitions are missing, report this to customer support with an unload of the generated JavaScript, the WSDL in text format, and all imported xsd files.

Example using Keep-Alive with .Net Web Services Studio

To use Keep-Alive with .Net Web Services Studio, perform the following actions.

First set the following:

- set "AllowWriteStreamBuffering" to True
- set "BasicAuthUsername" to falcon
- set "KeepAlive" to True
- set "UserCookieContainer" to True

Execute a RetrieveIncident action and search for the incident with the number IM1001.

- When you click **Send**, the "Set-Cookie" and "Connection" headers can be seen in the response window.
- Click **Send** again, only the "Connection" header can be seen in the response.

In the sm.log, these two requests (one per send) will belong to one session, meaning they have the same Process ID (Thread ID) combination.

```
2052(  6096) 05/05/2008 15:30:31  RTE I Using "utalloc" memory manager
2052(  6096) 05/05/2008 15:30:31  RTE I Process sm 7.01.048
  System: 13080 (0x784dfb00) on PC running Windows
  XP Professional (5.1 Build 2600) from server (127.0.0.1)
2052(  6096) 05/05/2008 15:30:31  RTE I Connected to SOAP client
  at 127.0.0.1
2052(  6096) 05/05/2008 15:30:31
  RTE I Attaching to resources with key 0x784dfb00
2052(  6096) 05/05/2008 15:30:31
  RTE I Info: SQL State: 01000-5701
  Message: [Microsoft][SQL Native Client][SQL Server]Changed database
  context to 'sm701'.
2052(  6096) 05/05/2008 15:30:31  RTE I Info: SQL State: 01000-5703
```

```
    Message: [Microsoft][SQL Native Client][SQL Server]Changed language setting
    to us_english.
  2052(  6096) 05/05/2008 15:30:31  RTE I sqmssqlExec info statement= SQL CONNEC
T
  2052(  6096) 05/05/2008 15:30:31  RTE I Connection established to
    dbtype 'sqlserver' database 'sm701' user 'sm7'
  2052(  6096) 05/05/2008 15:30:31  RTE I Connected to Data source
    'sm701' SQL server 'server\SQLEXPRESS' version: 9.0.3042
     Using database 'sm701' as user 'sm7'
  2052(  6096) 05/05/2008 15:30:31  RTE I MS SQL Server collation
    'Latin1_General_BIN', varchar codepage 1252, comparison 0:
    case sensitive, accent sensitive
  2052(  6096) 05/05/2008 15:30:31  RTE I Thread
    912DAAD51D1B0A53B251147F6665B7EE initialization done.
```

First execution of .Net Web Services Studio

The following code shows an example of the first execution of code when using Keep-Alive with .Net Web Services Studio.

```
  2052(  6096) 05/05/2008 15:30:31  RTE D Parsing request document:
<?xml version="1.0" encoding="utf-8"?><soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soap:Body>
    <RetrieveIncidentRequest
      xmlns="http://servicecenter.peregrine.com/PWS">
      <model query="">
        <keys query="">
          <IncidentID/>
        </keys>
        <instance query="" recordid="" uniquequery="">
          <IncidentID>IM1001</IncidentID>
        </instance>
        <messages>
          <message xmlns="http://sc62server/PWS/Common"
            module=""/>
        </messages>
      </model>
    </RetrieveIncidentRequest>
  </soap:Body>
</soap:Envelope>
  2052(  6096) 05/05/2008 15:30:31  RTE D Done parsing request document
  2052(  6096) 05/05/2008 15:30:31  RTE D doCardinalOperation entered for
     SOA Mode 2 operation 1 - Retrieve
  2052(  6096) 05/05/2008 15:30:31  RTE D Calling loginAuthenticate
     with user=falcon and password=#########
  2052(  6096) 05/05/2008 15:30:31  RTE D Authentication succeeded
```

```
2052(  6096) 05/05/2008 15:30:31  RTE D Calling agend()
2052(  6096) 05/05/2008 15:30:31  RTE D Calling agstart()
2052(  6096) 05/05/2008 15:30:31  RTE D Calling login with user=falcon
   and password=########
2052(  6096) 05/05/2008 15:30:32  RTE I User falcon logged in.
   Already licensed
2052(  6096) 05/05/2008 15:30:32  RTE D Login succeeded
2052(  6096) 05/05/2008 15:30:32  RTE D Setting uname to falcon
2052(  6096) 05/05/2008 15:30:32  RTE D Operation will be carried out on
   file probsummary
2052(  6096) 05/05/2008 15:30:32  RTE D doQuery using query string
  number="IM1001"
2052(  6096) 05/05/2008 15:30:33  RTE D doGet query returned 1
2052(  6096) 05/05/2008 15:30:33  RTE D SOA revision time is 2005-03-15
2052(  6096) 05/05/2008 15:30:33  RTE D SOA revision level is 0
```

Second execution of .Net Web Services Studio

The following code shows an example of the second execution of code when using Keep-Alive with .Net Web Services Studio.

```
2052(  6096) 05/05/2008 15:33:40  RTE D Parsing request document:
  <?xml version="1.0" encoding="utf-8"?><soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soap:Body>
  <RetrieveIncidentRequest
    xmlns="http://servicecenter.peregrine.com/PWS">
    <model query="">
      <keys query="">
        <IncidentID/>
      </keys>
      <instance query="" recordid="" uniquequery="">
        <IncidentID>IM1001</IncidentID>
      </instance>
      <messages>
        <message xmlns="http://servicecenter.peregrine.com/PWS/Common"
          module=""/>
      </messages>
    </model>
  </RetrieveIncidentRequest>
</soap:Body>
</soap:Envelope>
2052(  6096) 05/05/2008 15:33:40  RTE D Done parsing request document
2052(  6096) 05/05/2008 15:33:40  RTE D doCardinalOperation entered for
  SOA Mode 2 operation 1 - Retrieve
2052(  6096) 05/05/2008 15:33:40  RTE D User falcon is already logged
  in for this process - skipping login processing
```

```
2052(  6096) 05/05/2008 15:33:40  RTE D Operation will be carried out
  on file probsummary
2052(  6096) 05/05/2008 15:33:40  RTE D doQuery using query string
  number="IM1001"
2052(  6096) 05/05/2008 15:33:40  RTE D doGet query returned 1
2052(  6096) 05/05/2008 15:33:41  RTE D SOA revision time is 2005-03-15
2052(  6096) 05/05/2008 15:33:41  RTE D SOA revision level is 0
```

Consuming a Service Manager Web Service

A Service Manager Web service can be consumed by a custom client or by an application that directly consumes Web Services, such as Service Manager or Connect-It.

General Information

A Web Service development tool kit that can generate a complete Web service application from a `.wsdl` file is required to create a custom client that can access the Service Manager Web service. A good understanding of Web Services and SOAP versions 1.1 or 1.2 is also recommended.

Note: Service Manager users and application designers can choose any third-party Web Services development tool kit. However, Service Manager publishes only the WSDL files for the Web Service. Troubleshooting the client application is the responsibility of the application developer, and outside the scope of Service Manager Customer Support.

Use the steps below as a guide to create your custom Web Service client.

1. Publish the Service Manager tables that you want your client to access. You can use the Service Manager Web Services out-of-the-box or customize the extaccess records to meet your needs.

2. Obtain a Web Services client development tool that can create a complete Web Service application, such as Microsoft .NET or Apache Axis, or obtain a tool that generates a complete Web Service application by evaluating the target WSDL file.

3. Browse to the URL of your Service Manager server and download the WSDL files for the services you want your custom clients to use. Use your Web Services client development tool to browse the WSDL and determine which features you want your custom client to use. The URL of your server must include the port and the Web service name.
   For example:
   http://<Service Manager server>:<httpPort (use a dedicated port, do not use loadBalancer port) >/SM/7/PWS/IncidentManagement.wsdl connects to the Service Manager server host on the specified port and requests the IncidentManagement WSDL.

4. Use your Web Services client development tool to generate the programming language client code (classes) that will invoke the Service Manager Web services. Tools such as .NET wsdl.exe or Axis wsdl2java generate client code that can be used to invoke the Service Manager Web service from the WSDL. Your custom Web Services client invokes the client code rather than the WSDL directly.

5. Write a client application in the appropriate language of your client development tool. For example, .NET requires either Microsoft Visual C# or Visual Basic®, and Axis requires Java.

Dynamic and static Web Services clients

Tools such as Visual Studio or .NET allow for simple creation of Web Service clients from a WSDL. These clients are static Web Service consumers and have to be rebuilt every time the WSDL changes. To get around the tedious work of rebuilding the client code for every WSDL change (new fields, new methods, new objects), you can create dynamic Web Services clients. These clients read the WSDL each time they use it and dynamically refer to the objects and methods within.

When an external client consumes Service Manager data, the client code can be written for dynamic or static WSDL consumption. When Service Manager consumes external data, it uses static consumption always.

What happens if an exposed table is changed?

The WSDL for a service does not change automatically as a result of making tailoring changes such as adding a new field to a table. Only if you include the new field in the Web Services API by adding it to the extaccess record will the new field be exposed.

If you change the caption (alias name) by which a field is exposed in a Web Service, you are going to have to modify and recompile any SOAP client applications which reference this field. You can rename the internal Service Manager field names, even for fields which are exposed via Web Services, without impacting deployed Web Services, as long as you do not change the alias name by which the field is known to Web Services.

Finally, if you add a new field, make the new field a required field and you have previously deployed Web Services applications which do not populate this field, you must provide tailoring in the server to generate a valid default value for the field when a value is not provided. Otherwise, inserts and updates via Web Services will fail because the new field has not been populated when the record goes through validation.

Updating Service Manager tables

By design, the Service Manager server expects that the client application will specify only those fields to be updated. It ignores missing or empty elements in the update request. If you specify a new value to update a field and that field is an array, ensure that you match the number of new values for the array elements to the number of existing array elements; otherwise, the number of elements in the array will dynamically resize to contain only the new values.

You can code a global attribute on the request element called ignoreEmptyElements and set it to true or false. If you specify ignoreEmptyElements=false, any missing or empty element in the update request causes the named field to be cleared to null values.

If you want to clear a specific field, specify `xsi:nil=true` as an element attribute.

Requirements for developing custom Web Services clients

You can create custom Web Services clients to access the HP Service Manager Web Services API. If you choose to create a custom Web Services client, ensure that you review the statement of technical support for custom Web Services clients, and that you have the following skills and tools:

- A good understanding of the W3C recommendation for SOAP version 1.1 or 1.2. Service Manager supports both versions, but recommends SOAP version 1.2.
- A Web Service development tool kit that can generate a complete Web service application from a .wsdl file.
- Familiarity with the debughttp server parameter and the HTTP.LOG it generates.

  Note: There are several Web services development tool kits that you can use to develop custom Web Services clients, such as Microsoft Visual Studio .NET™, Systinet WASP™, Glue™, Apache Axis™, or Sun Web Services Developer Pack™.

In order to support custom Web Services client connections to Service Manager you need:

- An installed Service Manager server instance (Your custom Web services clients can connect to the normal server listener port)
- A list of the Service Manager tables and actions you want to permit access to (you can grant or deny access from the extaccess table)

Checklist: Creating a custom Web Services client

You can create custom Web Services client applications to connect and conduct transactions with the HP Service Manager Web service. Any custom clients you create must be able to send and receive from the Service Manager server valid SOAP messages.

1. Publish the Service Manager tables to which you want the custom client to connect as Web Services. You can use Service Manager Web Services API out-of-the-box, or customize the Web Services to meet your business needs.
2. Obtain a Web Services client development tool that can create a complete Web service application, such as Microsoft .NET™ or Apache Axis™, or obtain a tool that generates a complete Web Service application by evaluating the target WSDL file, such as GotDotNet™ WebServiceStudio™.
3. Browse to the URL of your Service Manager server and download the WSDL files for the services you want your custom clients to use. Use your Web Services client development tool to browse the WSDL and determine which features you want your custom client to use.

   Note: The URL of your server must include the listener port and the Web service name. For

example, http://smserver:13081/IncidentManagement.wsdl connects to the smserver host on port 13081 and requests the IncidentManagement WSDL.

Important: Do not use the Load Balancer listener port for all incoming Web Services requests. Instead, dedicate one or more Service Manager server processes to serve Web Services requests by adding the "debugnode" parameter to the process you wish to dedicate to serve Web Services requests.

4. Use your Web Services client development tool to generate the programming language client code that invokes the Service Manager Web Services for the Service Manager Web Services. Tools such as .NET wsdl.exe or Axis wsdl2java can generate client code that can be used to invoke the Service Manager Web Service from the WSDL.

5. Write a client application in the appropriate language of your client development tool. For example, .NET requires Microsoft C#™ or Visual Basic™; Axis requires Java.

   Tip: The HP Service Manager installation DVD contains source code for several sample Web Services client applications you can use as templates for your own custom clients. The source code includes Axis and .NET examples.

   Note: There are many Web Service application development tools available such as Microsoft Visual Studio .NET™, Systinet WASP™, Glue™, Apache Axis™, or Sun Web Services Developer Pack™. Service Manager users and application designers can choose any third-party tool with the understanding that HP publishes only the WSDL files for the web service. Troubleshooting the client application is the responsibility of the application developer, and outside the scope of Service Manager Customer Support.

Technical support for custom Web Services clients

Custom Web Services clients and any code or scripting that you add to interface with the HP Service Manager products are outside the scope of the HP product suite and are not covered under maintenance and support contracts. Ensure that you have full access to the appropriate resources to assist you with training, debugging, and maintaining any code that you add to your Service Manager environment.

HP provides a working example database and several sample Web Services clients that can help you troubleshoot your custom clients and determine where errors occur.

Sample Web Services client for sc62server PWS URL

The HP Service Manager server includes two sample Web Services client applications for the `http://servername:port_number/sc62server/PWS/service_name.wsdl`. One was created for Apache™ Axis and the other for Microsoft™ Visual Studio .NET. If you have one of these two Web Services development tools installed, you can review and update the source code of the sample applications as well as generate updated proxy code to test the Service Manager Web Services functionality. The Apache Axis samples are written in Java while the Microsoft .NET samples are written in C#. The sample client applications are included with the server installation in the following folders:

- <Service Manager server installation folder>\webservices\sample\sc62webservices
  - AxisSample
  - DotNetSample

Each sample includes the source code for the client applications as well as support files for the Web Services development environment. The Apache Axis sample also includes a library of Axis jar files as well as batch files that you can use to run each class from the Windows command prompt after you have compiled the sample Java. You can use the sample applications as examples of how to create your own custom Web Services client applications.

Note: All the sample applications use a command line interface. To see the usage information for the command line interface, type: **dotNetSample -example ClassName** where ClassName is the name of the sample application class.

The Apache Axis sample client applications assume that you have a Service Manager server instance running from the local host. If this is not the case, you can change the server host name and port number using the sample's command line interface.

The Microsoft .Net sample client applications assume that you have a Service Manager server instance running from the local host. If this is not the case, you can change the server host name and port number using the sample's command line interface or from Visual Studio .NET's Web reference URL.

Important: To use attachments with .Net samples, you must install Microsoft Web Services Enhancements (WSE) 2.0 SP2. Be sure to select the "Visual Studio Developer" option during installation. If you add WSE2 after building the examples, you must delete the old reference files ("reference.cs" and "reference.map"), update the web references, and then rebuild the sample applications.

Each of the sample folders includes a readme file that contains valuable information about using the sample application found in each of the sample folder.

Configuration Management sample

The sample client applications contain the following classes for Configuration Management. Refer to the sample application source code for comments on the usage of each class.

Field	Description
ConfigurationManagementServiceUtility	• Provides the CreateService method to initialize an object for the service. • Provides the InitServiceAuthentication method to send the host name, communications port, operator name, and operator password with each SOAP request. • Provides the InitServiceforAttachments method to initialize the service to handle MIME attachments.
CreateContactSample	Creates a contact record with the supplied parameters.
DeleteContactSample	Deletes the contact record listed in the supplied parameters.
RetrieveContactSample	Retrieves a single contact record matching the supplied parameters.
UpdateContactSample	Updates a contact record with the supplied parameters.

Incident Management sample

The sample client applications contain the following classes for Incident Management. Refer to the sample application source code for comments on the usage of each class.

Class	Description
CloseIncidentSample	Closes an incident record with the supplied parameters.
CreateIncidentSample	Creates an incident record with the supplied parameters.
IncidentManagementServiceUtility	• Provides the CreateService method to initialize an object for the service. • Provides the InitServiceAuthentication method to send the host name, communications port, operator name, and operator password with each SOAP request. • Provides the InitServiceforAttachments method to initialize the service to handle MIME attachments.
ResolveIncidentSample	Resolves an incident record matching the supplied parameters.

Class	Description
RetrieveIncidentListSample	Retrieves multiple incident records matching the supplied parameters.
RetrieveIncidentSample	Retrieves a single incident record matching the supplied parameters.
UpdateIncidentSample	Updates an incident record with the supplied parameters.

Command line arguments for the .NET samples

The .NET sample application runs from the Windows command prompt. After you have compiled the .NET sample into an executable, you can perform configuration and incident management tasks with the following arguments.

Note: To see the usage information for the .NET sample application, type: **dotNetSample -example ClassName** where ClassName is the name of a sample application class.

The following commands invoke Configuration Management functionality.

Operation	Command-line example
Create contact	dotnetsample -example CreateContact -name sneveau -lastName Neveau -firstName Sophie -workPhone "(858) 481-5000" -extension 3573 -fullname "Sophie Neveau"
Delete contact	dotNetSample -example DeleteContact -name sneveau -lastName Neveau -firstName Sophie
Retrieve contact	dotNetSample -example RetrieveContact -name "FALCON, JENNIFER"
Update Contact	dotNetSample -example UpdateContact -name "FALCON, JENNIFER" -workPhone "(858) 481-5000" -extension 3573

The following commands invoke Incident Management functionality.

Operation	Command-line example
Close incident	dotNetSample -example CloseIncident -incidentId IM10001 -closeCode "User Closer" -resolution "Problem disappeared"
Create incident	dotNetSample -example CreateIncident -title ".NET sample brief description" -category incident -problemType "not specified" -description ".NET sample incident" -severity 1 -subCategory data -productType "storage limit exceeded" -initialImpact 1 -primaryAssignmentGroup "Operating System Support (South America)" -service "Printing (Africa)"

Operation	Command-line example
Resolve incident	dotNetSample -example ResolveIncident -incidentId IM10006 -resolution "Problem disappeared"
Retrieve incident list	dotNetSample -example RetrieveIncidentList -incidentId IM10001:IM10002
Retrieve incident	dotNetSample -example RetrieveIncident -incidentId IM10001
Update incident	dotNetSample -example UpdateIncident -incidentId IM10006 -journalUpdates "User provided more information"

Command line arguments for the Axis sample application

The Axis sample application runs from the command prompt using Java. After you have compiled the Axis sample into an executable class file, you can perform configuration and incident management tasks with the following arguments.

Note: To see the usage information for the Axis sample application, type: **ClassName** where ClassName is the name of a sample application class.

Configuration Management

The following commands invoke Configuration Management functionality. These examples assume you are using the batch files provided with the Axis sample application to automatically set the class path and call the proper executable class.

Operation	Command-line example
Create contact	CreateContactSample -name sneveau -fullname "Sophie Neveau"
Delete contact	DeleteContactSample -username falcon -name "sneveau"
Retrieve contact	RetrieveContactSample -name "FALCON, JENNIFER"
Update Contact	UpdateContactSample -name "FALCON, JENNIFER"

Incident Management

The following commands invoke Incident Management functionality. These examples assume you are using the batch files provided with the Axis sample application to automatically set the class path and call the proper executable class.

Operation	Command-line example
Close incident	CloseIncidentSample -incidentId IM10001 -closeCode "User Closer" -resolution "Problem disappeared"
Create incident	CreateIncidentSample -briefDescription "Java sample brief description" -category incident -incidentDescription "This is a description" -severity 1 -subCategory hardware -productType "missing or stolen" -initialImpact 1 -service Applications -primaryAssignmentGroup Network
Resolve incident	ResolveIncidentSample -incidentId IM10006 -resolution "Problem disappeared"
Retrieve incident list	RetrieveIncidentListSample -incidentId IM10001:IM10002
Retrieve incident	RetrieveIncidentSample -incidentId IM1001
Update incident	UpdateIncidentSample -incidentId IM10006 -journalUpdates "User provided more information"

Using query syntax

As shown in the example above, Service Manager supports queries using a special query syntax with special characters such as # ("starts with"), or relational operators such as > or < preceding an actual data value. With Web Services this syntax is available for string data as well. If the field is of a type other than string (for example an integer or dateTime type) and you are using a strongly typed programming language such as Java or C# to write your client code, you will not be able to leverage this feature, since the special characters would not be acceptable data types for these fields. To generate queries with this syntax on all types of fields, fill in the query=”xxx” section as shown below.

The request

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:pws="http://servicecenter.peregrine.com/PWS"
 xmlns:com="http://servicecenter.peregrine.com/PWS/Common">
   <soapenv:Header/>
   <soapenv:Body>
      <pws:RetrieveIncidentKeysListRequest attachmentInfo="?"
       attachmentData="?" ignoreEmptyElements="true">
         <pws:model query="">
            <pws:keys query=" update.time>'05/01/08'">
               <pws:IncidentID type="String" mandatory="?" readonly="?">
               </pws:IncidentID>
            </pws:keys>
            <pws:instance query=" " uniquequery="?" recordid="?">
               <pws:IncidentID type="String" mandatory="?" readonly="?">
```

```
</pws:IncidentID>
<pws:Category type="String" mandatory="?" readonly="?">
</pws:Category>
<pws:OpenTime type="DateTime" mandatory="?" readonly="?">
</pws:OpenTime>
<pws:OpenedBy type="String" mandatory="?" readonly="?">
</pws:OpenedBy>
<pws:severity type="String" mandatory="?" readonly="?">
</pws:severity>
<pws:UpdatedTime type="DateTime" mandatory="?"
 readonly="?"></pws:UpdatedTime>
<pws:PrimaryAssignmentGroup type="String" mandatory="?"
 readonly="?"></pws:PrimaryAssignmentGroup>
<pws:ClosedTime type="DateTime" mandatory="?"
 readonly="?"></pws:ClosedTime>
<pws:ClosedBy type="String" mandatory="?" readonly="?">
</pws:ClosedBy>
<pws:ClosureCode type="String" mandatory="?" readonly="?">
</pws:ClosureCode>
<pws:ConfigurationItem type="String" mandatory="?"
 readonly="?"></pws:ConfigurationItem>
<pws:Location type="String" mandatory="?" readonly="?">
</pws:Location>
<pws:IncidentDescription type="Array">
   <pws:IncidentDescription type="String" mandatory="?"
    readonly="?"></pws:IncidentDescription>
</pws:IncidentDescription>
<pws:Resolution type="Array">
   <pws:Resolution type="String" mandatory="?"
    readonly="?"></pws:Resolution>
</pws:Resolution>
<pws:AssigneeName type="String" mandatory="?" readonly="?">
</pws:AssigneeName>
<pws:Contact type="String" mandatory="?" readonly="?">
</pws:Contact>
 <pws:JournalUpdates type="Array">
   <pws:JournalUpdates type="String" mandatory="?"
    readonly="?"></pws:JournalUpdates>
</pws:JournalUpdates>
<pws:AlertStatus type="String" mandatory="?" readonly="?">
</pws:AlertStatus>
<pws:ContactLastName type="String" mandatory="?"
 readonly="?"></pws:ContactLastName>
<pws:ContactFirstName type="String" mandatory="?"
 readonly="?"></pws:ContactFirstName>
<pws:Company type="String" mandatory="?" readonly="?">
</pws:Company>
<pws:BriefDescription type="String" mandatory="?"
 readonly="?"></pws:BriefDescription>
```

```
            <pws:TicketOwner type="String" mandatory="?" readonly="?">
            </pws:TicketOwner>
            <pws:UpdatedBy type="String" mandatory="?" readonly="?">
            </pws:UpdatedBy>
            <pws:IMTicketStatus type="String" mandatory="?"
             readonly="?"></pws:IMTicketStatus>
            <pws:Subcategory type="String" mandatory="?" readonly="?">
            </pws:Subcategory>
            <pws:SLAAgreementID type="Decimal" mandatory="?"
             readonly="?"></pws:SLAAgreementID>
            <pws:SiteCategory type="String" mandatory="?" readonly="?">
            </pws:SiteCategory>
            <pws:ProductType type="String" mandatory="?" readonly="?">
            </pws:ProductType>
            <pws:ProblemType type="String" mandatory="?" readonly="?">
            </pws:ProblemType>
            <pws:ResolutionFixType type="String" mandatory="?"
             readonly="?"></pws:ResolutionFixType>
            <pws:UserPriority type="String" mandatory="?" readonly="?">
            </pws:UserPriority>
            <pws:Solution type="Array">
               <pws:Solution type="String" mandatory="?" readonly="?">
               </pws:Solution>
            </pws:Solution>
            <pws:InitialImpact type="String" mandatory="?"
             readonly="?"></pws:InitialImpact>
            <pws:folder type="String" mandatory="?" readonly="?">
            </pws:folder>
         </pws:instance>
         <pws:messages>
             <com:message type="String" mandatory="?"
              readonly="?" severity="?" module="?">?</com:message>
         </pws:messages>
      </pws:model>
   </pws:RetrieveIncidentKeysListRequest>
</soapenv:Body>
</soapenv:Envelope>
```

The response

```
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
     <RetrieveIncidentKeysListResponse message="Success"
      query="" returnCode="0" schemaRevisionDate="2007-04-14"
      schemaRevisionLevel="1" status="SUCCESS"
      xsi:schemaLocation="http://servicecenter.peregrine.com/PWS
      http://<sm server>.americas.hpqcorp.net:13701/sc62server/ws/
```

```
Incident.xsd"
xmlns="http://servicecenter.peregrine.com/PWS"
xmlns:cmn="http://servicecenter.peregrine.com/PWS/Common"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <keys>
     <IncidentID type="String">IM10055</IncidentID>
  </keys>
  <keys>
     <IncidentID type="String">IM10063</IncidentID>
  </keys>
  <keys>
     <IncidentID type="String">IM10070</IncidentID>
  </keys>
  <keys>
     <IncidentID type="String">IM10077</IncidentID>
  </keys>
  <keys>
     <IncidentID type="String">IM10090</IncidentID>
  </keys>
  <keys>
     <IncidentID type="String">IM10115</IncidentID>
  </keys>
  <keys>
     <IncidentID type="String">IM10116</IncidentID>
  </keys>
  <keys>
     <IncidentID type="String">IM10117</IncidentID>
  </keys>
  <keys>
     <IncidentID type="String">IM10118</IncidentID>
  </keys>
  <keys>
     <IncidentID type="String">IM10119</IncidentID>
  </keys>
  <keys>
     <IncidentID type="String">IM10120</IncidentID>
  </keys>
  <keys>
     <IncidentID type="String">IM10121</IncidentID>
  </keys>
  <keys>
     <IncidentID type="String">IM10122</IncidentID>
  </keys>
  <keys>
     <IncidentID type="String">IM10123</IncidentID>
  </keys>
  <keys>
     <IncidentID type="String">IM10124</IncidentID>
```

```
         </keys>
         <keys>
            <IncidentID type="String">IM10125</IncidentID>
         </keys>
      </RetrieveIncidentKeysListResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Note: The field names in such directly entered query strings reflect either the actual field names (such as update.time) or the Caption (such as UpdateTime). Clients who want to submit an expert/advanced query should use either the query attribute on the <keys> element or the query attribute on the <instance> element. Both are provided because some requests do not define any <instance> element. During SOAP API request processing, the server will look first at the <keys> element and, if there is no query attribute there, will look at the <instance> element. Query attributes defined on any other element are never consulted during inbound SOAP request processing.

Retrieving data from Service Manager

Retrieval methods are not defined in the extaccess record. The following list shows the methods for retrieval that are available and under which circumstances to use each one:

• `Retrieve<FileName>` — Used if only one record will be returned. Throws a fault if multiple records are returned.

• `Retrieve<FileName>KeysList` — Retrieves the list of unique keys (which does not have to be the unique key of the Service Manager dbdicts). The list can either be passed as an array to the `Retrieve<FileName>List` method, or looped through to pass to the `Retrieve<FileName>` method.

• `Retrieve<FileName>List` — Retrieves a list of records with information that was gathered either in the `Retrieve<FileName>KeysList` method or by passing in a query directly through the instance block. This method expects an array of keys unless the query approach is used.

Note: When retrieving data from a single table rather than a Service such as the contacts table, request the WSDL for the alias name defined in `extaccess`, such as "Contact" (singular form, upper-case “C”) rather than for `contacts` (the actual file name).

There are different approaches to retrieving a list of records. When developing a custom client there are actually two separate methods that can be used to retrieve list data.

The first approach uses the following steps:

1. Send the data query (such as `<open.time>&gt;6/30/05</open.time>`) to the `RetrieveKeysList` method.

2. The result is a list of records where each record contains only the “primary key” (such as Incident ID) for those records that match the query.

3. You can either provide the list to the `RetrieveList` method and receive all records defined by the list in a single XML document, or loop through the list, one record at a time, calling `Retrieve` once for each record by key.

The second approach uses these steps:

1. Send the data query (such as <open.time>>6/30/05</open.time>) directly to the RetrieveList method. Place the query in the “<instance>” block instead of the “<keys>” block.

2. This single method call returns the entire result set (all fields for all records matching the query) in a single XML response.

Note: The second approach returns the entire query result set in one method call. If the result set is large, use the first approach to increase performance.

Example: Retreiving data from Service Manager via a Web service

The simplest way to perform retrieval operations is via Query-by-example (QBE). This is done by creating an instance of a particular kind of object (such as an Incident) and populating one or more fields with values to determine the result set. You have to only supply values for the fields on which you wish to select.

The instance is then passed into a RetrieveXXXKeysList request. In a program, such as the sample programs provided with Service Manager, you would be assigning values to properties or calling setter methods on various Java or C# or other objects. In the following example, we submit a RetrieveIncidentKeysList object, supplying a value for OpenedBy and UpdatedBy. In this example, we will use Service Manager query syntax to find all incidents where the OpenedBy element starts with “fal” as well as pass a literal value for the UpdatedBy field.

We get back a RetrieveIncidentKeysListResponse object listing the primary keys of the matching Incident objects.

Combining multiple values in this QBE style selection connects the query terms with AND. To create a query using OR, supply an expert query as a string in the <pws:instance query="xxx" …> area.

The request

```
<?xml version="1.0" encoding="UTF-8" ?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:pws="http://servicecenter.peregrine.com/PWS"
xmlns:com="http://servicecenter.peregrine.com/PWS/Common">
   <soapenv:Header/>
   <soapenv:Body>
      <pws:RetrieveIncidentKeysListRequest attachmentInfo="?"
      attachmentData="?" ignoreEmptyElements="true">
         <pws:model query="">
            <pws:keys query="">
               <pws:IncidentID type="String" mandatory="?" readonly="?">
               </pws:IncidentID>
            </pws:keys>
            <pws:instance query="" uniquequery="?" recordid="?">
               <pws:IncidentID type="String" mandatory="?" readonly="?">
               </pws:IncidentID>
               <pws:Category type="String" mandatory="?" readonly="?">
```

```
</pws:Category>
<pws:OpenTime type="DateTime" mandatory="?" readonly="?">
</pws:OpenTime>
<pws:OpenedBy type="String" mandatory="?" readonly="?">#fal
</pws:OpenedBy>
<pws:severity type="String" mandatory="?" readonly="?">
</pws:severity>
<pws:UpdatedTime type="DateTime" mandatory="?"
readonly="?"></pws:UpdatedTime>
<pws:PrimaryAssignmentGroup type="String" mandatory="?"
readonly="?"></pws:PrimaryAssignmentGroup>
<pws:ClosedTime type="DateTime" mandatory="?" readonly="?">
</pws:ClosedTime>
<pws:ClosedBy type="String" mandatory="?" readonly="?">
</pws:ClosedBy>
<pws:ClosureCode type="String" mandatory="?" readonly="?">
</pws:ClosureCode>
<pws:ConfigurationItem type="String" mandatory="?"
readonly="?"></pws:ConfigurationItem>
<pws:Location type="String" mandatory="?" readonly="?">
</pws:Location>
<pws:IncidentDescription type="Array">
   <pws:IncidentDescription type="String" mandatory="?"
   readonly="?"></pws:IncidentDescription>
</pws:IncidentDescription>
<pws:Resolution type="Array">
   <pws:Resolution type="String" mandatory="?"
   readonly="?"></pws:Resolution>
</pws:Resolution>
<pws:AssigneeName type="String" mandatory="?" readonly="?">
</pws:AssigneeName>
<pws:Contact type="String" mandatory="?" readonly="?">
</pws:Contact>
 <pws:JournalUpdates type="Array">
   <pws:JournalUpdates type="String" mandatory="?"
   readonly="?"></pws:JournalUpdates>
</pws:JournalUpdates>
<pws:AlertStatus type="String" mandatory="?" readonly="?">
</pws:AlertStatus>
<pws:ContactLastName type="String" mandatory="?"
readonly="?"></pws:ContactLastName>
<pws:ContactFirstName type="String" mandatory="?"
readonly="?"></pws:ContactFirstName>
<pws:Company type="String" mandatory="?" readonly="?">
</pws:Company>
<pws:BriefDescription type="String" mandatory="?"
readonly="?"></pws:BriefDescription>
<pws:TicketOwner type="String" mandatory="?" readonly="?">
</pws:TicketOwner>
```

```
            <pws:UpdatedBy type="String" mandatory="?"
            readonly="?">falcon</pws:UpdatedBy>
            <pws:IMTicketStatus type="String" mandatory="?"
            readonly="?"></pws:IMTicketStatus>
            <pws:Subcategory type="String" mandatory="?" readonly="?">
            </pws:Subcategory>
            <pws:SLAAgreementID type="Decimal" mandatory="?"
            readonly="?"></pws:SLAAgreementID>
            <pws:SiteCategory type="String" mandatory="?" readonly="?">
            </pws:SiteCategory>
            <pws:ProductType type="String" mandatory="?" readonly="?">
            </pws:ProductType>
            <pws:ProblemType type="String" mandatory="?" readonly="?">
            </pws:ProblemType>
            <pws:ResolutionFixType type="String" mandatory="?"
            readonly="?"></pws:ResolutionFixType>
            <pws:UserPriority type="String" mandatory="?" readonly="?">
            </pws:UserPriority>
            <pws:Solution type="Array">
               <pws:Solution type="String" mandatory="?" readonly="?">
               </pws:Solution>
            </pws:Solution>
            <pws:InitialImpact type="String" mandatory="?"
           readonly="?"></pws:InitialImpact>
            <pws:folder type="String" mandatory="?" readonly="?">
            </pws:folder>
         </pws:instance>
         <pws:messages>
             <com:message type="String" mandatory="?" readonly="?"
             severity="?" module="?">?</com:message>
         </pws:messages>
      </pws:model>
   </pws:RetrieveIncidentKeysListRequest>
</soapenv:Envelope>
```

The response

```
<?xml version="1.0" encoding="utf-16"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
      <RetrieveIncidentKeysListResponse message="Success" query=""
       returnCode="0" schemaRevisionDate="2007-04-14"
       schemaRevisionLevel="1" status="SUCCESS"
       xsi:schemaLocation="http://servicecenter.peregrine.com/PWS
       http://<sm server>.americas.hpqcorp.net:13701/sc62server/ws/
       Incident.xsd" xmlns="http://servicecenter.peregrine.com/PWS"
       xmlns:cmn="http://servicecenter.peregrine.com/PWS/Common"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

```
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
          <keys>
             <IncidentID type="String">IM10001</IncidentID>
          </keys>
          <keys>
             <IncidentID type="String">IM10004</IncidentID>
          </keys>
          <keys>
             <IncidentID type="String">IM10009</IncidentID>
          </keys>
          <keys>
             <IncidentID type="String">IM10016</IncidentID>
          </keys>
          <keys>
             <IncidentID type="String">IM10027</IncidentID>
          </keys>
          <keys>
             <IncidentID type="String">IM10038</IncidentID>
          </keys>
          <keys>
             <IncidentID type="String">IM10049</IncidentID>
          </keys>
          <keys>
             <IncidentID type="String">IM10060</IncidentID>
          </keys>
          <keys>
             <IncidentID type="String">IM10061</IncidentID>
          </keys>
       </RetrieveIncidentKeysListResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Having retrieved a list of <keys> elements we can now retrieve these Incidents using a RetrieveIncidentList request, by supplying the collection of keys elements in that request.

You can submit a variable number of <keys> elements in a RetrieveXXXList request, subject only to your program's ability to handle large XML responses. Java client programs can sometimes run out of memory if the server returns very large responses.

To help prevent the RetrieveXXXListRequest Web Service from causing a Java Heap Space out-of-memory error when retrieving a list of records, the following applies for the response of a RetrieveXXXList request:

- When there is neither a start attribute nor a count attribute, return all records/keys.
- When there is a valid start value but no count attribute, return all records starting from the start attribute.

- When there is a valid start attribute and valid count attribute, return the number of keys/records starting from the start attribute.
- When there is a negative start attribute, return from the first record.
- When there is a negative count attribute, return one record.
- When the start attribute is bigger than the total number of records/keys, no record is returned.

Retrieve data from Service Manager using Pagination

Retrieving a list of data objects may result in a very large XML response document that could cause performance and memory utilization issues either on the client or the server. To avoid these problems, Service Manager supports the use of pagination, the process of returning pages of data instead of one large response.

To enable pagination, a Web service request can make use of these additional attributes:

- count – number of records/keys to return. By default all records are returned. This attribute indicates you want to use pagination. (Optional attribute)
- start – the starting record/key number. By default a retrieve request will start at record 0. (Optional attribute)
- handle – a record-list handle returned on a previous retrieve request that specified a count.

When pagination is used, the Web service does the following:

- If there is not a 'start' or 'count' attribute, the service returns all records or keys.
- With a valid 'start' value and 'no count' attribute, the service returns all records starting from the 'start' attribute.
- With a valid 'start' attribute and valid 'count' attribute, the service returns the number of keys or records starting from the 'start' attribute.
- With a negative 'start' attribute, the service returns records from the first record and creates a warning in log file.
- With a negative 'count' attribute, the service returns one record and creates a warning message in the log file.
- With the 'start' attribute greater than the total number of records or keys, the service returns no records and creates a message in the error log file.

Example: Use Web Service with pagination to retrieve data from Service Manager

In the previous example the Web Service returned a list of eight Incident keys. To illustrate pagination, this example limits the number of records to returned to four. This requires the use of the additional attributes count and start.

For example to limit the response to a maximum of eight records starting at the first record:

Request with pagination

```
<?xml version="1.0" encoding="UTF-8" ?><soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:pws="http://servicecenter.peregrine.com/PWS"
xmlns:com="http://servicecenter.peregrine.com/PWS/Common">
    <soapenv:Header/>
     <soapenv:Body>
        <pws:RetrieveIncidentKeysListRequest attachmentInfo="?"
          attachmentData="?" ignoreEmptyElements="true" count=8  start=0>
           <pws:model query="">
           <pws:keys query="">
           <pws:IncidentID type="String" mandatory="?"
             readonly="?">
           </pws:IncidentID>
           </pws:keys>
        <pws:instance query="" uniquequery="?" recordid="?">
        <pws:IncidentID type="String" mandatory="?"
          readonly="?">
        </pws:IncidentID>
        <pws:Category type="String" mandatory="?" readonly="?">
        </pws:Category>
        <pws:OpenTime type="DateTime" mandatory="?"
        readonly="?">
        </pws:OpenTime>
        <pws:OpenedBy type="String" mandatory="?"
          readonly="?">#fal
        </pws:OpenedBy>
        <pws:severity type="String" mandatory="?" readonly="?">
        </pws:severity>
        <pws:UpdatedTime type="DateTime" mandatory="?"
           readonly="?"></pws:UpdatedTime>
        <pws:PrimaryAssignmentGroup type="String" mandatory="?"
        readonly="?"></pws:PrimaryAssignmentGroup>
        <pws:ClosedTime type="DateTime" mandatory="?"
        readonly="?">
        </pws:ClosedTime>
        <pws:ClosedBy type="String" mandatory="?" readonly="?">
```

```
</pws:ClosedBy>
<pws:ClosureCode type="String" mandatory="?"
readonly="?">
</pws:ClosureCode>
<pws:ConfigurationItem type="String" mandatory="?"
  readonly="?"></pws:ConfigurationItem>
<pws:Location type="String" mandatory="?" readonly="?">
</pws:Location>
<pws:IncidentDescription type="Array">
<pws:IncidentDescription type="String" mandatory="?"
readonly="?"></pws:IncidentDescription>
</pws:IncidentDescription>
<pws:Resolution type="Array">
<pws:Resolution type="String" mandatory="?"
  readonly="?"></pws:Resolution>
</pws:Resolution>
<pws:AssigneeName type="String" mandatory="?"
readonly="?">
</pws:AssigneeName>
<pws:Contact type="String" mandatory="?" readonly="?">
</pws:Contact>
<pws:JournalUpdates type="Array">
<pws:JournalUpdates type="String" mandatory="?"
  readonly="?"></pws:JournalUpdates>
</pws:JournalUpdates>
<pws:AlertStatus type="String" mandatory="?"
readonly="?">
</pws:AlertStatus>
<pws:ContactLastName type="String" mandatory="?"
  readonly="?"></pws:ContactLastName>
<pws:ContactFirstName type="String" mandatory="?"
  readonly="?"></pws:ContactFirstName>
<pws:Company type="String" mandatory="?" readonly="?">
</pws:Company>
<pws:BriefDescription type="String" mandatory="?"
readonly="?"></pws:BriefDescription>
<pws:TicketOwner type="String" mandatory="?"
  readonly="?">
</pws:TicketOwner>
<pws:UpdatedBy type="String" mandatory="?"
  readonly="?">falcon</pws:UpdatedBy>
<pws:IMTicketStatus type="String" mandatory="?"
readonly="?"></pws:IMTicketStatus>
<pws:Subcategory type="String" mandatory="?"
readonly="?">
</pws:Subcategory>
<pws:SLAAgreementID type="Decimal" mandatory="?"
readonly="?"></pws:SLAAgreementID>
<pws:SiteCategory type="String" mandatory="?"
```

```
readonly="?">
</pws:SiteCategory>
<pws:ProductType type="String" mandatory="?"
  readonly="?">
</pws:ProductType>
<pws:ProblemType type="String" mandatory="?"
readonly="?">
</pws:ProblemType>
<pws:ResolutionFixType type="String" mandatory="?"
  readonly="?"></pws:ResolutionFixType>
<pws:UserPriority type="String" mandatory="?"
  readonly="?">
</pws:UserPriority>
<pws:Solution type="Array">
<pws:Solution type="String" mandatory="?"
readonly="?">
</pws:Solution>
</pws:Solution>
<pws:InitialImpact type="String" mandatory="?"
readonly="?"></pws:InitialImpact>
<pws:folder type="String" mandatory="?" readonly="?">
</pws:folder>
</pws:instance>
<pws:messages>
  <com:message type="String" mandatory="?" readonly="?"
   severity="?" module="?">?</com:message>
</pws:messages>
</pws:model>
</pws:RetrieveIncidentKeysListRequest>
</soapenv:Envelope>
```

Response with pagination

The response indicates that eight records were returned (count=8). In addition the response indicates that there are more records to retrieve (more=1).

```
<?xml version="1.0" encoding="utf-16"?>
<SOAP-ENV:Envelope xmlns:SOAPENV="
   http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
      <RetrieveIncidentKeysListResponse count=8 more=1 handle=probsummary4d67db4
80000c0082000f1a0 message="Success" query=""
       returnCode="0" schemaRevisionDate="2007-04-14"
       schemaRevisionLevel="1" status="SUCCESS"
       xsi:schemaLocation="http://servicecenter.peregrine.com/PWS
       http://<sm server>.americas.hpqcorp.net:13701/sc62server/ws/
                Incident.xsd" xmlns="http://servicecenter.peregrine.com/PWS"
```

```
xmlns:cmn="http://servicecenter.peregrine.com/PWS/Common"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
     <keys>
         <IncidentID type="String">IM10001</IncidentID>
     </keys>
         <keys>
           <IncidentID type="String">IM10004</IncidentID>
         </keys>
         <keys>
         <IncidentID type="String">IM10009</IncidentID>
         </keys>
         <keys>
           <IncidentID type="String">IM10016</IncidentID>
         </keys>
         <keys>
           <IncidentID type="String">IM10027</IncidentID>
         </keys>
         <keys>
           <IncidentID type="String">IM10038</IncidentID>
         </keys>
         <keys>
           <IncidentID type="String">IM10049</IncidentID>
         </keys>
         <keys>
           <IncidentID type="String">IM10060</IncidentID>
         </keys>
      </RetrieveIncidentKeysListResponse>
      </SOAP-ENV:Body>
 </SOAP-ENV:Envelope>
```

Next pagination request

To retrieve the next page of records the request would increment the start attribute by the previous count and supply the handle returned on the previous response.

This request is for the next set of records.

```
<?xml version="1.0" encoding="UTF-8" ?>
<soapenv:Envelope
 xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:pws="http://servicecenter.peregrine.com/PWS"
 xmlns:com="http://servicecenter.peregrine.com/PWS/Common">
 <soapenv:Header/>
 <soapenv:Body>
    <pws:RetrieveIncidentKeysListRequest attachmentInfo="?"
```

```
            attachmentData="?" ignoreEmptyElements="true" count=8  start=8 handl
e= probsummary4d67db480000c0082000f1a0>
    <pws:model query="">
    <pws:keys query="">
    <pws:IncidentID type="String" mandatory="?"
            readonly="?">
    </pws:IncidentID>
    </pws:keys>
    <pws:instance query="" uniquequery="?" recordid="?">
    <pws:IncidentID type="String" mandatory="?"
            readonly="?">
    </pws:IncidentID>
    <pws:Category type="String" mandatory="?" readonly="?">
    </pws:Category>
    <pws:OpenTime type="DateTime" mandatory="?"
            readonly="?">
    </pws:OpenTime>
    <pws:OpenedBy type="String" mandatory="?"
            readonly="?">#fal
    </pws:OpenedBy>
    <pws:severity type="String" mandatory="?" readonly="?">
    </pws:severity>
    <pws:UpdatedTime type="DateTime" mandatory="?"
            readonly="?"></pws:UpdatedTime>
    <pws:PrimaryAssignmentGroup type="String" mandatory="?"
            readonly="?"></pws:PrimaryAssignmentGroup>
    <pws:ClosedTime type="DateTime" mandatory="?"
            readonly="?">
    </pws:ClosedTime>
    <pws:ClosedBy type="String" mandatory="?" readonly="?">
    </pws:ClosedBy>
    <pws:ClosureCode type="String" mandatory="?"
            readonly="?">
    </pws:ClosureCode>
    <pws:ConfigurationItem type="String" mandatory="?"
            readonly="?"></pws:ConfigurationItem>
    <pws:Location type="String" mandatory="?" readonly="?">
    </pws:Location>
    <pws:IncidentDescription type="Array">
    <pws:IncidentDescription type="String" mandatory="?"
            readonly="?"></pws:IncidentDescription>
    </pws:IncidentDescription>
    <pws:Resolution type="Array">
    <pws:Resolution type="String" mandatory="?"
            readonly="?"></pws:Resolution>
    </pws:Resolution>
    <pws:AssigneeName type="String" mandatory="?"
            readonly="?">
    </pws:AssigneeName>
```

```
<pws:Contact type="String" mandatory="?" readonly="?">
</pws:Contact>
<pws:JournalUpdates type="Array">
<pws:JournalUpdates type="String" mandatory="?"
        readonly="?"></pws:JournalUpdates>
</pws:JournalUpdates>
<pws:AlertStatus type="String" mandatory="?"
        readonly="?">
</pws:AlertStatus>
<pws:ContactLastName type="String" mandatory="?"
        readonly="?"></pws:ContactLastName>
<pws:ContactFirstName type="String" mandatory="?"
        readonly="?"></pws:ContactFirstName>
<pws:Company type="String" mandatory="?" readonly="?">
</pws:Company>
<pws:BriefDescription type="String" mandatory="?"
        readonly="?"></pws:BriefDescription>
<pws:TicketOwner type="String" mandatory="?"
        readonly="?">
</pws:TicketOwner>
<pws:UpdatedBy type="String" mandatory="?"
        readonly="?">falcon</pws:UpdatedBy>
<pws:IMTicketStatus type="String" mandatory="?"
        readonly="?"></pws:IMTicketStatus>
<pws:Subcategory type="String" mandatory="?"
        readonly="?">
</pws:Subcategory>
<pws:SLAAgreementID type="Decimal" mandatory="?"
        readonly="?"></pws:SLAAgreementID>
<pws:SiteCategory type="String" mandatory="?"
        readonly="?">
</pws:SiteCategory>
<pws:ProductType type="String" mandatory="?"
        readonly="?">
</pws:ProductType>
<pws:ProblemType type="String" mandatory="?"
        readonly="?">
</pws:ProblemType>
<pws:ResolutionFixType type="String" mandatory="?"
        readonly="?"></pws:ResolutionFixType>
<pws:UserPriority type="String" mandatory="?"
        readonly="?">
</pws:UserPriority>
<pws:Solution type="Array">
<pws:Solution type="String" mandatory="?"
        readonly="?">
</pws:Solution>
</pws:Solution>
<pws:InitialImpact type="String" mandatory="?"
```

```
            readonly="?"></pws:InitialImpact>
    <pws:folder type="String" mandatory="?" readonly="?">
    </pws:folder>
    </pws:instance>
    <pws:messages>
    <com:message type="String" mandatory="?" readonly="?"
            severity="?" module="?">?</com:message>
    </pws:messages>
   </pws:model>
   </pws:RetrieveIncidentKeysListRequest>
</soapenv:Envelope>
```

Next pagination response

The response indicates that one record was returned (count=1). In addition, the response indicates that there are no more records to retrieve (more=0).

```
<?xml version="1.0" encoding="utf-16"?>
<SOAP-ENV:Envelope xmlns:SOAPENV="
http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
    <RetrieveIncidentKeysListResponse count=1 more=0 handle=probsummary4d67db480
000c0082000f1a0 message="Success" query=""
        returnCode="0" schemaRevisionDate="2007-04-14"
        schemaRevisionLevel="1" status="SUCCESS"
        xsi:schemaLocation="http://servicecenter.peregrine.com/PWS
        http://<sm server>.americas.hpqcorp.net:13701/sc62server/ws/
        Incident.xsd" xmlns="http://servicecenter.peregrine.com/PWS"
        xmlns:cmn="http://servicecenter.peregrine.com/PWS/Common"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <keys>
           <IncidentID type="String">IM10061</IncidentID>
        </keys>
     </RetrieveIncidentKeysListResponse>
 </SOAP-ENV:Body>
 </SOAP-ENV:Envelope>
```

Retrieve data from Service Manager for Optimistic Locking

You can retrieve data via a Web service for later use in an update using optimistic locking by using an additional attribute:

- updatecounter – a boolean value specifying if the response should include the attribute updatecounter that can be used on a subsequent update request to optimistically lock the record. The value returned in the updatecounter attribute can be specified on a subsequent update request as the value of the updatecontraint attribute. If the value of the updateconstraint matches the value in the database the update is allowed. If the updateconstraint value does not match the value in the database the update is rejected with a return code of 51 and the message "Record modified since last retrieved"

For example to retrieve the data with the updatecounter.

Request with updatecounter

```
<?xml version="1.0" encoding="UTF-8" ?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:pws="http://servicecenter.peregrine.com/PWS"
xmlns:com="http://servicecenter.peregrine.com/PWS/Common">
<soapenv:Header/>
<soapenv:Body>
<pws:RetrieveIncidentKeysListRequest attachmentInfo="?"
attachmentData="?" ignoreEmptyElements="true">
<pws:model query="" updatecounter=true>
…
…
…
```

Response with updatecounter

The response indicates the updatecounter for each record returned.

```
<?xml version="1.0" encoding="utf-16"?>
<SOAP-ENV:Envelope xmlns:SOAPENV="
http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<RetrieveIncidentKeysListResponse count=8 more=1
```

```
handle=probsummary4d67db480000c0082000f1a0 message="Success" query=""
returnCode="0" schemaRevisionDate="2007-04-14"
schemaRevisionLevel="1" status="SUCCESS"
xsi:schemaLocation="http://servicecenter.peregrine.com/PWS
http://<sm server>.americas.hpqcorp.net:13701/sc62server/ws/
Incident.xsd"
xmlns="http://servicecenter.peregrine.com/PWS"
xmlns:cmn="http://servicecenter.peregrine.com/PWS/Common"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<keys updatecounter=”7”>
<IncidentID type="String">IM10001</IncidentID>
</keys>
<keys updatecounter =”45”>
<IncidentID type="String">IM10004</IncidentID>
</keys>
<keys updatecounter =”9”>
<IncidentID type="String">IM10009</IncidentID>
</keys>
<keys updatecounter =”13”>
<IncidentID type="String">IM10016</IncidentID>
</keys>
…
…
…
```

Web Services examples in the RUN directory

All valid Web Services examples for Axis and .NET are contained in the <sm install>\\webservices\sample directory.

Both the AxisSample and the DotNetSample directories contain documents with setup instructions. In AxisSample this document is readme.txt. In DotNetSample the document is WebServices README.doc.

The AxisSample\bin directory contains a selection of batch files that can be run directly, if JDK 1.4 or higher and Apache Ant are both installed on the machine. The DotNet samples have to be compiled before running.

Note: Axis 1.x defaults to using its own httpSender class which is not compatible with HTTP 1.1 Keep-Alive. Axis 1.x must be configured to use the commons-httpclient jar file in order to get keep-alive behavior. Running the Web Service without Keep-Alive negatively impacts performance. The Web Service client needs to be Cookie aware so that the servlet container session is maintained. HP strongly recommend using Axis 2 that provides HTTP 1.1 keep-alive support and cookie-aware behavior.

Example: Retrieving Service Manager Release Management changes into a text file using Connect-It

1. Create the Service Manager Web Services Connector with the following connection parameter settings. (All other settings remain the defaults.)

   • **Server name:** <server>:<httpPort (use a dedicated port, not the loadBalancer port)>

   • **Context Path:** sc62server/PWS

   • **Service name:** Change Management

   • Enter the SysAdmin userID and password

2. Click **Test** to verify that the connection works.

3. Click **Finish** to save the changes made to the new connector.

4. In our example, the Web Service simply fills information into a delimited text field. The settings for that connector are as follows:

   • **Name:** Changes

   • **Processing Mode:** write

   • **Connection protocol:** Local / network files

   • Enter a folder name and decide whether to create a separate file for each record retrieved or write all records into one file (recommended).

   • On the next screen, decide whether to append to the same file (recommended) or overwrite with each run, or how many files to keep.

   • Enter the path to the descriptor file (see below) or create a new descriptor file.

5. Click **Finish** to create and save the connector.

   To create a description file for the text output file (comma delimited text in our example), the following code is a sample .dsc file that can be used for retrieving change information:

```
{ TextFileFormat SMChange
       Extension=
```

```
        FormatType=Delimited
        EscapeChar="\\"
        Quote="\""
        Extracolumn=1
        WriteColumn=1
        Delimiter=,
        { String "Change Number"
                UserType=Default
        }
        { String "Category"
                UserType=Default
        }
        { String "Status"
                UserType= Default
        }
        { String "Approval Status"
                UserType=Default
        }
        { String "Requested By"
                UserType=Default
        }
        { String "Assigned To"
                UserType= Default
        }
        { String "Coordinator"
                UserType=Default
        }
        { String "Coordinator Phone"
                UserType=Default
        }
        { TimeStamp "Planned Start Date"
                UserType=TimeStamp
        }
        { TimeStamp "Planned End Date"
                UserType=TimeStamp
        }
        { String "Reason"
                UserType=Default
        }
        { String "Current Phase"
                UserType= Default
        }
        { String "Risk Assessment"
                UserType=Default
        }
}
```

6. Finally the source (Web Services) data and the target data (Delimited Text file) must be mapped, in this case based on the matching names.

7. Because Web Services need to be prompted to produce output, another text file connector must be created that helps create the request sent to the Web service. This text file connector is defined as follows:

   • **Processing Mode:** read

   • **Connection Protocol:** Local/Network files

   • **Location:** Read files, file name: <path and filename>

   • Upon successful processing, leave the file in the folder.

   Use this .dsc file. Enter <path and filename> and click Find (magnifying glass) to create or modify a description file.

8. Select a document type. Click the down arrow and enter SMChange.

9. Click **Next**.

10. Select a file for the preview. Accept the default <path and filename from step 6> and click **Next**.

11. Select the appropriate delimiter.

12. Enter the information on the screen:
    - Write the column headers: checked
    - Do not generate errors if a line contains... : checked
    - Number of skipped lines: 0
    - Quote character: "
    - Start of the comment line: //
    - Escape character: /

13. Click **Next**.

14. Enter the column names and type: for example, Change Number - text

15. Click **Finish** to create and save the connector.

16. Now create a mapping between this text file and the Service Manager Web Service by connecting the two.

17. Click the first text connector and click **Produce Now** (the F5 button) to fill information from the cm3r file in Service Manager into a delimited text file via Web Services.

Example: Getting change information from another Service Manager system

This example retrieves change information from the cm3r file and uses the Web Service to create a new change.

1. Click **Tailoring> Web Services >Run WSDL to JS**.

2. In the “Please enter the URL for the WSDL file” field, enter the following:
   http://<serenade>:<port number>/sc62server/PWS/ChangeManagement.wsdl

3. Click **Proceed** and then **Add** to create the ChangeManagement JavaScript record in the ScriptLibrary.

   Note: Service Manager always uses the user name and password you provided to access the remote Web Service unless you override the values at run time. For example, create custom JavaScript to use the currently logged in user's credentials instead of the user name and password you provided to access the remote Web Service.

```
function ChangeManagement( )
{
this.location = new String( "http://hostname:13080/sc62server/ws" );

this.user = null;
this.password = null;
this.connectTimeOut = 10;
this.sendTimeOut = 10;
this.recvTimeOut = 10;
this.soapEnvelope = null;
this.soapBody = null;
this.soapHeader = null;
this.attachments = new Array();
this.resultXML = null;
this.invoke = invoke;
[…]
```

   Note: The namespace specified in “this.location” does not have to be resolvable.

   The user can be filled in here (this.user), or in the invoking script. As a best practice, fill in the user in the invoking script.

4. Now that the Script has been added to the ScriptLibrary, you need to write another JavaScript that can be used to retrieve the change or create a new change. Enter the JavaScript code as listed below:

```
function RetrieveChangeKeysList(query)
{
try
{
var ChangeMgmtService=new system.library.ChangeManagement.
ChangeManagement();

////////////////////////////////////////////////////////
//             set Connection information (optional)
////////////////////////////////////////////////////////
ChangeMgmtService.user = "falcon";
ChangeMgmtService.password = "";


////////////////////////////////////////////////////////
//     create the request object
////////////////////////////////////////////////////////
var RetrieveChangeListRequest = new system.library.
ChangeManagement.RetrieveChangeKeysListRequest();


////////////////////////////////////////////////////////
//             Request Data Fill Section
////////////////////////////////////////////////////////////////////////

if (query!=null)
{
RetrieveChangeListRequest.model.instance.query=query;
}
else
print("Please enter a valid Query statement.")


////////////////////////////////////////////////////////////////////////
//     Invoke and final processing
////////////////////////////////////////////////////////////////////////
var RetrieveChangesResponse = ChangeMgmtService.invoke( RetrieveChangeListReq
uest );

if ( RetrieveChangesResponse.isFault() )
{
print( RetrieveChangesResponse.messages.getContent() );
return(-1);
}
else
{
print("Success")
```

```
print(RetrieveChangesResponse.keys[0].ChangeNumber.getValue())
print(RetrieveChangesResponse.keys[1].ChangeNumber.getValue())
return(RetrieveChangesResponse.keys);
}

}
catch( e )
{
print( e );
}

}

function RetrieveChange(number)
{
try
{
var ChangeMgmtService=new system.library.ChangeManagement.ChangeManagement();

////////////////////////////////////////////////////////////////////////
//             set Connection information (optional)
////////////////////////////////////////////////////////////////////////
ChangeMgmtService.user = "falcon";
ChangeMgmtService.password = "";


////////////////////////////////////////////////////////////////////////
//             create the request object
////////////////////////////////////////////////////////////////////////
var RetrieveChangeRequest = new system.library.ChangeManagement.
RetrieveChangeRequest();

////////////////////////////////////////////////////////////////////////
//     Request Data Fill Section
////////////////////////////////////////////////////////////////////////

if (number!=null)
{
RetrieveChangeRequest.model.instance.header.ChangeNumber.setValue(number);
}
else
print("Please pass in a valid Change Number.");

////////////////////////////////////////////////////////////////////////
//     Invoke and final processing
////////////////////////////////////////////////////////////////////////
var RetrieveChangeResponse = ChangeMgmtService.invoke( RetrieveChangeRequest
```

```
);

if ( RetrieveChangeResponse.isFault() )
{
print( RetrieveChangeResponse.messages.getContent() );
return(-1);
}
else
{
print("Success")
return(RetrieveChangeResponse.model.instance);
}

}
catch( e )
{
print( e );
 }

}
```

5. Create a JavaScript in the same ScriptLibrary record that reads a change record and then creates a new record in the other Service Manager system. Enter the Java Script shown below:

```
        function CreateChangeFromChange(change)
{
       try
       {
               var ChangeMgmtService=new system.library.ChangeManagement.
                ChangeManagement();

////////////////////////////////////////////////////////////////////////
//             set Connection information (optional)
////////////////////////////////////////////////////////////////////////
               ChangeMgmtService.user = "falcon";
               ChangeMgmtService.password = "";

////////////////////////////////////////////////////////////////////////
//             create the request object
////////////////////////////////////////////////////////////////////////
               var CreateChangeRequest = new system.library.ChangeManagement.
                CreateChangeRequest();
```

```
////////////////////////////////////////////////////////////////////
//     Request Data Fill Section
////////////////////////////////////////////////////////////////////

               if (change!=null)
               {
                       CreateChangeRequest.model.instance.header.Category.

                        setValue(change.header.Category.getValue());
                       CreateChangeRequest.model.instance.header.RequestedBy.

                        setValue(change.header.RequestedBy.getValue());
                       CreateChangeRequest.model.instance.header.Reason.

                        setValue(change.header.Reason.getValue());
                       CreateChangeRequest.model.instance.header.Coordinator.

                        setValue(change.header.Coordinator.getValue());
                       CreateChangeRequest.model.instance.InitialAssessment.

                        setValue(change.InitialAssessment.getValue());
                       CreateChangeRequest.model.instance.Urgency.

                        setValue(change.Urgency.getValue());
                       CreateChangeRequest.model.instance.ReleaseType.

                        setValue(change.ReleaseType.getValue());


                       for (var i=0; i<change.description_structure.

                        Description.Description.length; i++)
                       {
               CreateChangeRequest.model.instance.description_structure.

                        Description.Description_newInstance().

                        setValue(change.description_structure.Description.

                        Description[i].getValue());
                       }
               }
               else
                       print("Please pass in a valid Change Request.");


////////////////////////////////////////////////////////////////////
//     Invoke and final processing
////////////////////////////////////////////////////////////////////
               var CreateChangeResponse = ChangeMgmtService.

                invoke( CreateChangeRequest );
```

```
                if ( CreateChangeResponse.isFault() )
                {
            print( CreateChangeResponse.messages.getContent() );
            return(-1);
                }
                else
                {
                        print("Success")
                        return(CreateChangeResponse.model.instance.header.
                         ChangeNumber.getValue());
                }

        }
        catch( e )
        {
                print( e );
        }
}

The following section shows how to call the above WebServices from JavaScrip
t:


////////////////////////////////////////////////////////////////////////
// Test Call
////////////////////////////////////////////////////////////////////////
var rc_Code_List=RetrieveChangeKeysList("category=\"" + "Release Management"
         + "\"" + "and number <=\"" + "C10027" + "\"");

//var rc_Code_List=RetrieveChangeKeysList("number>=\"" + "C10026" + "\"");

if (rc_Code_List != -1)
{
       for (var i = 0; i < rc_Code_List.length; i+=1)
       {
                var rc_Code=RetrieveChange(rc_Code_List[i].ChangeNumber.
                 getValue());
                var rc_Create=CreateChangeFromChange(rc_Code);
       }
}
```

Example to close an existing incident record

To run an example of closing an existing Incident Record from the Axis2Sample directory:

1. Follow the instructions in the Axis README.txt file located in the Axis2Sample\resources directory.
2. Enter the following in the DOS command prompt to close an incident:

```
C:\scs\sm920\server\webservices\sample\sm7webservices\
   Axis2Sample\bin>CloseIncidentSample -host <sm server>
  -port <sm port> -username falcon -incidentId IM10001
  -closeCode Fault -resolution "It works now"
```

Note: If the username and password are not entered, they default to “falcon” with no password.

Special considerations for using Keep-Alive with Service Manager

A Service Manager user session starts when the Service Manager Server receives the first request from a SOAP client and ends when the SOAP client closes the HTTP connection. The user login process is performed in the first SOAP client request and the user logout process is performed when the SOAP client ends this Service Manager session. A SOAP client can reduce the login and logout overhead by enabling HTTP persistent connections, also called HTTP keep-alive. If you want to use HTTP 1.1 Keep-Alive connections with a SOAP API client, the SOAP API client must support cookies. When Service Manager responds to the first POST request from the SOAP API client, it includes a Set Cookie header that conveys the servlet container sessionid to the client.

- Configure the SOAP stack with which the SOAP API client is written to support cookies. Axis and .NET can both be configured to do this.
- If the SOAP toolkit supports HTTP 1.1 Keep-Alive but not cookies, you can arrange for the application to echo back the JSESSIONID value in a Cookie header by adding code to the client application to manually create the HTTP header on the second and subsequent requests

Notes:

- In HTTP/1.1, persistent connections are the default behavior of any connection.
- If you use HTTP 1.0 you have to manually set the HTTP header “connection” to keep-alive.
- A SOAP client ends a Service Manager session by sending a request with the HTTP header “connection” set to “close”. If a close request is never received by the Service Manager server then this session is terminated by the Service Manager server when the webservices_ sessiontimeout time is reached.

Keep-Alive example for Service Manager

The following shows an example of the code for using Keep-Alive with Service Manager.

```
================================================================
client request:
```

```
POST /SM/7/ws HTTP/1.1
accept: application/xop+xml, text/xml image/gif, image/jpeg, *; q=.2,
*/*; q=.2
authorization: Basic ZmFsY29uOg==
soapaction: "Create"
connection: Keep-Alive
.....

SM server response:
HTTP/1.1 200
Set-Cookie: JSESSIONID=ED61093038F9FF8CE9CF44E34C9366AC; Path=/SM
Keep-Alive: timeout=1200000, max=1000
Connection: Keep-Alive
Content-Type: text/xml;charset=utf-8
Content-Length: 2112
.....
```

Then you need to echo back the JSESSIONID for each client use the following:

```
POST /SM/7/ws HTTP/1.1
accept: application/xop+xml, text/xml image/gif, image/jpeg, *; q=.2,
*/*; q=.2
authorization: Basic ZmFsY29uOg==
soapaction: "Retrieve"
connection: Keep-Alive
cookie: JSESSIONID=ED61093038F9FF8CE9CF44E34C9366AC; Path=/SM;
....
================================================================
```

Note:

If you use Axis2 to implement the client then Axis2 can maintain the session by calling the setManageSession(true).

There is an Axis2 example in SM711 installation directory:

(...\Server\webservices\sample\sm7webservices\Axis2Sample)

Use SSL to consume Service Manager Web Services

Provide the client keystore and the client keystore password to the Web Service consumer. He will need to enter this information into the proper location on his client. For example, in SOAPUI, you can enter this information in the **File > Preferences > SSL Settings** section. To find out how to create the client keystore, refer to the Service Manager Trusted Sign-on White paper. Ensure that the client's cacerts file contains the information on the authority that signed the keystore.

Attachment handling

What do the following error messages mean?

```
Error Message: Warning: incoming add attachment request 1 has no href
attribute

Error: unable to match incoming add attachment request 1 with no href
attribute to an attachment part
```

They indicate that the <attachment> elements in the XML in the SOAP requests do not have a href or contentId attribute value. The same value is supposed to be in the MIME message part as the Content-ID: value. In SOAP with attachments, we need a correlation between the XML element/attributes that describe the attachment, and the actual binary or base64 attachment content which is in a MIME message. This correlation is typically a unique ID specified in an href or Content-ID attribute.

The Service Manager server deliberately allows requests that omit the href or contentId and attempts to match up the XML and the attachment parts.

We report the missing href or contentId value with a message in the `sm.log` file, as follows:

```
Warning: incoming add attachment request 1 has no href attribute
```

The server first tries to get an href or contentId value out of the XML; if it succeeds, it finds the associated MIME attachment by looking for a MIME message part whose id has the same value. If there is no href or Content-ID, the server tries to match up the <attachment> element with a particular attachment part. This assumes that there is a one-to-one correspondence between <attachment> elements and attachment parts and uses the index of the DOM node of the <attachment> element as an index into the array of binary attachment parts.

This strategy does not work when there are miscellaneous white-space nodes in the DOM document, because the index number of the DOM node for the <attachment> element is greater than it would otherwise be.

Service Manager allows requests with no href or content-id

The reason Service Manager allows requests with no href or Content-ID is that with some tools and toolkits it is difficult to arrange for the unique id of an attachment part to be the same in the XML as in the binary attachment part. Though this is trivial when using .NET, when using Axis, the Java code would generate a unique cid value in the MIME message part dynamically during message serialization. Unless you write code to set up a handler to participate in serialization (via a callback), it is impossible to match the value in the XML to the value in the MIME message part. To prevent these problems, Service Manager:

- Relaxed the schema such that href was not strictly required (use=optional).
- Added an alternative, optional attribute called Content-ID, which is used instead of href when serving responses containing attachments to Axis clients.
- Added code to try to guess the href value that should be present in the XML, if it is missing. If we are processing the Nth <attachment> element (the Nth DOM Node within the set of DOM children for the <attachments> element, where the <attachment> element has neither an href nor a Content-ID attribute), Service Manager tries to look at the attachment part with the same

index value to check whether the name, length, and type match. If the number of DOM Node children under <attachments> does not match the number of attachment parts, Service Manager cannot process the attachment. It prints the following error in the `sm.ini` file:

```
Error: unable to match incoming add attachment request 2 with no href
attribute to an attachment part
```

This message says “attachment request 2”, which seems to be incorrect; because there is only 1 <attachment> element, it should apparently be “attachment request 1.” However, the attachment element is the second DOM child node of <attachments> due to the white space text present as DOM child node 1; the first child node of <attachments> is white space that may be ignored.

The workaround is either do not serialize with pretty-printing (such as adding white space nodes to make the XML easier to read for the human eye) when sending requests to Service Manager, or write code that ensures that requests containing attachment operations have either an href or Content-ID attribute on the <attachment> element.

Supported attachment types in Service Manager are MIME and MTOM. We often get the question if the consumer does not support these attachment types, if the SYSATTACHMENTS file can be exposed to get the attachments out of Service Manager. This is not supported. The attachments are compressed and cut into <=32K pieces and cannot easily be read from an outside source. A workaround that customers use frequently is receiving the attachment with the parent record, for example via a RetrieveIncident request, and then transforming it into base64 and sending to the consuming application where it can be transformed into the required format.

Sample script to send a ticket with attachments within Service Manager

The sample script below sends a ticket with attachments from Service Manager to ServiceManager. First, you will need to have a generated JavaScript for both Service Managers: Service Manager 1, called IncidentManagement, and Service Manager 2, called IncidentManagementTarget.

Note: The lines in bold font perform the attachment handling.

```
try
{
       var attach;
       var imService = new lib.IncidentManagement.IncidentManagement();
       var request = new lib.IncidentManagement.RetrieveIncidentRequest();
       request.model.instance.IncidentID.setValue( "IM1001" );
       request.attachmentInfo = true;
       request.attachmentData = true;

       imService.user = "falcon";
       imService.pwd = "";

       imService.setHost("localhost");
```

```
var incidentResp = imService.invoke( request );
if ( incidentResp.isFault() )
{
        print ( incidentResp.detail );
}
else
{
        if ( incidentResp.messages.message.length > 0 )
        {
                for ( i=0;i<incidentResp.messages.message.length;i++ )
                {
                        print ( "Message: " + incidentResp.messages.message[i].getValue()
                }
        }
        print( incidentResp.model.instance.attachments.attachment.length );
        print( incidentResp.model.instance.attachments.attachment[0].name );
        print( incidentResp.model.instance.attachments.attachment[0].len );

        attach = imService.attachments[0];
}
var imService2 = new lib.IncidentManagementTarget.IncidentManagement();
var newAttach = new Array();
attach.href = "12345";
attach.action="add";

newAttach.push( attach );

var createIM = new
lib.IncidentManagementTarget.CreateIncidentRequest();
imService2.setAttachments( newAttach );
imService2.setHost("localhost");
imService2.user = "falcon";
imService2.pwd = "";

createIM.attachmentData = true;
createIM.attachmentInfo = true;

createIM.model.instance.OpenedBy.setValue("BOB.HELPDESK");

var a = createIM.model.instance.IncidentDescription.
  IncidentDescription_newInstance();
a.setValue("Incident Description");
createIM.model.instance.BriefDescription.setValue("Brief Description");

createIM.model.instance.Category.setValue("telecoms");
createIM.model.instance.Subcategory.setValue("fixed infrastructure");
createIM.model.instance.ProductType.setValue("fixed infrastructure");
```

```
        createIM.model.instance.ProblemType.setValue("not specified");
        createIM.model.instance.InitialImpact.setValue("1");
        createIM.model.instance.Severity.setValue("1");
        createIM.model.instance.PrimaryAssignmentGroup.setValue("TELECOMS");

        var attachmentXml =
        createIM.model.instance.attachments.attachment_newInstance();

        attachmentXml.action  = attach.action;
        attachmentXml.name    = attach.name;
        attachmentXml.type    = attach.type;
        attachmentXml.len     = attach.value.length;
        attachmentXml.attachmentType = attach.attachmentType;

        print( createIM.model.instance.attachments.attachment.length );

        response = imService2.invoke(createIM);

        if ( response.isFault() )
        {
                print ( response.detail );
        }
        else
        {
                if ( response.messages.message.length > 0 )
                {
                        for ( i=0;i<response.messages.message.length;i++ )
                        {
                                print ( "Message: " + response.messages.
                                  message[i].getValue() );
                        }
                }

        }

}
catch( e )
{
       print( e );
}
```

Consume an external Web Service

You can configure Service Manager to connect to and exchange information with remote Web Services. This functionality allows Service Manage to act as a client to other servers that publish Web Services. Service Manager uses JavaScript™ to create and format the proper SOAP messages.

Note: For a production application that needs services that are not available within your corporate intranet (such as postal address verification, email address verification, and currency conversions) HP recommends that you investigate offerings from established for-fee Web Services vendors. Although there are a lot of free and demo Web Services, we do not recommend basing a production application on such services, since availability of the service is not guaranteed. Several Web sites such as www.xmethods.net publish lists of available free and fee-based Web Services. (Be sure to click the full list button to see the complete list of Web Services.)

Use the WSDL2JS utility

The WSDL2JS (Web Services description language to JavaScript) utility translates the operations and types in the WSDL into objects, methods and functions in JavaScript that can be called from another JavaScript record.

The WSDL2JS utility is a JavaScript script library record named SOAP. It is written based on the W3C specifications for WSDL to interpret the content of the WSDL.

To consume a Web Service from Service Manager perform the following steps.

1. Obtain the URL to the Web Service's WSDL file.
2. Examine the WSDL either as a text file, or using a third-party graphical WSDL analysis tool to determine what functions, inputs, and formats the Web service expects. Some third-party Web Services tools allow you to experiment interactively with Web Services. HP recommends that you familiarize yourself with the Web Service using such a tool before beginning any Service Manager JavaScript work.
3. Execute the Run WSDL to JS wizard to obtain and convert the Web service's WSDL into JavaScript.
4. Write custom JavaScript to call the JavaScript functions generated by the WSDL to JS wizard. These functions will enable you to create and send the SOAP messages required to interact with the Web service. HP recommends that you write a short “standalone” script and invoke it from the Script Library utility to test it prior to implementing the JavaScript call from Format Control, Triggers, or Display Options. After you have determined and debugged the JavaScript code required to invoke the service, you can then integrate the script with your Service Manager application.
5. Tailor your Service Manager application to invoke your custom JavaScript when you want to connect to a remote a Web service. Usually Web Services are invoked from the Document Engine, Format Control, Links, Display application, or from similar tailoring tools.

Best practices for writing a JavaScript to consume a Web service

Never modify the JavaScript that is automatically generated by WSDL2JS unless you are specifically instructed to do so by Service Manager Customer Support. To invoke the Web service, write a JavaScript record that calls the functions generated by WSDL2JS. The JavaScript that invokes an external Web service should perform the following tasks:

1. Create the Service Object.
2. Create the Request Object.
3. Fill the Request Object with information that defines the request.
4. Invoke the Service Object and pass in the Request Object.
5. Return either the Response Object, an instance of the Response Object, or a specific value of that instance.
6. Perform error handling to test each response. Use try {…}, throw {…}, catch {…}, and the isFault function.

As a best practice, do not reuse the names of variables and functions in the calling JavaScript that are the names of variables and functions in the generated script. This can help avoid confusion.

Important: Never use the “new” keyword on a subordinate object unless it is an array. Unlike conventionally compiled applications that invoke a Web service, the generated function objects described in this document already use “new” when instantiating all children, so it is not necessary to do so in the calling JavaScript. The only exception is for arrays, where you use the newInstance() function to generate the array and fill its elements.

Date/Time handling

The self-written JavaScript is responsible for correct formatting of xml schema dateTime fields. The WSDL2JS-generated functions do not reformat the values assigned to them to convert them into the correct format. If a field in an outgoing SOAP message is defined as a dateTime, and the script writer assigns a value to the field, it needs to be a valid XML Schema dateTime, duration or other date/time string value. It cannot be a Service Manager datetime string nor should it be a JavaScript dateTime string. To get the valid XML Schema date/time, the script writer should use the XMLDate global object. For example:

```
// get today’s date/time from Javascript Date() object and store in a
new XMLDate object
var xmlDt = new XMLDate( new Date() );

// coerce the datetime value stored in the XMLDate object to ISO/XML
schema format
print( xmlDt.getISODateTimeString() );
```

There are a variety of methods for the XMLDate object that you can look up in the white paper with the title of JavaScript Programmers Guide.pdf.

The constructor for XMLDate can handle several different formats. You can pass it a string, a number of milliseconds, or a JS Date object (as in the example above).

Example: Interface to another system

This is an example for interfacing the Service Manager ServiceCatalog with HP's PPM Demand service via Web Services.

When writing an interface to a different system, it is very important to understand the data structure and the methods available, as well as understand how to interpret the generated JavaScript code. In this example, we will not only discuss the methods and fields published by the WSDL, but reading the generated code to successfully write the invoking code as well.

1. Determine the correct URL to enter into the WSDL to JS tool, check with the PPM administrator.
2. In Service Manager, go to Menu Navigator and click **Utilities > Tools > Web Services**. Click **Run WSDL to JS**.
3. Enter the URL to the WSDL, such as: http://<hostname>:8080/itg/ppmservices/DemandService?wsdl
4. Click **Proceed**.
5. Click **Add** to add the new ScriptLibrary record called DemandService.
6. Write an interfacing JavaScript record in the ScriptLibrary called DemandServiceInvoke.

Generated JavaScript interfaces

This section helps provide a general understanding of how the generated JavaScript interfaces with the invoking JavaScript. As a best practice, find the proper objects and methods using a tool such as SoapUI and test the Web Service there prior to writing the invoking script. It should not be necessary to interpret the generated code when taking that approach.

Check these sections in the "master" JavaScript to write the calling JavaScript:

The first line in the master code gives the name of the main function or Service Object to call in the calling JavaScript:

```
function DemandService( )
{
  this.location = new String( "http://<ppm server>:15000/itg/
ppmservices/DemandService" );
```

Create a request for a new project

Find the function that creates the desired result; in this case create a request for a new project:

```
this.SOAPOperations[ "createRequest" ] = new soap_Operation

  ( "createRequest", "urn:createRequest", "document", "createRequest",

  "createRequestResponse" );
…
function createRequest(  )
{
  this.$$nsPrefix = "ns1";
  this.$$elementFormDefault = "qualified";
  this.$$attributes = new Array();
  this.$$xmlNames = new Array();
  this.$$objNames = new Array();
  this.$$minOccurs = new Array();
  this.$$refs = new Array();
  this.getName = getName;
  this.getXmlName = getXmlName;
  this.setContent = setContent;
  this.addContent = addContent;
  this.getContent = getContent;
  this.isFault = isFault;
  this.$$elementChildren = new Array();
  this.$$name = "createRequest";
  this.$$xmlNames[ "createRequest" ] = "createRequest";
  this.xmlns_ns1 = new String("http://mercury.com/ppm/dm/service/1.0");
  this.$$attributes.push( "xmlns_ns1" );
  this.$$xmlNames["xmlns_ns1"] = "xmlns:ns1";
  this.$$objNames["xmlns:ns1"] = "xmlns_ns1";
  this.requestObj = new Request();
  this.$$elementChildren.push( "requestObj" );
}
```

The structure of the request

The bold $$elementChildren.push section shows that the requestObj Child element is of type Request(). To find the structure of the Request, find the definition of that function in the generated code.

```
function Request(  )
{
  this.$$nsPrefix = "ns8";
  this.$$elementFormDefault = "qualified";
  this.$$attributes = new Array();
  this.$$xmlNames = new Array();
  this.$$objNames = new Array();
  this.$$minOccurs = new Array();
  this.$$refs = new Array();
  this.getName = getName;
```

```
this.getXmlName = getXmlName;
this.setContent = setContent;
this.addContent = addContent;
this.getContent = getContent;
this.isFault = isFault;
this.$$elementChildren = new Array();
this.$$name = "Request";
this.$$xmlNames[ "Request" ] = "Request";
this.$$minOccurs[ "id" ] = 0;
this.id = new xs_string();
this.$$elementChildren.push( "id" );
this.requestType = new xs_string();
this.$$elementChildren.push( "requestType" );
this.simpleFields = new Array();  // of SimpleField
this.simpleFields.$$nsPrefix = "ns8"
this.simpleFields_newInstance = function()
{
  var newLen = this.simpleFields.push( new SimpleField() );
  return this.simpleFields[ newLen-1 ];
}
this.$$elementChildren.push( "simpleFields" );
this.tables = new Array();  // of Table
this.tables.$$nsPrefix = "ns8"
this.tables_newInstance = function()
{
  var newLen = this.tables.push( new Table() );
  return this.tables[ newLen-1 ];
}
this.$$elementChildren.push( "tables" );
this.notes = new Array();  // of Note
this.notes.$$nsPrefix = "ns8"
this.notes_newInstance = function()
{
  var newLen = this.notes.push( new Note() );
  return this.notes[ newLen-1 ];
}
this.$$elementChildren.push( "notes" );
this.fieldChangeNotes = new Array();  // of FieldChangeNote
this.fieldChangeNotes.$$nsPrefix = "ns8"
this.fieldChangeNotes_newInstance = function()
{
  var newLen = this.fieldChangeNotes.push( new FieldChangeNote() );
  return this.fieldChangeNotes[ newLen-1 ];
}
this.$$elementChildren.push( "fieldChangeNotes" );
this.URLReferences = new Array();  // of URLReference
this.URLReferences.$$nsPrefix = "ns8"
this.URLReferences_newInstance = function()
{
```

```
    var newLen = this.URLReferences.push( new URLReference() );
    return this.URLReferences[ newLen-1 ];
  }
  this.$$elementChildren.push( "URLReferences" );
  this.remoteReferences = new Array();  // of RemoteReference
  this.remoteReferences.$$nsPrefix = "ns8"
  this.remoteReferences_newInstance = function()
  {
    var newLen = this.remoteReferences.push( new RemoteReference() );
    return this.remoteReferences[ newLen-1 ];
  }
  this.$$elementChildren.push( "remoteReferences" );
}
```

Request object

The Request object can contain simple Fields, Notes, Field Change Notes, Tables etc. This example examines these fields.

```
function SimpleField(  )
{
  this.$$nsPrefix = "ns8";
  this.$$elementFormDefault = "qualified";
  this.$$attributes = new Array();
  this.$$xmlNames = new Array();
  this.$$objNames = new Array();
  this.$$minOccurs = new Array();
  this.$$refs = new Array();
  this.getName = getName;
  this.getXmlName = getXmlName;
  this.setContent = setContent;
  this.addContent = addContent;
  this.getContent = getContent;
  this.isFault = isFault;
  this.$$elementChildren = new Array();
  this.$$name = "SimpleField";
  this.$$xmlNames[ "SimpleField" ] = "SimpleField";
  this.token = new xs_string();
  this.token.$$nsPrefix = "ns7"
  this.$$refs[ "token" ] = true;
  this.$$elementChildren.push( "token" );
  this.stringValue = new Array();  // of xs_string
  this.stringValue.$$nsPrefix = "ns8"
  this.stringValue_newInstance = function()
  {
    var newLen = this.stringValue.push( new xs_string() );
    return this.stringValue[ newLen-1 ];
  }
  this.$$elementChildren.push( "stringValue" );
```

```
  this.$$minOccurs[ "dateValue" ] = 0;
  this.dateValue = new xs_dateTime();
  this.$$elementChildren.push( "dateValue" );
}
```

Simple fields

Simple fields consist of tokens (of type xs_string), as well as instances of string Values (where each element is of type xs_string).

Check the xs_string() function

When checking the xs_string() function, you will find that the JavaScript uses the setValue function to fill the elements with data.

```
function xs_string( val )
{
  this.$$nsPrefix = "xsd";
  this.$$elementFormDefault = "qualified";
  this.$$attributes = new Array();
  this.$$xmlNames = new Array();
  this.$$objNames = new Array();
  this.$$minOccurs = new Array();
  this.$$refs = new Array();
  this.getValue = getValue;
  this.setValue = setValue;
  this.$$value = val;
  this.xsi_type = new String("xsd:string");
  this.$$attributes.push( "xsi_type" );
  this.$$xmlNames["xsi_type"] = "xsi:type";
  this.$$objNames["xsi:type"] = "xsi_type";
}

function setValue( value )
{
 this.$$value = value;
}
```

Check expected parameters in invoke() function

Check which parameters the invoke() function expects.

```
function invoke( requestObj, headerObj, bEmitXsiTypeAttributes )
```

In this case, the headerObj and the bEmitXsiTypeAttributes are optional. They are “nullsub’ed” in the JavaScript code), so the requestObj is the only required argument.

Check the syntax for the Response function

```
function createRequestResponse(  )
{
  this.$$nsPrefix = "ns1";
  this.$$elementFormDefault = "qualified";
  this.$$attributes = new Array();
  this.$$xmlNames = new Array();
  this.$$objNames = new Array();
  this.$$minOccurs = new Array();
  this.$$refs = new Array();
  this.getName = getName;
  this.getXmlName = getXmlName;
  this.setContent = setContent;
  this.addContent = addContent;
  this.getContent = getContent;
  this.isFault = isFault;
  this.$$elementChildren = new Array();
  this.$$name = "createRequestResponse";
  this.$$xmlNames[ "createRequestResponse" ] = "createRequestResponse";
  this.$$xmlNames["_return"] = "return";
  this.$$objNames["return"] = "_return";
  this._return = new RemoteReference();
  this.$$elementChildren.push( "_return" );
}
```

Use getValue

Use getValue (or a similarly defined function) to read the result of the request.

```
function getValue( )
{
 return this.$$value;
}
```

Write the invoking JavaScript code

Now that the generated JavaScript gave information on the structure of the code to use for invoke, write the invoking JavaScript code. In this case, the invoking code gets passed in information from a Service Catalog Item. This information is then processed and passed through to the PPM Web Service.

```
function CreateDemandRequest(CartItem)
{

       try
       {
//////////////////////////////////////////////////////////////////////
```

```
//    Initialization Section
////////////////////////////////////////////////////////////////////
//             first, initialize the Service Object for this JavaScript
               var DemandService=new system.library.DemandService.DemandService();


//             set Connection information (optional)
               DemandService.user = "admin";
               DemandService.password = "admin";
//             DemandService.location="http://<ppm server>:15000
                 /itg/ppmservices/DemandService";
               DemandService.location="http://localhost:15001
                 /itg/ppmservices/DemandService";

//             create the request object
               var RequestDemRequest =
                 new system.library.DemandService.createRequest();


////////////////////////////////////////////////////////////////////////
//     Data Fill Section
////////////////////////////////////////////////////////////////////////

// Data from Cart Item

               var PlannedStart=system.library.xmlFill.
                getValue(CartItem.options, "PlannedStart");
               var PlannedEnd=system.library.xmlFill.
                getValue(CartItem.options, "PlannedEnd");
               var ProjectName=system.library.xmlFill.
                getValue(CartItem.options, "ProjectName");
               var ProjectManager=system.library.xmlFill.getValue
           (CartItem.options, "ProjectManager");
               var Region=system.library.xmlFill.getValue
                (CartItem.options, "Region");
               var ProjectType=system.library.xmlFill.getValue
                (CartItem.options, "ProjectType");

// Fill into Web Request
               RequestDemRequest.requestObj.requestType.setValue
               ( "PFM - Proposal" );

               RequestDemRequest.requestObj.simpleFields_newInstance();

               RequestDemRequest.requestObj.simpleFields[0].token.setValue
                ("REQ.VP.KNTA_PLAN_START_DATE");
                var String0=RequestDemRequest.requestObj.simpleFields[0]
                .stringValue_newInstance();
               String0.setValue(PlannedStart)
```

```
                RequestDemRequest.requestObj.simpleFields_newInstance();

                RequestDemRequest.requestObj.simpleFields[1].token.setValue
                  ("REQ.VP.KNTA_PLAN_FINISH_DATE");
                var String1=RequestDemRequest.requestObj.simpleFields[1]
                   .stringValue_newInstance();
                String1.setValue(PlannedEnd)

                RequestDemRequest.requestObj.simpleFields_newInstance();
                RequestDemRequest.requestObj.simpleFields[2].token.setValue
                 ("REQ.VP.KNTA_PROJECT_NAME");
                var String2=RequestDemRequest.requestObj.simpleFields[2]
                 .stringValue_newInstance();
                String2.setValue(ProjectName)

                RequestDemRequest.requestObj.simpleFields_newInstance();
                RequestDemRequest.requestObj.simpleFields[3].token
                  .setValue("REQ.VP.KNTA_PROJECT_MANAGER");
                var String3=RequestDemRequest.requestObj.simpleFields[3]
                  .stringValue_newInstance();
                String3.setValue(ProjectManager)

                RequestDemRequest.requestObj.simpleFields_newInstance();
                RequestDemRequest.requestObj.simpleFields[4].token
                   .setValue("REQ.VP.KNTA_REGION");
                var String4=RequestDemRequest.requestObj.simpleFields[4]
                   .stringValue_newInstance();
                String4.setValue(Region)

                RequestDemRequest.requestObj.simpleFields_newInstance();
                RequestDemRequest.requestObj.simpleFields[5].token.setValue
                  ("REQ.VP.KNTA_PROJECT_TYPE");
                var String5=RequestDemRequest.requestObj.simpleFields[5]
                  .stringValue_newInstance();
                String5.setValue(ProjectType)

//              var ProjectNotes=RequestDemRequest.requestObj.
                   notes_newInstance();
//              ProjectNotes.content.setValue("notes");
RequestDemRequest.requestObj.simpleFields[4].token.getValue())

////////////////////////////////////////////////////////////////////////
//     Invoke and final processing
////////////////////////////////////////////////////////////////////////
                var DemandResponse = DemandService.invoke
                ( RequestDemRequest );

                if ( DemandResponse.isFault() )
                {
```

```
            print( DemandResponse.faultcode.getValue() );
            print( DemandResponse.faultstring.getValue() );
            print( DemandResponse.detail.getValue() );
            return("Failure");
               }
               else
               {
                       print("Success")
                       return("Success");
               }

       }
       catch( e )
       {
               print( e );
       }

}

////////////////////////////////////////////////////////////////////////
// Test Call
////////////////////////////////////////////////////////////////////////

//var rc_Code=CreateDemandRequest(CartItem);
```

Determine the structure of the request and response

To determine the structure of the request and response, it is very helpful to look at both the request and response using a tool such as SOAP UI. The PPM WSDL shown below generated the request and response in the next section using SOAP UI.

PPM WSDL:

```
- <wsdl:definitions xmlns:ds="http://mercury.com/ppm/dm/service/1.0"
   xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
   xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
   xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
   xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:dmMsg="http://mercury.com/ppm/dm/msg/1.0"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    targetNamespace="http://mercury.com/ppm/dm/msg/1.0">
  <wsdl:documentation>DemandService</wsdl:documentation>
- <wsdl:types>
- <xs:schema xmlns:dm="http://mercury.com/ppm/dm/1.0"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:common="http://mercury.com/ppm/common/1.0"
   attributeFormDefault="qualified" elementFormDefault="qualified"
   targetNamespace="http://mercury.com/ppm/dm/service/1.0">
```

```
  <xs:import namespace="http://mercury.com/ppm/dm/1.0"
   schemaLocation="DemandService?xsd=xsd0" />
- <xs:element name="createRequest">
- <xs:complexType>
- <xs:sequence>
  <xs:element name="requestObj" type="dm:Request" />
  </xs:sequence>
  </xs:complexType>
  </xs:element>
- <xs:element name="createRequestResponse">
- <xs:complexType>
- <xs:sequence>
  <xs:element name="return" nillable="true" type="dm:RemoteReference" />
  </xs:sequence>
  </xs:complexType>
  </xs:element>
- <xs:element name="getRequests">
- <xs:complexType>
- <xs:sequence>
  <xs:element maxOccurs="unbounded" name="requestIds"
   nillable="true" type="dm:Identifier" />
  </xs:sequence>
  </xs:complexType>
  </xs:element>
- <xs:element name="getRequestsResponse">
- <xs:complexType>
- <xs:sequence>
  <xs:element maxOccurs="unbounded" name="return"
   nillable="true" type="dm:Request" />
  </xs:sequence>
  </xs:complexType>
  </xs:element>
- <xs:element name="addRequestNotes">
- <xs:complexType>
- <xs:sequence>
  <xs:element name="requestId" type="dm:Identifier" />
  <xs:element maxOccurs="unbounded" name="notes" type="common:Note" />
  </xs:sequence>
  </xs:complexType>
  </xs:element>
- <xs:element name="addRequestNotesResponse">
- <xs:complexType>
- <xs:sequence>
  <xs:element name="return" nillable="true" type="xs:int" />
  </xs:sequence>
  </xs:complexType>
  </xs:element>
- <xs:element name="deleteRequests">
- <xs:complexType>
```

```
- <xs:sequence>
  <xs:element maxOccurs="unbounded" name="requestIds"
   type="dm:Identifier" />
  </xs:sequence>
  </xs:complexType>
  </xs:element>
- <xs:element name="deleteRequestsResponse">
- <xs:complexType>
- <xs:sequence>
  <xs:element name="return" nillable="true" type="xs:int" />
  </xs:sequence>
  </xs:complexType>
  </xs:element>
- <xs:element name="setRequestRemoteReferenceStatus">
- <xs:complexType>
- <xs:sequence>
  <xs:element name="receiver" type="dm:Identifier" />
  <xs:element name="source" type="dm:Identifier" />
  <xs:element name="status" type="xs:string" />
  <xs:element maxOccurs="unbounded" name="fields" nillable="true"
   type="dm:SimpleField" />
  </xs:sequence>
  </xs:complexType>
  </xs:element>
- <xs:element name="setRequestRemoteReferenceStatusResponse">
- <xs:complexType>
- <xs:sequence>
  <xs:element name="return" nillable="true" type="xs:int" />
  </xs:sequence>
  </xs:complexType>
  </xs:element>
- <xs:element name="setRequestFields">
- <xs:complexType>
- <xs:sequence>
  <xs:element name="requestId" type="dm:Identifier" />
  <xs:element maxOccurs="unbounded" name="fields"
   type="dm:SimpleField" />
  </xs:sequence>
  </xs:complexType>
  </xs:element>
- <xs:element name="setRequestFieldsResponse">
- <xs:complexType>
- <xs:sequence>
  <xs:element name="return" nillable="true" type="xs:int" />
  </xs:sequence>
  </xs:complexType>
  </xs:element>
- <xs:element name="executeWFTransitions">
- <xs:complexType>
```

```
- <xs:sequence>
  <xs:element name="receiver" type="dm:Identifier" />
  <xs:element name="transition" type="xs:string" />
  </xs:sequence>
  </xs:complexType>
  </xs:element>
- <xs:element name="executeWFTransitionsResponse">
- <xs:complexType>
- <xs:sequence>
  <xs:element name="return" nillable="true" type="xs:string" />
  </xs:sequence>
  </xs:complexType>
  </xs:element>
  </xs:schema>
  </wsdl:types>
- <wsdl:message name="setRequestRemoteReferenceStatusMessage">
  <wsdl:part name="part1" element="ds:setRequestRemoteReferenceStatus" />
  </wsdl:message>
- <wsdl:message name="setRequestRemoteReferenceStatusResponseMessage">
  <wsdl:part name="part1"
   element="ds:setRequestRemoteReferenceStatusResponse" />
  </wsdl:message>
- <wsdl:message name="addRequestNotesMessage">
  <wsdl:part name="part1" element="ds:addRequestNotes" />
  </wsdl:message>
- <wsdl:message name="addRequestNotesResponseMessage">
  <wsdl:part name="part1" element="ds:addRequestNotesResponse" />
  </wsdl:message>
- <wsdl:message name="createRequestMessage">
  <wsdl:part name="part1" element="ds:createRequest" />
  </wsdl:message>
- <wsdl:message name="createRequestResponseMessage">
  <wsdl:part name="part1" element="ds:createRequestResponse" />
  </wsdl:message>
- <wsdl:message name="deleteRequestsMessage">
  <wsdl:part name="part1" element="ds:deleteRequests" />
  </wsdl:message>
- <wsdl:message name="deleteRequestsResponseMessage">
  <wsdl:part name="part1" element="ds:deleteRequestsResponse" />
  </wsdl:message>
- <wsdl:message name="setRequestFieldsMessage">
  <wsdl:part name="part1" element="ds:setRequestFields" />
  </wsdl:message>
- <wsdl:message name="setRequestFieldsResponseMessage">
  <wsdl:part name="part1" element="ds:setRequestFieldsResponse" />
  </wsdl:message>
- <wsdl:message name="getRequestsMessage">
  <wsdl:part name="part1" element="ds:getRequests" />
  </wsdl:message>
```

```
- <wsdl:message name="getRequestsResponseMessage">
  <wsdl:part name="part1" element="ds:getRequestsResponse" />
  </wsdl:message>
- <wsdl:message name="executeWFTransitionsMessage">
  <wsdl:part name="part1" element="ds:executeWFTransitions" />
  </wsdl:message>
- <wsdl:message name="executeWFTransitionsResponseMessage">
  <wsdl:part name="part1" element="ds:executeWFTransitionsResponse" />
  </wsdl:message>
- <wsdl:portType name="DemandServicePortType">
- <wsdl:operation name="setRequestRemoteReferenceStatus">
  <wsdl:input message="dmMsg:setRequestRemoteReferenceStatusMessage"
   wsaw:Action="urn:setRequestRemoteReferenceStatus" />
  <wsdl:output message=
   "dmMsg:setRequestRemoteReferenceStatusResponseMessage"
   wsaw:Action="http://mercury.com/ppm/dm/msg/1.0/
   DemandServicePortType/setRequestRemoteReferenceStatus" />
  </wsdl:operation>
- <wsdl:operation name="addRequestNotes">
  <wsdl:input message="dmMsg:addRequestNotesMessage"
   wsaw:Action="urn:addRequestNotes" />
  <wsdl:output message="dmMsg:addRequestNotesResponseMessage"
   wsaw:Action="http://mercury.com/ppm/dm/msg/1.0
   /DemandServicePortType/addRequestNotesResponse" />
  </wsdl:operation>
- <wsdl:operation name="createRequest">
  <wsdl:input message="dmMsg:createRequestMessage"
   wsaw:Action="urn:createRequest" />
  <wsdl:output message="dmMsg:createRequestResponseMessage"
   wsaw:Action="http://mercury.com/ppm/dm/msg/1.0/
   DemandServicePortType/createRequestResponse" />
  </wsdl:operation>
- <wsdl:operation name="deleteRequests">
  <wsdl:input message="dmMsg:deleteRequestsMessage"
   wsaw:Action="urn:deleteRequests" />
  <wsdl:output message="dmMsg:deleteRequestsResponseMessage"
   wsaw:Action="http://mercury.com/ppm/dm/msg/1.0
   /DemandServicePortType/deleteRequestsResponse" />
  </wsdl:operation>
- <wsdl:operation name="setRequestFields">
  <wsdl:input message="dmMsg:setRequestFieldsMessage"
   wsaw:Action="urn:setRequestFields" />
  <wsdl:output message="dmMsg:setRequestFieldsResponseMessage"
   wsaw:Action="http://mercury.com/ppm/dm/msg/1.0
   /DemandServicePortType/setRequestFieldsResponse" />
  </wsdl:operation>
- <wsdl:operation name="getRequests">
  <wsdl:input message="dmMsg:getRequestsMessage"
   wsaw:Action="urn:getRequests" />
```

```
  <wsdl:output message="dmMsg:getRequestsResponseMessage"
   wsaw:Action="http://mercury.com/ppm/dm/msg/1.0
   /DemandServicePortType/getRequestsResponse" />
  </wsdl:operation>
- <wsdl:operation name="executeWFTransitions">
  <wsdl:input message="dmMsg:executeWFTransitionsMessage"
   wsaw:Action="urn:executeWFTransitions" />
  <wsdl:output message="dmMsg:executeWFTransitionsResponseMessage"
   wsaw:Action="http://mercury.com/ppm/dm/msg/1.0
   /DemandServicePortType/executeWFTransitionsResponse" />
  </wsdl:operation>
  </wsdl:portType>
- <wsdl:binding name="DemandServiceSOAP11Binding"
  type="dmMsg:DemandServicePortType">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
   style="document" />
- <wsdl:operation name="setRequestRemoteReferenceStatus">
  <soap:operation soapAction="urn:setRequestRemoteReferenceStatus"
   style="document" />
- <wsdl:input>
  <soap:body use="literal" />
  </wsdl:input>
- <wsdl:output>
  <soap:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="addRequestNotes">
  <soap:operation soapAction="urn:addRequestNotes" style="document" />
- <wsdl:input>
  <soap:body use="literal" />
  </wsdl:input>
- <wsdl:output>
  <soap:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="createRequest">
  <soap:operation soapAction="urn:createRequest" style="document" />
- <wsdl:input>
  <soap:body use="literal" />
  </wsdl:input>
- <wsdl:output>
  <soap:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="deleteRequests">
  <soap:operation soapAction="urn:deleteRequests" style="document" />
- <wsdl:input>
  <soap:body use="literal" />
  </wsdl:input>
```

```
- <wsdl:output>
  <soap:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="setRequestFields">
  <soap:operation soapAction="urn:setRequestFields" style="document" />
- <wsdl:input>
  <soap:body use="literal" />
  </wsdl:input>
- <wsdl:output>
  <soap:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="getRequests">
  <soap:operation soapAction="urn:getRequests" style="document" />
- <wsdl:input>
  <soap:body use="literal" />
  </wsdl:input>
- <wsdl:output>
  <soap:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="executeWFTransitions">
  <soap:operation soapAction="urn:executeWFTransitions"
   style="document" />
- <wsdl:input>
  <soap:body use="literal" />
  </wsdl:input>
- <wsdl:output>
  <soap:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
  </wsdl:binding>
- <wsdl:binding name="DemandServiceSOAP12Binding"
   type="dmMsg:DemandServicePortType">
  <soap12:binding transport="http://schemas.xmlsoap.org/soap
   /http" style="document" />
- <wsdl:operation name="setRequestRemoteReferenceStatus">
  <soap12:operation soapAction="urn:setRequestRemoteReferenceStatus"
  style="document" />
- <wsdl:input>
  <soap12:body use="literal" />
  </wsdl:input>
- <wsdl:output>
  <soap12:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="addRequestNotes">
  <soap12:operation soapAction="urn:addRequestNotes" style="document" />
```

```
- <wsdl:input>
  <soap12:body use="literal" />
  </wsdl:input>
- <wsdl:output>
  <soap12:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="createRequest">
  <soap12:operation soapAction="urn:createRequest" style="document" />
- <wsdl:input>
  <soap12:body use="literal" />
  </wsdl:input>
- <wsdl:output>
  <soap12:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="deleteRequests">
  <soap12:operation soapAction="urn:deleteRequests" style="document" />
- <wsdl:input>
  <soap12:body use="literal" />
  </wsdl:input>
- <wsdl:output>
  <soap12:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="setRequestFields">
  <soap12:operation soapAction="urn:setRequestFields" style="document" />
- <wsdl:input>
  <soap12:body use="literal" />
  </wsdl:input>
- <wsdl:output>
  <soap12:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="getRequests">
  <soap12:operation soapAction="urn:getRequests" style="document" />
- <wsdl:input>
  <soap12:body use="literal" />
  </wsdl:input>
- <wsdl:output>
  <soap12:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="executeWFTransitions">
  <soap12:operation soapAction="urn:executeWFTransitions"
  style="document" />
- <wsdl:input>
  <soap12:body use="literal" />
  </wsdl:input>
```

```
- <wsdl:output>
  <soap12:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
  </wsdl:binding>
- <wsdl:binding name="DemandServiceHttpBinding"
   type="dmMsg:DemandServicePortType">
  <http:binding verb="POST" />
- <wsdl:operation name="setRequestRemoteReferenceStatus">
  <http:operation location="setRequestRemoteReferenceStatus" />
- <wsdl:input>
  <mime:content type="text/xml" />
  </wsdl:input>
- <wsdl:output>
  <mime:content type="text/xml" />
  </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="addRequestNotes">
  <http:operation location="addRequestNotes" />
- <wsdl:input>
  <mime:content type="text/xml" />
  </wsdl:input>
- <wsdl:output>
  <mime:content type="text/xml" />
  </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="createRequest">
  <http:operation location="createRequest" />
- <wsdl:input>
  <mime:content type="text/xml" />
  </wsdl:input>
- <wsdl:output>
  <mime:content type="text/xml" />
  </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="deleteRequests">
  <http:operation location="deleteRequests" />
- <wsdl:input>
  <mime:content type="text/xml" />
  </wsdl:input>
- <wsdl:output>
  <mime:content type="text/xml" />
  </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="setRequestFields">
  <http:operation location="setRequestFields" />
- <wsdl:input>
  <mime:content type="text/xml" />
  </wsdl:input>
```

```
- <wsdl:output>
  <mime:content type="text/xml" />
  </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="getRequests">
  <http:operation location="getRequests" />
- <wsdl:input>
  <mime:content type="text/xml" />
  </wsdl:input>
- <wsdl:output>
  <mime:content type="text/xml" />
  </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="executeWFTransitions">
  <http:operation location="executeWFTransitions" />
- <wsdl:input>
  <mime:content type="text/xml" />
  </wsdl:input>
- <wsdl:output>
  <mime:content type="text/xml" />
  </wsdl:output>
  </wsdl:operation>
  </wsdl:binding>
- <wsdl:service name="DemandService">
- <wsdl:port name="DemandServiceSOAP11port_http"
  binding="dmMsg:DemandServiceSOAP11Binding">
  <soap:address location="http://<ppm server>:15000/itg/ppmservices/
  DemandService" />
  </wsdl:port>
- <wsdl:port name="DemandServiceSOAP12port_http"
   binding="dmMsg:DemandServiceSOAP12Binding">
  <soap12:address location="http://<ppm server>:15000/itg/
   ppmservices/DemandService" />
  </wsdl:port>
- <wsdl:port name="DemandServiceHttpport1"
   binding="dmMsg:DemandServiceHttpBinding">
  <http:address location="http://<ppm server>:15000/itg/
   ppmrest/DemandService" />
  </wsdl:port>
  </wsdl:service>
  </wsdl:definitions>
```

PPM request

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/
  soap/envelope/" xmlns:ns="http://mercury.com/ppm/dm/service/1.0"
  xmlns:ns1="http://mercury.com/ppm/dm/1.0"
  xmlns:ns2="http://mercury.com/ppm/common/1.0">
```

```
<soapenv:Header/>
<soapenv:Body>
   <ns:createRequest>
      <ns:requestObj>
         <!--Optional:-->
         <ns1:id></ns1:id>
         <ns1:requestType></ns1:requestType>
         <!--Zero or more repetitions:-->
         <ns1:simpleFields>
            <ns2:token>REQ.VP.KNTA_PLAN_START_DATE</ns2:token>
            <!--Zero or more repetitions:-->
            <ns1:stringValue>May 2008</ns1:stringValue>
            <!--Optional:-->
            <ns1:dateValue></ns1:dateValue>
         </ns1:simpleFields>
         <!--1 or more repetitions:-->
         <ns1:tables>
            <ns2:token></ns2:token>
            <!--1 or more repetitions:-->
            <ns2:columns>
               <ns2:token></ns2:token>
               <!--1 or more repetitions:-->
               <ns2:values></ns2:values>
               <!--1 or more repetitions:-->
               <ns2:dates></ns2:dates>
            </ns2:columns>
         </ns1:tables>
         <!--1 or more repetitions:-->
         <ns1:notes>
            <!--Optional:-->
            <ns2:author></ns2:author>
            <!--Optional:-->
            <ns2:creationDate></ns2:creationDate>
            <!--Optional:-->
            <ns2:content></ns2:content>
         </ns1:notes>
         <!--1 or more repetitions:-->
         <ns1:fieldChangeNotes>
            <!--Optional:-->
            <ns2:author></ns2:author>
            <!--Optional:-->
            <ns2:creationDate>?</ns2:creationDate>
            <!--Optional:-->
            <ns2:content></ns2:content>
            <ns1:fieldPrompt></ns1:fieldPrompt>
            <ns1:oldValue></ns1:oldValue>
            <ns1:newValue></ns1:newValue>
         </ns1:fieldChangeNotes>
         <!--1 or more repetitions:-->
```

```
            <ns1:URLReferences>
               <!--Optional:-->
               <ns1:addedBy></ns1:addedBy>
               <!--Optional:-->
               <ns1:creationDate></ns1:creationDate>
               <!--Optional:-->
               <ns1:description></ns1:description>
               <!--Optional:-->
               <ns1:name></ns1:name>
               <ns1:refURL></ns1:refURL>
            </ns1:URLReferences>
            <!--1 or more repetitions:-->
            <ns1:remoteReferences>
               <!--Optional:-->
               <ns1:addedBy></ns1:addedBy>
               <!--Optional:-->
               <ns1:creationDate></ns1:creationDate>
               <!--Optional:-->
               <ns1:description></ns1:description>
               <!--Optional:-->
               <ns1:name></ns1:name>
               <!--Optional:-->
               <ns1:displayURL></ns1:displayURL>
               <ns1:identifier>
                  <ns1:id></ns1:id>
                  <!--Optional:-->
                  <ns1:serverURL></ns1:serverURL>
               </ns1:identifier>
               <!--Optional:-->
               <ns1:status></ns1:status>
            </ns1:remoteReferences>
         </ns:requestObj>
      </ns:createRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

PPM response

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/
  soap/envelope/">
   <soapenv:Header/>
   <soapenv:Body>
      <soapenv:Fault>
         <faultcode>INTERNAL ERROR</faultcode>
         <faultstring>Internal error has occurred while calling
          PPM Web Service. Please contact PPM support with the detail
          information if the problem persists. (KNTA-11186)
          Details: Missing 'T' separator in dateTime</faultstring>
```

```
        <detail>
           <exception:exceptionDetails xmlns:exception=
             "http://www.mercury.com/ppm/ws/exception">
              <exception:detail>Missing 'T' separator in dateTime
               </exception:detail>
           </exception:exceptionDetails>
        </detail>
      </soapenv:Fault>
   </soapenv:Body>
</soapenv:Envelope>
```

Web Services with a proxy server

It is possible to consume Web Services through a proxy server in Service Manager. The proxy server settings allow your Service Manager server to connect to remote sites and download the WSDL for the remote Web Services. The following parameters have to be added to the `sm.ini` file for the Web service to connect through a proxy server.

- `JVMOptionX:-Dhttp.proxyHost=proxyserver.domain.company.com`
- `JVMOptionY:-Dhttp.proxyPort=<port number, 8088>`

You can also specify a list of hosts to bypass the proxy:

`JVMOptionZ:-Dhttp.nonProxyHosts="*.americas.hpqcorp.net|localhost"`

The `http.nonProxyHosts` property indicates the hosts which should connect directly and not through the proxy server. The value can be a list of hosts, each separated by a |, and in addition a wildcard character (*) can be used for matching.

The X, Y and Z represent three consecutive numbers. The first JVMOption in the sm.ini will be number 1, the next will be number 2 and 3 etc. If these three are the only JVMOptions in your `sm.ini`, they will be:

- `JVMOption1:-Dhttp.proxyHost=proxyserver.domain.company.com`
- `JVMOption2:-Dhttp.proxyPort=<port number, e.g. 8088>`
- `JVMOption3:-Dhttp.nonProxyHosts="*.domain.company.com|localhost"`

Connecting to a secure Web service

If you are consuming a secure Web service that requires mutual authentication from Service Manager application using Javascript and WSDL2JS, follow these steps:.

If you are consuming an SSL-protected Web service using Javascript in SM 7x, which uses java.xml.soap.SOAPConnection to send the request, the SSL configuration is done using Java key stores. Refer to the documentation for the list of sm.ini parameters required for SSL configuration.

When you consume a secure Web service or Web site from JavaScript all you need to do is use an https:// URL. There are no facilities for configuring SSL in WSDL2JS or in your script. The SSL

communication that is initiated by the WSDL2JS-generated code relies on the SSL configuration that is in place for the server itself. The Service Manager server's server certificate in effect becomes the client certificate for the outbound request.

To supply a Basic Authorization header when consuming a Web service using JavaScript generated by WSDL2JS, basic authentication is supplied automatically if you supply userid and password values on the service object generated by WSDL2JS as shown in the below example:

```
var service = new system.library.IncidentManagement.
IncidentManagement();

service.user = "falcon";
service.password = "";
...
```

If on the other hand, you are coding a REST-style GET directly in your script, you need to code it manually, because you have to code the HTTP request yourself. Add the following style code in the JavaScript to perform this:

```
// HTTP GET example with Basic Auth header

    var headers = new Array();

    try
    {
      if ( result.userid != undefined )
      {
        var authHeader = new Header();

        authHeader.name = "Authorization";
        authHeader.value = "Basic " + base64Encode
       ( result.userid + ":" + result.password );

        headers.push( authHeader );
      }

   strWSDLsrc = doHTTPRequest( "GET", wsdlURL, headers, null,
       10, 10, 10 );
 }
    catch( e )
    {
      print( "WSDL request failed with exception " + e );
      ...
    }
```

Use SSL connections to connect to an external Web service

When using SSL connections to an external Web service, the Service Manager server acts like a client and must be set up accordingly. The Web service provider must send the root certificate or

the certificate authority's (CA) certificate to the Service Manager administrator. If it is a certificate hierarchy, all certificates must be sent. Add this certificate to the Service Manager cacerts file using keytool.

For an anonymous SSL connection with an external Web Service using WSDL2JS, you need a root certificate file which includes the certificate for the CA that signed the remote Web Server's certificate. The cacerts file that is shipped with Service Manager may not contain the needed CA certificates and needs to be edited as described above.

When the root certificate file is saved, the following parameters must be entered into the Service Manager server `sm.ini`, if they do not already exist. These parameters identify the name of the root certificate or authority's certificate as well as the Service Manager server's keystore.

Parameter	Description
-truststoreFile	The TrustStore file to use to validate client certificates. Default to the cacerts in the RUN\jre\security directory.
-truststorePass	Identifies the password to the keystore file containing the external Web Servics's CA certificate. The pass phrase for the TrustStore file
-keystoreFile	Identifies the keystore file containing the Service Manager's server's certificate and private key. Server keystore
-keystorePass	Identifies the password to the keystore file containing the Service Manager's certificate and private key. Pass phrase for server keystore.

To enable the SSL encryption:

1. Stop the Service Manager server.
2. Open the sm.ini file with a text editor.
3. Add the following parameters and their values:
   a. keystoreFile
   b. keystorePass
   c. truststoreFile:cacerts
   d. truststorePass
4. Save sm.ini.
5. Restart the Service Manager sever.
6. Login to Service Manager with SysAdmin privileges.

7. Click **Tailoring > Web Services > Run WSDL to JS**.

8. Update the endpoint URL to the external Web Service to include the HTTPS protocol. For example: `https://remote_server.remote_domain.com:13445/remote_service.wsdl`

If the `https://<fully qualified server path>:<portnumber>/<Service>.wsdl` connection does not work after you make these changes, it is possible that the distinguished name (DN) used to create the certificate is not identical to the fully qualified server path in the URL. Check which DN the certificate is using by asking the provider of the certificate. If it is different from the fully qualified path used in the URL, request a new certificate where the DN matches the URL. If this cannot be done in a timely manner, the following workaround can be tested:

Go to the server's hosts file (which is located in etc/hosts on UNIX® systems, and located in c:\winnt\system32\drivers\etc\hosts on Windows systems). In the hosts file, add a line with the fully qualified name of the certificate and the IP address of the machine that runs the Web Service. For example:

mymachine.hp.com 10.2.5.77

where my"machine.hp.com" is the distinguished name (DN) of the certificate and 10.2.5.77 is the IP address for the server that hosts the Web Service.

Note: This is a temporary workaround, and not a permanent fix. Once the new certificate is issued, that certificate should be put into the root certificate file, and the entry in the hosts file should be removed.

Important: When you use SSL connections on high-frequency Web Services where many requests per second are made, performance is negatively impacted because an SSL handshake occurs for each SOAP request. The SSL handshake involves numerous TCP requests and responses as the two partners identify each other, negotiate the encryption algorithm, and perform other required tasks. To avoid this issue, ensure to use keep-alive connections. These will perform the handshake once and then SSL is set up for the length of the session.

Web Services connections through a firewall

If your Service Manager server is behind a firewall, you may need to configure a proxy server redirection to send and receive WSDL and SOAP requests. If your firewall uses the SOCKS protocol, then it can likely handle Web Services redirection requests transparently to the user. If your firewall does not recognize the SOCKS protocol, then you can install a dedicated redirector application for SOCKS traffic such as that generated by Web Services requests.

If you install a redirector application for your Web Services SOAP traffic, then you need to modify the URLs you use to connect to the remote Web Services. To download the remote WSDL, change the URL listed in the WSDL to JS wizard to point to the dedicated socket you have established for the remote Web Service. To send and receive SOAP messages to the Web Service, you can change the location object of your custom JavaScript to the dedicated socket you have established for the remote Web Service.

Example: dedicated socket connection

Define a dedicated socket on port 8888 to the Amazon Search Service using the following `proxyconnect` command of the `connect.c` application:

```
proxyconnect -p 8888 -S 192.168.1.254:1080
```

```
http://soap.amazon.com/onca/soap280
```

To obtain the WSDL for the Amazon Search Service through this example proxy connection, update the `WSDL to JS` URL to point to:

```
http://localhost:8888/soap/servlet/rpcrouter
```

To send to and to receive from the Amazon Search Service SOAP messages , you could update the custom calling script `AmazonSearchServiceTestwith` the following new line just after the `AmazonSearchService.ActorSearchRequest` class is initialized.

```
actorSearchRequest.location =
```

```
"http://localhost:8888/soap/servlet/rpcrouter"
```

Chapter 4

RESTful API

Service Manager also supports a REST API Framework. You can use the REST API Framework to support lightweight queries and operations on Service Manager data via a single URI. Using the REST API Framework you can create an application that can perform Create, Read, Update, and Delete actions on Service Manager objects.

The REST API Framework re-implements most of the functionality that the Service Manager SOAP implementation. Therefore, the REST API Framework uses the same actions on objects as the SOAP implementation, and the implementation methods are similar.

For more information on the available actions, see the Allowed Actions tab field definitions in the *Web Services Guide*.

Service Document

Service Manager supports the automatic generation of RESTful Service Documents by providing an HTML (text/html) representation.

RESTful Service Documents represent server-defined group of Collections used to initialize the process of creating and editing resources.

After **RESTful Enabled** is checked into **Web Service Configuration** for a Web service, its description will be generated.

At the top of the document, there is a group of all of Service Manager RESTful services. You can go to the detailed description of each service by clicking **Service Name**.

In the detail of the description, you can find the URI and supported HTTP Methods for each Resource Type. You can also find the supported actions' descriptions.

For example:

Change

Resource type	Supported Methods	URI
Resource Collection	GET/POST	http://localhost:14930/SM/9/rest/changes
Single Resource	GET/POST/PUT	http://localhost:14930/SM/9/rest/changes/{header,number}
Resource Attachment Collection	GET/POST/DELETE	http://localhost:14930/SM/9/rest/changes/{header,number}/attachments
Resource Single Attachment	GET/PUT/DELETE	http://localhost:14930/SM/9/rest/changes/{header,number}/attachments/{id}

Supported actions:

Action name	Supported Methods	URI
Update	POST	http://localhost:14930/SM/9/rest/changes/{header,number}/action/update
Retract	POST	http://localhost:14930/SM/9/rest/changes/{header,number}/action/retract
Reopen	POST	http://localhost:14930/SM/9/rest/changes/{header,number}/action/reopen
MoveToNextPhase	POST	http://localhost:14930/SM/9/rest/changes/{header,number}/action/movetonextphase
Deny	POST	http://localhost:14930/SM/9/rest/changes/{header,number}/action/deny
Close	POST	http://localhost:14930/SM/9/rest/changes/{header,number}/action/close
Approve	POST	http://localhost:14930/SM/9/rest/changes/{header,number}/action/approve
Create	POST	http://localhost:14930/SM/9/rest/changes/{header,number}/action/create

Consuming Service Manager RESTful API

A Service Manager Restful Web service can be consumed by a custom client or by an application that directly consumes Restful Web Services.

RESTful Syntax

A RESTful query allows you to send a request or execute an operation by sending a single Universal Resource Identifier (URI) space to Service Manager. In general, the format for the URI resembles the following:

- `http://{host}:{port}/{initialPath}`

This example uses the following placeholders:

- The {*host*} placeholder indicates the host or domain name on which the service is available.
- The {*port*} placeholder indicates the TCP port number on which the service is available.
- The {*initialPath*} placeholder indicates any initial path that is part of the URI for a given deployment. This might be the path to the deployment point on the given server. This value should start with **/SM/9/rest**.

Resource Types

The REST API Framework in Service Manager allows you to perform standard CRUD operations on resources. URI endpoints for resources are roughly divided into the following resource types:

- Resource Collection
- Resource Instance

- Resource with Actions
- Attachment Collection
- Attachment Instance

URI Structure

The following table describes the structure of URIs that can be used in the REST API Framework.

URI	Resource	Comments	Example
/	Service Document	This is the service document for an entire business service. Only the GET method is supported.	http://{host}:{port}/SM/9/rest
/{*resources*}	Resource Collection	This is the collection resources. Only the GET and POST methods are supported.	http://{host}:{port}/SM/9/rest/incidents
/{*resources*}/{*key*}	Resource Instance	This is an individual resource.	http://{host}:{port} /SM/9/rest/incidents/IM10001 If with multiple keys, it should be {key1}/{key2}/{key3}. If one of the key value is null, it should look like: {key1}/null/{key3}
/{*resources*}/{*key*} /attachments	Attachments Collection	This is the attachment list for an individual resource.	http:/{host}:{port} /SM/9/rest/incidents/IM10001/attachments
/{*resources*}/{*key*} /attachments/{*attachment-id*}	Attachment	This is an individual attachment.	http://{host}:{port} /SM/9/rest/incidents/IM10002/attachments/cid:51dd0b6d0002c0042075d798
/{*resources*}/{*key*}/action/{*action*}	Resource With Actions	Only the POST method is supported.	POST http://{host}:{port} /SM/9/rest/incidents/IM10134/action/reopen

This table uses the following placeholders:

- The {*resources*} placeholder represents the resource key of the individual object.
- The {*attachment-id*} placeholder represents the ID of the attachment for an individual resource.
- The {*key*} placeholder represents any unique key specified in the extaccess record.
- The {*action*} placeholder represents any action specified in the Allowed Actions tab of the External Access Definition. It must be lower case. For example, if the action in the Allowed Actions tab is "Reopen", the value of {action} should be "reopen".
- action, attachments and view are keywords that are used in the URI.

RESTful Authentication

The RESTful API framework supports the following authentication methods:

- HTTP Basic Authentication
- CAC (Common Access Card)
- TSO (Trusted Sign On)
- LW-SSO (Light Weight Single Sign On)

RESTful Commands

The REST API Framework supports the following HTTP commands:

- `GET`
- `POST`
- `PUT`
- `DELETE`

The functionality of these commands varies according to the type of object to which it is applied and the actions that are associated with that object. The following table shows example resources and illustrates how these commands are used:

Object	Example Commands	Result
Service Document	`GET /`	Returns all accessible URLs.

Query	`GET/<incidents>`	Returns all Incidents
	`GET/<incidents>/<id>`	Returns an Incident with the specified ID.
	`GET/<incidents>?query=<url-encoded-string>&sort=number:ascending`	Returns a subset of Incidents as specified by the URL encoded string, in ascending order.
Resources with Actions	`POST /<incidents>`	Creates an Incident
	`PUT /<incidents>/<id>`	Updates an specific Incident
	`POST /<incidents>/<id>/action/<action>`	Invokes a customized action on a specific Incident

RESTful Queries

The REST API Framework also supports several parameters to query resources to return different views of a resource or to filter for desired entries in a list. You can run a query on all resource types.

Example of a query:

- `/incidents?field1=value1&field2=value2 //Simple Query`
- `/incidents?query=<url-encoded-string>&sort=number:ascending  //Service Manager Native Query`

Notes:

- A query string must use HTML URL Encoding.
- Datetime fields must use ISO standard formats.

The following table describes the parameters you can use:

Parameter Name	Type	Description:

query	native sm query	Field name could be either "Caption" or "Field".
		Examples:
		`Category="incident" and (Title="Desktop screen out of order" or Title="Network logon failure")`
		`Category="incident" and (brief.description="Desktop screen out of order" or brief.description="Network logon failure")`

sort	string	Returns the collection members in sorted order according to the arguments specified. Arguments are listed in pairs where the first argument of the pair specifies the attribute name on which to base the sort, and the second argument of the pair indicates whether to sort ascending or descending. More than one attribute can be specified on the sort list. The attributes “ascending” and “descending” can be used as well. `sort={primaryField}:{ascending|descending}[,{secondaryField}:{ascending|descending}...]` For example, `sort=Urgency` or `sort=severity` or `sort=Urgency:ascending,field2:descending` By default, the sortorder is ascending.
start	int	Indicates the index of the member that the collection response representation begins with.
count	int	Indicates the number of collection members to be included in the response. The minimum value for this parameter is 1. By default, its behavior is to return all members. The behavior is the same as “view=summary” if “count=0”.

view	string	Represents a collection.
		Supported values:
		`view=summary`
		`view=condense (default)`
		`view=expand`
		summary: It returns the number of members if there are any, it does not return the actual members.
		condense: Returns the value of unique key field, it does not return the whole record. This is the default behavior if the view query parameter is not specified.
		expand: It returns all the fields defined in the extaccess record.

Resource Representations

The following topics describe the media types and supported commands for each resource.

Media Types for an Individual Resource

The following table describes the supported commands and media types for an individual resource.

Action	Supported?	Supported Request Media Types	Supported Response Media Types

GET	√	-	application/json
POST	√	application/json	application/json
PUT	√	application/json	application/json
DELETE	√	-	application/json

Resource Collection Media Types

The following table describes the supported commands and media types for a resource collection.

Action	Supported?	Supported Request Media Types	Supported Response Media Types
GET	√	-	application/json
POST	√	application/json	application/json
PUT	-	-	-
DELETE	-	-	-

Media Types for an individual attachment

The following table describes the supported commands and media types for an individual attachment .

Action	Supported?	Supported Request Media Types	Supported Response Media Types
GET	√	-	The attachment binary
POST	-	-	-
PUT	√	The attachment binary	application/json
DELETE	√	-	application/json

Resource Collection Media Types

The following table describes the supported commands and media types for an attachment collection.

Action	Supported?	Supported Request Media Types	Supported Response Media Types
GET	√	-	application/json

POST	√	The attachment binary	application/json
PUT	-	-	-
DELETE	√	-	application/json

Enable a Resource for REST

To enable a resource to use the RESTful API Framework, follow these steps:

1. Navigate to **Tailoring** > **Web Services** > **Web Service Configuration**.
2. If needed, enter the information in the **Service Name**, **Name**, or **Object Name** fields to specify the resource you want to enable for RESTful.
3. Click **Search**, and then select the appropriate resource from the list.
4. Under **External Access Definition**, click the **RESTful** tab.
5. Check the **RESTful Enabled?** check box.
6. Specify the following required fields:

Field	Function
Resource Collection Name:	This is the name of the Resource Collection. For example, you may specify the group of incidents from the probsummary table as "incidents".
Resource Name:	This is the name of the individual Resource. For example, you may specify that any individual incident from the probsummary table be referred to as an "Incident".
Unique Keys	This field specifies one or more fields that will function as a unique identifier for a Resource from the Resource Collection.

7. If needed, specify the following additional fields:

Field	Function
Resource Collection Action **POST:**	This field represents the default action for resource collection.

Resource Actions **POST:** **PUT:** **DELETE:**	This field specifies the action to take when an individual resource is part of a POST, PUT, DELETE command. These actions are specified in the **Allowed Actions** tab of the **External Access Definition**.

RESTful Capability Word

The REST API Framework introduces the following new capability word to Service Manager.

RESTful API

You MUST add this capability word to an user's operator record for a user to be able to log in and execute a RESTful API request.

Note: Existing OMi integration RESTful functions will not check it and keep unchanged. The rest actions go through document engine and follow the same permission process as normal client.

HTTP Header

Request Header

`Autorization:`

`Keep-Alive:`

`Connnection:`

`Accept-Language:`

`Content-Type:` (Required for attachment POST/PUT action)

`Content-Disposition:` (Required for attachment POST/PUT action. Value is attachment;filename=filename; Semicolon is reserved for character separator, so it is not allowed in file name.)

Note: Accept-Language is used for I10n.

Response Header

`code&msg: 200, 201, 400, 401, 404, 500`

`Content-Type:`

`Content-Length:`

HTTP Response Codes

Unless otherwise specified, these HTTP status codes are used:

Code	Cause
200	Successful operation. Viewing a list or detail page will return this code if no error occurred.
201	Successful POST operations. Returned after a successful create or update of a object.
400	Bad request. This response code may indicate that the URI was formed incorrectly.
401	Unauthorized operations.
404	Page not found. The page or resource does not exist.
500	Internal server error. This response code may indicate a bug. Please contact HP Support.

See Also

Network Working Group RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1
Network Working Group RFC 2616 Section 10: Status Code Definitions

OOB Resource Reference Example

The following topics provide detailed information on how specific resources are published (for example, Incident):

Web Service: Incident

External Access Definition:

Field	Value
Service Name:	IncidentManagement
Name:	probsummary
Object Name:	Incident

Default RESTful settings

Field	Value	

RESTful Enabled:	checked	
Attachments Enabled:	checked	
Resource Collection Name:	incidents	
Resource Name:	Incident	
Unique Keys:	number	

HTTP Resource Collection Actions

Command	Action	Result
GET		Retrieves a list of Incidents
POST	Create	Creates a new collection of resources

HTTP Resource Actions

Command	Action	Result
GET	-	Retrieves an individual Incident.
PUT:	Update	Saves the changes to the Incident record.
POST:	Update	Saves the changes to the Incident record.
DELETE:	-	

Samples

Return sample of single incident query


```
{"Messages":[],
 "ReturnCode":0,
 "Incident":{
   "UpdatedBy":"falcon",
   "Status":"Closed",
   "SLAAgreementID":168,
   "Urgency":"3",
   "Area":"failure",
   "OpenTime":"2007-08-31T20:21:00+00:00",
   "Location":"advantage/North America",
   "ClosedTime":"2007-09-01T01:13:00+00:00",
   "Title":"Printer malfunction",
   "Subarea":"job failed",
   "Solution":["Reset printer queue."],
   "ClosedBy":"Incident.Analyst",
   "OpenedBy":"Servicedesk.Manager",
   "IncidentID":"IM10001",
   "Assignee":"Incident.Analyst",
```

```
   "Company":"advantage",
   "Description":["Printjob keeps pending."],
   "TicketOwner":"Servicedesk.Manager",
   "ProblemType":"incident",
   "AssignmentGroup":"Office Supplies (North America)",
   "UpdatedTime":"2008-08-04T12:53:21+00:00",
   "Service":"Printing (North America)",
   "Impact":"4",
   "ClosureCode":"Solved by Workaround",
   "Category":"incident",
   "AffectedCI":"adv-nam-printer-hr-5550"
   }
}
```

Note: "Incident" is the Resource Name as defined in the extaccess record.

Return sample of incident list query

```
{
  "@totalcount":136,
  "@start":1,
  "@count":10,
  "Messages":[],
  "content":[
     {"Incident":{"IncidentID":"IM10001"}},
     {"Incident":{"IncidentID":"IM10002"}},
     {"Incident":{"IncidentID":"IM10003"}},
     {"Incident":{"IncidentID":"IM10004"}},
     {"Incident":{"IncidentID":"IM10005"}},
     {"Incident":{"IncidentID":"IM10006"}},
     {"Incident":{"IncidentID":"IM10007"}},
     {"Incident":{"IncidentID":"IM10008"}},
     {"Incident":{"IncidentID":"IM10009"}},
     {"Incident":{"IncidentID":"IM10010"}}
  ],
  "ReturnCode":0,
  "ResourceName":"Incident"
}
```

Request sample of creating an incident

```
{
       "Incident":
       {
         "AffectedCI" :"adv-nam-server-mail",
      "AlertStatus" : "updated",
      "Area" :"failure",
      "Assignee" : "Incident.Analyst",
```

```
        "AssignmentGroup": "Network",
        "Category" : "incident",
        "ClosureCode" : "Solved by Workaround",
        "Company" : "advantage",
        "Contact" : "FALCON, JENNIFER",
        "ContactFirstName" :"FALCON",
        "ContactLastName" : "JENNIFER",
        "Description" : [ "test" ],
        "Impact" : "2",
        "JournalUpdates":
             [ "08/04/08 12:54:14 US/Mountain (falcon):",
            "test",
            "08/04/08 12:54:14 US/Mountain (falcon):",
            "test"
          ],
        "Location":"advantage/North America",
        "OpenTime" : "2007-09-02T07:51:00+00:00",
        "OpenedBy":"Jurr.Fleijs",
        "ProblemType" : "incident",
        "ResolutionFixType" : "incident",
        "SLAAgreementID" : 168,
        "Service" : "E-mail / Webmail (North America)",
        "SiteCategory" : "incident",
        "Solution" : ["Solution by rest api"],
        "Status" : "Work In Progress",
        "Subarea" :"function or feature not working",
        "TicketOwner" : "Jurr.Fleijs",
        "Title" : "test",
        "UpdatedBy" : "problem",
        "UpdatedTime" : "2008-08-04T12:54:26+00:00",
        "Urgency" : "3",
        "UserPriority": "3 - Average",
        "explanation" : ["test"],
        "folder" :"advantage"
         }
}
```

Chapter 5

Troubleshooting

The combination of debugging tools and information gathered from faults usually helps you find the root cause of an issue easily. Unfortunately, not all Web Services give sufficient fault messages, which makes debugging the issue more challenging.

Understanding the return codes provided by Web Services

Currently the status attribute always contains either "SUCCESS" or "FAILURE." A best practice is to check either the status to see if it has the value “SUCCESS” or to check the return code to see if it is zero. All other values equate to FAILURE. The value of the message attribute is a string which corresponds to the return code value. The Service Manager server global JavaScript method called RCtoString()will convert a particular integer return code value to the corresponding message text. The following are the currently defined values:

Value	Definition
0	Success
1	Bad Length
2	Bad Serial Number
3	Resource Unavailable
4	Unable to Terminate
5	Resource Not Available
6	Resource Expired
7	Specified Name Not Found
9	No (more) records found
10	No messages
11	No query words
12	No stop words
13	No string
14	No such word
15	Not enough memory

Value	Definition
16	Already exists
17	Shutdown error
18	Stop words not found
19	Too many documents
20	Unable to open file for output
21	Waiting for resource
22	Word length too long
23	Duplicate file system
24	Duplicate IPC Key
25	IPC Key Not Found
27	Wrong owner
28	Not authorized
29	Invalid Userid Specified
30	Invalid Password Specified
31	New Password is Invalid
32	Password Expired
33	Authority Revoked
34	Max Attempts to Login Exceeded
35	Max Number of Logins Exceeded
36	Invalid terminal for user
37	Invalid Authorization Code
38	Maximum users exceeded
39	Named user already logged in
40	Not a named user and no floating users available
41	User Already Logged In
42	Forced synchronization
43	IR read count mismatch

Value	Definition
44	Seek error
45	24x7: DBLOG error
46	Open error
47	Error closing remote file
48	Duplicate key
49	Null key
50	All null keys
51	Record modified since last retrieved
52	Record deleted since last retrieved
53	Trigger Error
54	Not supported
55	Record no longer qualifies
56	Query timed out
57	Unable to delete file
58	Partially-keyed or non-keyed query
59	Error occurred in parsing
60	Shared memory version mismatch
61	Distributed Lock Manager cannot lock item
62	Refresh not needed
63	Userid expired
64	Userid inactive
65	SQL conversion skipped for this file
66	Query could not be parsed
67	file could not be opened
68	User is not located in LDAP
69	User is not allowed to use ODBC driver
70	Invalid SOAPaction / unrecognized application action

Value	Definition
71	Validation failed
72	User is not allowed to use SOAP API

Example of a failure return code and message

The following is an example of a failure return code and message:

```
message="No (more) records found" returnCode="9" status="FAILURE"
```

Detailed return codes from Document Engine

The System Administrator can manipulate the detailed return code by setting the value of `$L.exit` in the Document Engine process's final expressions to one of the following:

Action or Error Situation	$L.exit value
record has changed since it was selected	changed
cancel processing the record	cancel
record is locked	locked
Request failed validation	bad.val
record was deleted since it was selected	deleted
exit processing	exit
normal exit	normal
Record should get unlocked	unlock
Sets exit value to menu to return to the menu	menu
Record was added, screen will be refreshed	added
Processing will restart – starting with init of file variable	restart
Processing will proceed with a new state record	newstate
Displayed records will be refreshed	refresh
Displayed joinfile records will be refreshed	refreshjoinfile
Category changes	newcat
Position in record list will be changed	reposition

Record will be reset to original values	resetrec
Mode will be set to close and close processing will start	closestate
Restart processing starting with init of file variable	restart
Mode will be set to add, which goes into the open state	openstate
Initializing values to add record	setupadd
An undefined action was passed to the document engine	invalid.action
User is not authorized for this action	no auth

Troubleshooting SOAP API

This section lists common issues of SOAP API and describes the methods to troubleshoot.

Debugging

Three parameters are most frequently used: debughttp:1, RTM:3, and debugdbquery:999. It may be useful to use the msglog:1 parameter to have all messages written to the sm.log as well, especially for Connect-It Web Services integrations. As a best practice, put these debug parameters on the dedicated Web Services port such as shown below:

```
sm -httpPort:13087 –debugnode –debughttp:1
```

The debughttp parameter

Add the debughttp in the server `sm.ini` file or in the dedicated servlet container line of the sm.cfg file, restart the Service Manager server and rerun the Web service application to invoke the debugging parameter.

For consuming Web Services, the debughttp parameter writes to two files in the Service Manager server log directory, http.log and writes additional information into the sm.log file.

An excerpt of the http.log file follows. (To determine which areas of the log file are for the Web service call, search for “sm7server/ws”. Regular client communication uses SOAP UI instead.)

```
POST /sm7webserver/ws HTTP/1.1
content-type: text/xml;charset=UTF-8
soapaction: "EnableNewEmployee"
user-agent: Jakarta Commons-HttpClient/3.0.1
host: <server>:<port>
content-length: 2033

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmln
s:pws="http://servicecenter.peregrine.com/PWS" xmlns:com=
"http://servicecenter.peregrine.com/PWS/Common">
   <soapenv:Header/>
   <soapenv:Body>
```

```
      <pws:EnableNewEmployeeRequestQuoteRequest attachmentInfo="?
         attachmentData="?" ignoreEmptyElements="true">
         <pws:model query="?">
            <pws:keys query="?">
               <!--Optional:-->
               <pws:number type="String" mandatory="?"
                  readonly="?">?</pws:number>
            </pws:keys>
            <pws:instance query="?" uniquequery="?" recordid="?">
               <!--Optional:-->
               <pws:Priority type="String" mandatory="?"
                  readonly="?">?</pws:Priority>
               <!--Optional:-->
               <pws:Reason type="String" mandatory="?" readonly="?"
                  ?</pws:Reason>
               <!--Optional:-->
               <pws:RequestingDepartment type="String" mandatory="?"
                 readonly="?">?</pws:RequestingDepartment>
               <!--Optional:-->
               <pws:Requestor type="String" mandatory="?" readonly="?">?
                   </pws:Requestor>
               <!--Optional:-->
               <pws:Location type="String" mandatory="?" readonly="?">?
                     </pws:Location>
               <!--Optional:-->
               <pws:HireType type="String" mandatory="?" readonly="?">?
                  </pws:HireType>
               <!--Optional:-->
               <pws:attachments>
                  <!--Zero or more repetitions:-->
                  <com:attachment href="?" contentId="?" action="?
                      name="?" type="?" len="attachmentType="?"/>
               </pws:attachments>
            </pws:instance>
            <!--Optional:-->
            <pws:messages>
               <!--1 or more repetitions:-->
               <com:message type="String" mandatory="?" readonly="?"
                 severity="?" module="?">?</com:message>
            </pws:messages>
         </pws:model>
      </pws:EnableNewEmployeeRequestQuoteRequest>
   </soapenv:Body>
</soapenv:Envelope>

HTTP/1.1 401
Set-Cookie: JSESSIONID=94DCC5F90495E0202B84EFB1F998195A;
 Path=/sc62server
```

```
WWW-Authenticate: Basic realm="CASM"
Connection: close
Content-Type: text/html;charset=utf-8
Content-Length: 40
Date: Wed, 21 May 2008 17:16:05 GMT
```

Interpreting the http.log

The http.log may contain encoded or compressed messages from the Windows or Web Client communication. See below to turn off SOAP compression and FastInfoset encoding. Web Services communications are not encoded nor compressed.

The http.log contains information on all Processes / Threads that connect to the traced servlets. If more than one Process / Thread is traced, information in the log may overlap from different sessions, resulting in multiple POST or GET messages together rather than the POST – GET pair you would expect.

To turn off FastInfoset and compression for the clients, follow these steps:

1. On the shortcut starting the Windows client, add –DFastInfoset=false to the target.
2. The command line arguments for the Service Manager Windows client are:
   - -vmargs -option1 -option2
3. As an example, if -DFastInfoset=false is the only command line option ensure your command line is:
   - -vmargs -DFastInfoset=false
4. Turn off Compress SOAP Messages on the File - Connect – Connections screen's Advanced tab.
5. For the Web client, turn off FastInfoset by setting the JAVA Option -DFastInfoset=false. As an example, to turn off FastInfoset on Tomcat 5.5 or later, follow these steps:
   a. Go to the Apache Tomcat x.x folder on the Start Menu.
   b. Start the Configure Tomcat application.
   c. Select the Java tab in the configuration dialog.
6. Turn off the Compress SOAP Messages by editing the ...\WEB-INF\web.xml file

```
<!-- Compress network communication between the application server
         and the HP Service Manager server -->
    <init-param>
      <param-name>compress_soap</param-name>
      <param-value>false</param-value>
    </init-param>
```

RTM:3 and debugdbquery:999

Sometimes the Web Service itself is working correctly, but actions performed by the Document Engine within Service Manager are not performing as expected. (Error Message: soap_serve - Caught XML API exception scxmlapi(19) - Doc Engine call failed with cc -1)The RTM:3 and debugdbquery:999 parameters can expose such issues that occur within the application layer of Service Manager. The debugging information produced by these parameters can be found in the sm.log file in the Service Manager server RUN directory. It is not necessary to restart Service Manager to activate these debug parameters. Reconnecting the Web Service to Service Manager triggers the use of these debug parameters.

The allowwsdlretrieval parameter

This parameter is used to allow retrieval of the WSDL without having the SOAP UI license.

Error messages

Error Message: soap_serve - Caught XML API exception scxmlapi(19) - Doc Engine call failed with cc -1

Service Manager publisher:

This error message is issued either when the Document Engine did not attempt to write the record, because the Process called via extaccess does not perform a save operation, or if a validation failed and the save could not be performed. To fix this issue, ensure that the Process called does perform an action that adds or updates a database record. If it does, add the msglog:1 parameter to the sm.ini and rerun the Web Service. Check the sm.log file for any validation error messages and then either pass the required information or change the extaccess record to add any missing required fields to it. If your are still unsure what is the root cause of the issue after this, add `RTM:3` and `debugdbquery:999` to the `sm.ini` and retest the Web service operation. If your are still unsure what is the root cause of the issue after this, add RTM:3 and debugdbquery:999 to the `sm.ini` file and retest the Web service operation.

Error Message: Invalid or missing file name in XML request

Service Manager publisher or consumer:

Complete Error Message: <SOAP-ENV:Fault><faultcode>SOAP-ENV:Server</faultcode><faultstring>scxmlapi(16) - Invalid or missing file name in XML request</faultstring><detail><appFaultCode>16</appFaultCode><appFaultString>scxmlapi(16) - Invalid or missing file name in XML request</appFaultString></detail></SOAP-ENV:Fault>

This error message is issued if the binaries cannot successfully retrieve the name for the Object to access from the extaccess file. This issue occurs most often when the Object name is in “CamelCase” notation. To prevent this issue, do not use “CamelCase” notation (where the name contains compound words or phrases that are joined without spaces, and each word is capitalized

within the name.) in the Object Name in the extaccess file. As a best practice, use the name of the dbdicts as the Object Name as well.

If the underlying cause is not the camel case notation, you can modify the SOAP body by adding **filename="<filename>"** to work around this issue. For example:

<soap:Body>

<CreateProblemRequest filename="rootcause"

xmlns="http://<server>:<port>/SM/7">

Error Message: getType() in com.peregrine.webservice.ComputerInstanceTypeDevice cannot override getType() in com.peregrine.webservice.common.StructureType; attempting to use incompatible return type

The ConfigurationManagement WSDL is made up of the device extaccess record in addition to a number of device attribute files (such as computer). The following errors occur when you set the API Caption for the type field in the device extaccess record to "type" or "Type" and then attempt to compile the WSDL using Apache Ant:

```
build_java:
    [javac] Compiling 114 source files to C:\Service
Manager\server\webservices\sample\AxisSample\build
    [javac] C:\Service
Manager\server\webservices\sample\AxisSample\src\com\
peregrine\webservice\ComputerInstanceTypeDevice.java:225: getType() in
com.peregrine.webservice.ComputerInstanceTypeDevice cannot override
getType() in com.peregrine.webservice.common.StructureType; attempting to
use incompatible return type
    [javac] found   : com.peregrine.webservice.common.StringType
    [javac] required: java.lang.String
    [javac]     public com.peregrine.webservice.common.StringType
    getType() {
    [javac]
    [javac] 1 error

BUILD FAILED
C:\Service Manager\server\webservices\sample\AxisSample\
build.xml:184: Compile
 failed; see the compiler error output for details.
```

To avoid this or similar errors, make sure that the name is valid and does not conflict with previously defined names when you set up alias names ("API Captions"). All of the common.xsd definitions for data types such as StructureType, ArrayType, have a type attribute, for which Axis manufactures a getType Java method. When it generates a getType method for this new type property/field, those two methods conflict. It does not matter whether you specify "type" or "Type" because Axis uses camel-case naming conventions for its generated method names. Whenever an API caption can

cause a conflict with a pre-existing function, change it to be something unique; in this case, for example, make the API caption `CIType`.

Failure of the WSDL2JS utility

The WSDL to JS utility executes the SOAP JavaScript record. It reads the provided WSDL with all incorporated schema definitions and creates or updates a JavaScript record in the Service Manager ScriptLibrary table with the objects and methods that can be used for this web service. The generated code can then be called from a custom written JavaScript to consume the external Web Service. If the code generated by WSDL2JS is incorrect or incomplete, contact Customer Support for a new unload of the utility. If the issue is still not solved with the latest version of the WSDL2JS utility, send the WSDL and all imported / invoked xsd schemas to Customer Support together with an unload of the generated JavaScript record. It is very important that the location of the xsd files that are imported or invoked from the WSDL is set correctly, otherwise the WSDL2JS utility will generate incomplete code.

Important: Every time the SOAP JavaScript record is changed, all existing Web Services generated JavaScripts have to be re-generated by re-running the WSDL to JS utility and all invoking JavaScripts have to be re-compiled.

Testing your WSDL with a SOAP UI

To read the WSDL, go to **File > New Project** and enter a project name as well as the initial WSDL location and click on OK. The list of methods will be displayed on the left, the request is in the middle, the response on the right.

To pass authentication information, enter the Username and Password. If the password is blank, enter information and then remove the information again.

Note: SoapUI fills in each field value with a question mark symbol. For correct processing, remove these ? before submitting the request.

Running Web Services on a dedicated port (servlet)

To create a separate servlet within a horizontally or vertically scaled Service Manager system, add the debugnode parameter to the dedicated servlet container. The debugnode parameter stops the load balancer from distributing client load to this node. This can be used to set up a dedicated servlet for tracing and logging without adding an uncontrollable amount of load to that servlet. Another use is to create a dedicated servlet for a special purpose within the scaled solution. As an example, refer to the following `sm.cfg` file:

```
#load Balancer Port
sm -loadBalancer -httpPort:13080
#Ports for loadBalanced Connections
sm -httpPort:13081 -httpsPort:13082
sm -httpPort:13083 -httpsPort:13084
sm -httpPort:13085 -httpsPort:13086
#Port for Web Services
sm -httpPort:13087 -debugnode
```

Current limitations of running Web Services through the load balancer:

- The HTTP 307 redirect is not fully compliant with the specifications which can affect Web Services integrations through the Service Manager Load Balancer. The workaround is to connect directly to one of the Service Manager Application server servlets.
- Web Services through the Service Manager Load Balancer are not possible when SSL is enabled on the server.
- Web Services through the Service Manager Load Balancer are not possible for Web Services clients that can't handle a redirect.

Troubleshooting a Web service that is behind a closed firewall

Sometimes it is necessary to troubleshoot a Web Service that is not available. To do so, we can check whether a WSDL file that is stored on the local machine works using test data.

Step 1: Test the WSDL2JS

1. Store the WSDL file locally on the server.
2. Start the WSDL to JS utility and enter, file://<fully qualified path to the file>.wsdl
3. Click **Proceed**.

If the JavaScript file for the Web service is generated without error messages and ends with:

```
lib.SOAP.init();
/// End ----------------
```

… then the WSDL to JS program was able to interpret the WSDL file.

To correctly write the JavaScript functions to call this Web service and generated JavaScript, check the generated JavaScript for the function you want to use, in this case:

```
this.SOAPOperations[ "RetrieveIncident" ]
 = new soap_Operation( "RetrieveIncident", "Retrieve","document",

  "RetrieveIncidentRequest",
  "RetrieveIncidentResponse" );
```

The request can be found within that line and refers to the request function further down:

```
function RetrieveIncidentRequest(  )
{
  this.$$nsPrefix = "ns";
  this.$$attributes = new Array();
  this.$$xmlNames = new Array();
```

```
this.$$objNames = new Array();
this.$$minOccurs = new Array();
this.getName = getName;
this.getXmlName = getXmlName;
this.setContent = setContent;
this.addContent = addContent;
this.getContent = getContent;
this.isFault = isFault;
this.$$elementChildren = new Array();
this.$$name = "RetrieveIncidentRequest";
this.$$xmlNames[ "RetrieveIncidentRequest" ] = "ns:RetrieveIncidentRequest";
this.attachmentInfo = new Boolean();
this.$$attributes.push( "attachmentInfo" );
this.attachmentData = new Boolean();
this.$$attributes.push( "attachmentData" );
this.ignoreEmptyElements = new Boolean("true");
this.$$attributes.push( "ignoreEmptyElements" );
this.xmlns = new String("http://servicecenter.peregrine.com/PWS");
this.$$attributes.push( "xmlns" );
this.model = new RetrieveIncidentRequest_IncidentModelType();
this.$$elementChildren.push( "model" );
```

Step 2: Test the request

Once the automatically generated JavaScript code has been saved, write a calling JavaScript to execute the Web Service. The following is a simple example code for IncidentManagement record retrieval:

```
function RetrieveIncident(incident_id)
{
       var IncMgmtSvc = new system.library.IncidentManagement.
IncidentManagement();
       IncMgmtSvc.user="falcon"

       var retrieveReq = new system.library.IncidentManagement.
RetrieveIncidentRequest();

       retrieveReq.model.keys.IncidentID.setValue(incident_id);

       try
       {
                var retrieveResp = IncMgmtSvc.invoke(retrieveReq);
               if ( retrieveResp.isFault() )
               {
                       throw( "SOAP Fault: " + retrieveResp.
faultstring.getValue() );
               }
```

```
                return retrieveResp.model.instance;
        }
        catch( err )
        {
          return( "Error! " + err );
        }
}

retVal=RetrieveIncident("IM1001");

print("Testing the result " + retVal.IncidentID.getValue())
```

1. To test the request, enter `debughttp` in the sm.ini file and restart the server and client.
2. If the file `http.log` exists in the server's `RUN` directory, remove it or remove its contents so that there will be a fresh file to read.
3. Go into the calling JavaScript and click **Execute**. You will most likely get an error message because the Web service you are trying to reach is not available.
4. After the execution is complete, open the `http.log` file and search for the following:

```
POST /sc62server/ws HTTP/1.1
accept: application/fastinfoset, text/xml, text/html,
image/gif, image/jpeg, *; q=.2, */*; q=.2
authorization: Basic ZmFsY29uOg==
soapaction: Retrieve
connection: Close
content-type: text/xml; charset=utf-8
content-length: 841
cache-control: no-cache
pragma: no-cache
user-agent: Java/1.5.0_08
host: <server>:<port>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/
envelope/"
 xmlns:ns0="http://servername.port_number/SM/7/service_name.wsdl"
 xmlns:ns1="http://schemas.xmlsoap.org/wsdl/http/"
 xmlns:ns2="http://schemas.xmlsoap.org/wsdl/mime/"
 xmlns:ns3="http://servicecenter.peregrine.com/PWS
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-
 instance"><soap:Body><ns3:RetrieveIncidentRequest attachmentData=
 "false" attachmentInfo="false"
 ignoreEmptyElements="true">
 <ns3:model><ns3:keys><ns3:IncidentID mandatory="false"
  readonly="false" type="String">IM1001</ns3:IncidentID>
</ns3:keys><ns3:instance><ns3:IncidentDescription
```

```
 type="Array"/><ns3:Resolution type="Array"/>
<ns3:JournalUpdates
  type="Array"/>
<ns3:Solution type="Array"/></ns3:instance>
</ns3:model></ns3:RetrieveIncidentRequest></soap:Body></soap:Envelope>

HTTP/1.1 200
Set-Cookie: JSESSIONID=0405ED23EFF6C9A3874F77796FE4210D;
Path=/sc62server
Connection: close
Connection: close
Content-Type: application/fastinfoset;charset=utf-8
Content-Length: 1323
Date: Wed, 04 Jun 2008 22:09:56 GMT
Connection: close
Connection: closeSet-Cookie: SessionId=16.95.106.150:3487;
Version=1
```

5. You can copy the bold area into an XML editor such as Altova® XMLSpy® and check whether it is correct XML. If it is, then the request is deemed to be successful (which does not necessarily mean that it will return data).

Another method to check the request and response is to run the request through a tool such as tcpmon. To do so, start tcpmon, enter the server name and port of the receiving Web service and connect the invoking JavaScript to tcpmon. Both the request and the response are visible in the tcpmon screen and can be analyzed in an XML editor as well.

Step 3: Test the response

After the request has been submitted successfully, test the response to the request, which is written to the http.logfile. Look for the following text. (The section that is bolded indicates that this is the response message):

```
SOAP-ENV (http://schemas.xmlsoap.org/soap/envelope/
<Envelope><Body>http://servicecenter.peregrine.com/PWSÏ cmn,
http://servicecenter.peregrine.com/PWS/CommonÏ xsd¬http:
//www.w3.org/2001/XMLSchemaÏ xsi(http://www.w3.org/2001/
XMLSchema-instanceð=, RetrieveIncidentResponsex message No
more) records foundx   returnCode@9x schemaRevisionDate
2008-05-30x schemaRevisionLevel@1x status
FAILURE{„… schemaLocation dhttp://servicecenter.peregrine.
com/PWS http://<server>:<port>/sc62server/ws/Incident.xsdð=,
model=, keys}, IncidentIDx typeEStringð' IM1001ÿ},
instancexrecordid  IM1001 - x
uniquequery  number="IM1001"ðE ,ð ÿÿÿðà
< #document8ÏSOAP-ENV(http://schemas.xmlsoap.org/soap/
envelope/ð?  Envelope?   BodyxÍ%http://servicecenter.
peregrine.com/PWSÏ cmn,http://servicecenter.peregrine.com/
PWS/CommonÏ xsd¬http://www.w3.org/2001/XMLSchemaÏ xsi(http://
```

```
www.w3.org/2001/XMLSchema-instanceð=,
RetrieveIncidentResponsex message No (more) records foundx
returnCode@9x schemaRevisionDate  2008-05-30x
schemaRevisionLevel@1x status FAILURE{„… schemaLocation
dhttp://servicecenter.peregrine.com/PWS http://geist8440.
americas.hpqcorp.net:13701/sc62server/ws/Incident.xsdð=,
model=, keys}, IncidentIDx typeEStringð’ IM1001ÿ},
instancexrecordid  IM1001 - x
uniquequery  number="IM1001"ðF ,ð ÿÿÿÿ

HTTP/1.1 200
Keep-Alive: timeout=1200000, max=1000
Connection: Keep-Alive
Pragma: requestnum="185"
Content-Encoding: gzip
Content-Type: application/fastinfoset;charset=utf-8
Transfer-Encoding: chunked
Date: Wed, 04 Jun 2008 22:09:56 GMT
```

1. Copy the section mentioned above from the http.log file into a text file and assign it a name such as responsetest.xml. If you used tcpmon to get the information, you can use the XML response as is. If it came from the http.log, you will need to modify the special characters in the log to correct XML syntax.

2. Change the calling JavaScript to override the invoke function to read and interpret the contents of the responsetest.xml file. The following is the section of the code needed to do that.

```
// Temporarily override the "invoke" function to replace it
with
// a function which reads an XML response from a file
<ServiceObject>.invoke = function( ) {
var resultObj = new Object();
resultObj.responseObj = null;
var resultXML = new XML();
resultXML.setContent( "c:\\<path>\\<responsetest.xml>",
true );
try
{
lib.SOAP.deserialize( "<name of the generated JavaScript>",
resultXML.getDocumentElement(), resultObj );
}
catch( e )
{
print( "Error deserializing response: " + e.toString() );
return null;
}
try
{
```

```
this.soapEnvelope = resultObj.soap_Envelope;
this.soapBody = resultObj.soap_Envelope.soap_Body;
if ( this.soapEnvelope.soap_Header != undefined )
{
this.soapHeader = this.soapEnvelope.soap_Header;
}
else
this.soapHeader = null;
return resultObj.soap_Envelope.soap_Body.getContent();
}
catch( e )
{
print( "Error extracting Response Object: " + e.toString() );
return null;
}
}
```

3. Change the line of the calling JavaScript that invokes the Web Service from <ServiceObject>.invoke to simply invoke to call the invoke function defined within that calling JavaScript.

4. Click **Execute** to run this modified JavaScript. If it finishes without errors, the response is deemed successful.

If any of the above tests fail to complete, contact HP Service Manager Customer Support and provide the WSDL file, the request and response xml text files with any error messages, and the `sm.log` and `http.log` files with debughttp turned on.

Max sessions exceeded in Web Services

If a Web Services request contains "connection: keep-alive" or it uses HTTP/1.1 without a connection header, the Service Manager server will keep the session alive for a predefined interval that is defined by setting the "webservices_sessiontimeout" parameter in the `sm.ini` file. If a Web Services client does not reuse the session for subsequent requests by providing valid headers, the Service Manager server creates a new session for each subsequent request and quickly run out of available sessions.

To avoid running out of available sessions, there are two options to consider:

Option A: Set the HTTP header "connection:closed" so that the Service Managerserver will not keep a Web Services session open after the current request is finished.

Option B: Utilize the Web Services session persistence by doing one of the following to reuse the existing Web Services session on theService Managerserver.

1. Use `connection: keep-alive`. If the connection header is missing, it defaults to "keep-alive" for HTTP/1.1.

2. The Web Services client needs to supply a session cookie with the same user log-in information that created the session.

Note: Even with Web Services session persistence, each SOAP API request is stateless, so that requests are handled independently between one another.

Troubleshooting HTTP socket connections

The HP Service Manager server attempts to keep an HTTP socket connection open as long as possible, but the protocol requires that it must close if the server returns a SOAP fault. If there is no successful authentication, it must return a SOAP fault.

Redirected ports

To ensure the client has the correct hostname and port number, a SOAP client application can direct requests to the TCP port number used by the `sm -httpPort` instance, but must be able to recognize SOAP header values in the initial response:

- `redirectServerHost`
- `redirectServerPort`

The server returns these SOAP header values identifying the dynamically allocated TCP host and port number for the spawned process. During the client/server session, subsequent SOAP requests must be directed to the same hostname and port identified in the initial response.

TCP ECONNRESET messages

If a client/server connection using a spawned child thread terminates, the `sm -httpPort` child thread receives a TCP ECONNRESET message. The child thread responds to this by self-terminating to ensure that orphaned child thread does not collect on the server. However, poorly-designed client applications, or other third-party SOAP tools, that do not gracefully close a connection could cause the server process to see a TCP ECONNRESET message, and that also terminates the server thread.

Debugging SOAP errors

The best practice for troubleshooting SOAP errors is to start a new client connection process with a dedicated log file associated with it. Opening a new client connection process allows you to isolate any faulty client traffic from your regular client traffic.

1. To set up your system to debug SOAP traffic, do one of the following:

- Start HTTP debugging for the entire system. Type the following command on a single line in the `sm.ini` file and then save the file.

  ```
  debughttp:1
  ```

  Debug parameters in the `sm.ini` file affect all Service Manager processes and the log files

record all send/receive messages. This method is not recommended for a busy server however, since you have to restart the server for the debugging parameter to take effect.

- Start a separate client connection process to troubleshoot your SOAP errors. Type the following command in the operating system command line:

  ```
  sm -httpPort:unique portnumber -sslConnector:0 -debughttp:1
  -log:../logs/debug.log
  ```

  where

  `-httpPort` identifies a port where Web Services clients can connect
  `-log` defines a path to store the logs for this process

  Normally, all client connection processes for a particular Service Manager installation use the parameter values listed in the `sm.ini` file. This means that all client connections share the same log file specified in the `sm.ini` file. By starting a new client connection process with a different log parameter value, you can isolate the logs for a particular group of clients. Choose a port number that is not likely to be used by any other process.

2. Recreate the error.
3. Review the http.log, sm.log and log files from the Web Services consumer or publisher that Service Manager is communicating with and server, and client log files for information about the SOAP error. The HTTP log is in the server's RUN folder. The server logs are in the path you specified with the log parameter. The client logs are located in the following paths:

- For a Web Services client, see your Web Services client log
- For a Web client, see the log filed specified in the `log.properties` file on the web tier system
- For a Windows client, see the `.log` on the Windows client system

SOAP messages: Debugging HTTP traffic problems

If, after reviewing the client logs, you discover that there is an error in the HTTP transfer of SOAP messages, you must manually enable the HTTP debugging option on the Service Manager server. This option allows you to trace all HTTP and SOAP messages between the Service Manager server and client. You can trace HTTP traffic in one of two ways.

- Trace all HTTP connections to the server
- Trace a dedicated connection to the server

To review all the HTTP traffic, you can enable the `debughttp` parameter from the Service Manager initialization file (`sm.ini`) file. This causes the server to record all messages sent from and to the server to the following log files.

- logs\sm.log
- RUN\TEST.log

This method of debugging SOAP messages traces all Service Manager processes, but significantly reduces system performance because the log files the server produces contain all HTTP traffic, including HTTP headers and attachments. For this reason it is recommended that you not enable this parameter on production systems, but rather in test environments only.

To review HTTP traffic use the -httpPort paremeter as in the sample below. In addition, you can create a dedicated log file for this connection. By starting a new Service Manager process and specifying a separate log parameter, you reduce the amount of system resources needed to produce debugging output. For example, you can enter the following command from the server OS command line to create a dedicated servlet and log file.

```
sm -httpPort:portnumber -debughttp:1 -log:../logs/debug.log
```

For *portnumber*, type a communications port number on which you want the server to use for SOAP requests. You can use the `-log` parameter to define a path to any log file you want.

SOAP messages: Debugging problems with RAD applications

If your review of the client logs reveals potential problems in the RAD applications, you can enable the logging of RAD application messages by adding the `rtm` startup parameter to the Service Manager initialization file. This parameter causes the server to record all application-generated messages to `sm.log` file. For example, to receive detailed information about the RAD applications type the following command into the Service Manager initialization file (sm.ini).

```
rtm:3
```

You can use the RAD application logging messages to determine if any tailoring changes you made are the cause of SOAP faults.

Web Services client unable to connect

The most common error occurs when your Web Services client application fails to obtain a response, or you receive this error message:

```
Server Error in <name> Application

The underlying connection was closed: Unable to connect to the remote
server
```

The Web Services client may be directing SOAP requests to the wrong host or to the wrong TCP port number. HP Service Manager generates WSDL files that contain the hostname and TCP port number for the Service Manager server instance receiving the request. The Web Services client application may use the hostname and port number used during application development, but the production hostname and port might be different each time the application runs if they are dynamically allocated.

If the server instance generating the WSDL is different from the hostname or port number receiving the client application requests, the client/server connection will fail. Follow these rules to ensure successful client/server SOAP communication.

- Ensure that your Web Services request is not running against a common port with heavy server traffic. Otherwise, tracking request and response messages will be difficult.
- Type the following at the command line to generate debug logs:

  ```
  sm -httpPort:unique portnumber -sslConnector:0 -debughttp:1 -
  log:../logs/debug.log
  ```

  where

  `-httpPort` identifies a dedicated port for Web Service client connections
  `-log` defines a path to store the logs for this connection

Tip: It is easier to troubleshoot errors if each SOAP client application connects to its own TCP port number.

- Examine the HTTP.LOG file for response messages. You can use this information to determine where a Web Service client connection failure occurs.

Troubleshooting RESTful API

This section lists common issues of REST APIful and describes the methods to troubleshoot.

Debugging

The following three parameters are most frequently used for debugging RESTful API:

- debugrest
- dao_threadsperprocess
- dao_sessiontimeout

It is also useful to use the msglog:1 parameter to have all messages written to the sm.log as well.

The debugrest parameter

Add the debugrest in sm.ini or in the dedicated servlet container line of the sm.cfg file, restart the Service Manager and re-run the RESTful Web service application to invoke debugging parameter. It provides more detailed log trace for diagnostics.

The dao_threadsperprocess parameter

Similar to debugrest, it can be set in sm.ini or in the dedicated servlet container line of sm.cfg file

It means the maximum number of threads allowed concurrently running in the process for RESTful Web Service application. It can be defined in sm.ini and the default value is 10. It is better to start enough server threads to handle requests (suggest to maintaining the buffer of 30%~ 40% spare capacities).

The dao_sessiontimeout parameter

Like debugrest, the dao_sessiontimeout parameter can be set in sm.ini or in the dedicated servlet container line of sm.cfg file.

It indicates the seconds to wait before terminate the RESTful threads. It can be defined in sm.ini and the default value is 15 seconds. Unless the client sends subsequent requests within the timeout, the Service Manager server will recycle the session for re-use and re-allocate it on demand.

If there is big divergence, it is recommended to connect several servlets with different thresholds.

Chapter 6

Syntax for entity references in xml

Character represented	Entity Reference	xml code
>	greater than	>
<	less than	<
“	Quotation marks	"
&	ampersand	&
‘	apostrophy	'

Chapter 7

Definitions, acronyms, and abbreviations

Term	Definition
Consuming	Using a Web Service by calling its methods, supplying the appropriate calling parameters
Publishing	Providing a service over the Web by making public the services operations and objects in a Web Service.
WSDL	Web Services Description Language, which is a standard, structured way of describing SOAP messages and Web Services
REST	Representational State Transfer.

Chapter 8

Web Services resources

You can use the following resources to develop and publish your own Web Services.

The World Wide Web Consortium (W3C) has existed for almost ten years. Its objective is to develop common protocols and to recommend standards that promote Internet interoperability. There are over 400 member organizations who contribute to forming recommendations for standards and best practices among Internet developers. The W3C provides leadership in an array of Web technologies (including XML, HTML, and similar areas of interest) by creating working groups that gather and publish information and recommendations.

You can find the WSDL schema and SOAP schemas published and propagated by IBM and Microsoft at schemas.xmlsoap.org. The W3C has complete descriptions of the schema elements for both SOAP and WSDL. See the W3C Web site for the most recent working draft of SOAP and WSDL recommendations.

There are third party tool kits that simplify creating a Web Service. For example, Apache Axis and Microsoft Visual Studio .NET are development tool kits you can use to create a custom Web Services client directly from the Service Manager Web Services API WSDL.

If you are interested in examples of working Web service WSDL files, programmatic interfaces, tutorials, samples, and a list of available Web services, see the Xmethods Web site. Also see the resources listed below:

- *Service-Oriented Architecture : A Field Guide to Integrating XML and Web Services*, April 2004, Prentice Hall Publishing
- *Web Services: A Technical Introduction*, August 2004, Prentice Hall Publishing
- *Java Web Services*, March 2004, O'Reilly
- Apache Axis
- Microsoft Visual Studio .NET
- schemas.xmlsoap.org
- SOAP schemas
- World Wide Web Consortium
- Gzip Web site
- Apache Wink

- Representational state transfer
- Hypertext Transfer Protocol