

Writing web service  
applications in Perl for  
HP Service Manager 9.X and BP4SM  
9.X

Dr Christopher Vance and Greg Baker

August 2012

# Contents

<b>1</b>	<b>Agenda</b>	<b>1</b>
<b>2</b>	<b>SOAP Introduction</b>	<b>3</b>
2.1	What is SOAP/WSDL? . . . . .	4
2.2	SOAP . . . . .	10
2.3	WSDL . . . . .	16
2.4	How do I access the Service Manager SOAP service? . . . . .	26
2.5	Exercise . . . . .	28
<b>3</b>	<b>SOAP in Perl</b>	<b>29</b>
3.1	What options are there for SOAP/WSDL client-side in Perl? . . . . .	30
3.2	Building a Perl client . . . . .	33
3.3	SOAP::Lite — putting stuff in . . . . .	34
3.4	SOAP::SOM — getting stuff back out . . . . .	41
3.5	SOAP::Data — calling parameters . . . . .	48
3.6	Exercise . . . . .	50
<b>4</b>	<b>HP Service Manager Tickets</b>	<b>51</b>
4.1	Contacts and operators . . . . .	52
4.2	Interaction tickets and their lifecycle . . . . .	55
4.3	Incident tickets and their lifecycle . . . . .	62
4.4	Change tickets and their lifecycle . . . . .	64
4.5	Exercise . . . . .	67
<b>5</b>	<b>Creating Tickets</b>	<b>68</b>
5.1	Common fields . . . . .	69
5.2	Create skeleton . . . . .	71
5.3	What an interaction ticket needs . . . . .	76

---

5.4	What an incident ticket needs . . . . .	77
5.5	What a change ticket needs . . . . .	78
5.6	What ticket should I use? . . . . .	79
5.7	Exercise . . . . .	83
<b>6</b>	<b>Other Tickets and Tools</b>	<b>84</b>
6.1	Service Catalogue . . . . .	85
6.2	Configuration Items . . . . .	86
6.3	Creating a new web service interface . . . . .	87
6.4	Creating a mailer interface . . . . .	90
<b>7</b>	<b>Techniques</b>	<b>98</b>
7.1	Performance . . . . .	99
7.2	Tables . . . . .	102
	<b>Index</b>	<b>106</b>



*1*

**Agenda**

---

# Agenda

*Agenda*

---

- SOAP Introduction
- SOAP in Perl
- HP Service Manager Tickets
- Creating Tickets
- Other Tickets and Tools

---

## Notes.

- Introduction.
- Fire and emergency.
- Locations of restrooms.
- Break location and times.

2

**SOAP Introduction**

## 2.1 What is SOAP/WSDL?

### *SOAP/WSDL*

---

Like many parts of IT, SOAP and WSDL fit into a universe of dependencies.

**RPC** Remote Procedure Call

**XML** Extensible Markup Language

**HTTP** Hypertext Transfer Protocol

**SOAP** Simple Object Access Protocol

**WSDL** Web Services Definition (or Description) Language

---

**Notes.**



Remote Procedure Call is any mechanism used for a client program on one machine to invoke (or call) a procedure on another machine.

To make it work, the ends have to agree

- What procedures are available
- What arguments does each procedure take for input and output
- How are arguments and the choice of procedure expressed
- Where should information be sent to cause invocation

The results are invariably returned to the client using the same formats and methods as the invocation.

---

**Notes.**

A number of RPC mechanisms are used in the real world. Some of the earliest ones were originally motivated by the need to share files.

In 1984 Sun invented Sun RPC, mostly used these days for NFS on Unix and Linux systems.

In the same year CCITT (now called ITU-T) described ASN.1, still used for LDAP, SNMP, cryptographic keys, etc.

In the mid 1980s Apollo Computer (later bought by HP) invented NCS, which which later became DCE, used by Microsoft DCOM and ODBC.

In the late 1980s, IBM invented SMB, now called CIFS and most heavily used by Microsoft systems, and implemented by the free Samba software.

The named systems all use binary encodings to make transfer efficient, at the cost of interoperability. Some of these encodings are at least partially self-describing, while others are not.

The Extensible Markup Language attempts to provide a self-describing textual view of data using nested tags. Typical use is within disk files.

```
<person><name>Baz</name><height unit="metre">2.0
</height><moustache/></person>
```

It comes from the same heritage as HTML, used for web pages.

XML does not require the use of whitespace for indentation or readability, since it is intended primary for machine parsing.

---

### Notes.

XML enables better interoperability at the expense of verbosity.

Tags are expressed within angle brackets. They are case-sensitive, nest strictly, and must be explicitly closed. Content between the open and close tag may include nested tags and character data.

An opening tag may include attributes (see unit within height above).

A closing tag starting with </ must appear somewhere following the opening tag.

A tag ending with />, such as <moustache/> above, is self-closing. It may include attributes, but cannot contain other tags or non-tag data.

Whitespace may separate the / from its tag name, whether in </ or /> form.

```
<?xml version="1.0" encoding="UTF-8"?>
<Envelope>
  <Body>
    <RetrieveIncidentKeysListRequest>
      <model>
        <keys />
        <instance>
          <IncidentID />
        </instance>
      </model>
    </RetrieveIncidentKeysListRequest>
  </Body>
</Envelope>
```

---

**Notes.**

This example is edited down from real XML used with SOAP and HP Service Manager. The original unedited XML includes namespaces and attributes, and omits all the indentation and newlines.

The Hyper Text Transfer Protocol was invented for the world-wide web. A request includes

- a command line, starting with GET, POST, etc.
- a number of header lines, each effectively a name, followed by a colon (:), whitespace, and content to end-of-line
- one empty line
- text or binary content

The response is returned in the same similar format. Apart from the single command line, the file format, including header lines and content separation is derived from the format used for e-mail.

---

**Notes.**

```
POST /SM/7/ws HTTP/1.1
Host: bp4smdemo.hpswededucation.com:13080
Accept: text/xml
Accept: multipart/*
Accept: application/soap
Content-Length: 546
Content-Type: text/xml; charset=utf-8
SOAPAction: "#Retrieve"
```

```
<?xml version="1.0" encoding="UTF-8"?><soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
```

Most of the (XML) content disappears off the end of a single long line.

---

**Notes.**

The client program has a choice between

```
POST http://bp4smdemo.hpswededucation.com:13080/SM/7/ws HTTP/1.1
```

and

```
POST /SM/7/ws HTTP/1.1
Host: bp4smdemo.hpswededucation.com:13080
```

Both have the same effect.

## 2.2 SOAP

## SOAP

---

The Simple Object Access Protocol provides a way to express remote procedure call. It is an XML-based language. Service Manager uses SOAP v1.1.

- The request and its arguments are serialized into an XML document.
- The request document is passed via an HTTP POST action to the server.
- The server deserializes the XML, performs the requested procedure, and serializes the results into XML.
- The response document is passed back to the client as the result of the HTTP POST action.
- The client deserializes the response XML.

---

### Notes.

The HTTP headers required for SOAP with HP Service Manager include

- SOAPAction
- Content-Length

Omitting either of these, or incorrect values of them, will cause HP SM to return a failure or fault, rather than performing the requested service.

The XML part of a real SOAP request is typically a single line. Even with newlines put in to aid legibility, the full content does not fit on a slide, so see the course notes for the full content.

```
POST http://bp4smdemo.hpswededucation.com:13080/SM/7/ws HTTP/1.1
Authorization: Basic ...
Content-Length: 546
Content-Type: text/xml;charset=utf-8
SOAPAction: "RetrieveKeysList"
```

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:soapenc="http:
  <soap:Body>
    <RetrieveIncidentKeysListRequest xmlns="http://bp4smdemo.hpswededucation.com:13080/SM/7
      <model><keys/><instance><IncidentID/></instance></model>
    </RetrieveIncidentKeysListRequest>
  </soap:Body>
</soap:Envelope>
```

---

**Notes.**

Note that the SOAP request body contains a tag ending with Request. This is a convention followed by HP Service Manager, not a requirement of SOAP *per se*.

We show the request over several slides, and several pages in the course notes, to emphasize different parts.

The good news is that most of the material in any SOAP request you deal with is likely to be boilerplate generated by the Perl modules you'll use.

```
...  
  
<?xml version="1.0" encoding="UTF-8"?>  
<soap:Envelope  
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"  
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"  
  soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"  
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">  
  <soap:Body>  
    ...  
  </soap:Body>  
</soap:Envelope>
```

---

### Notes.

The `xmlns` attributes specify XML namespaces which define tags available in a document.

This slide shows the `Envelope` tag and its attributes.



```
...  
  
<?xml version="1.0" encoding="UTF-8"?>  
<soap:Envelope ...>  
  <soap:Body>  
    <RetrieveIncidentKeysListRequest  
      xmlns="http://bp4smdemo.hpsweduction.com:13080/SM/7">  
      <model>  
        <keys />  
        <instance>  
          <IncidentID />  
        </instance>  
      </model>  
    </RetrieveIncidentKeysListRequest>  
  </soap:Body>  
</soap:Envelope>
```

---

**Notes.**

This slide shows the body of the request.

This request returns the keys (IncidentID) for all Incident tickets.

This slide shows that the xmlns attribute can be use in more than one tag (pre-  
ceding ones were on the Envelope tag).

Here is the response to the preceding request.

HTTP/1.1 200 OK

Content-Length: 766

Content-Type: text/xml; charset=utf-8

Set-Cookie: ...

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <RetrieveIncidentKeysListResponse message="Success" query="" returnCode="0" schemaRev="1.0">
      <keys><IncidentID type="String">IM10420</IncidentID></keys>
      <keys><IncidentID type="String">IM10422</IncidentID></keys>
    </RetrieveIncidentKeysListResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

---

### Notes.

Just as the tag within Body ends with Response. Again, this is a HP Service Manager convention not a SOAP requirement.

Please note that the tags for the request and response are almost always going to differ by necessity of holding different internal structures.

```
...
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <RetrieveIncidentKeysListResponse
      message="Success"
      query=""
      returnCode="0"
      schemaRevisionDate="2011-12-04"
      schemaRevisionLevel="1"
      status="SUCCESS"
      xmlns="http://schemas.hp.com/SM/7"
      xmlns:cmn="http://schemas.hp.com/SM/7/Common"
      xmlns:xmime="http://www.w3.org/2005/05/xmlmime"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://schemas.hp.com/SM/7 http://bp4smdemo.hpsweducation.com
      ...
    </RetrieveIncidentKeysListResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

---

**Notes.**

The response includes a bunch of `xmlns` attributes, as well as some additional information.

Just as the request message is mostly boilerplate assembled by the Perl modules you'll be using, the response message is also mostly boilerplate which is understood and deconstructed by the same Perl modules.

## 2.3 WSDL

## WSDL

---

WSDL is the Web Services Definition Language (v1.1) or Web Services Description Language (v2.0), and provides a way to express the remote procedure calls available at a service endpoint, and the arguments required of each call. It is an XML-based language.

HP Service Manager, like many implementations, uses WSDL 1.1, perhaps at least in part because the Business Process Execution Language, BPEL, only supports 1.1.

We won't show a complete WSDL example, but there will be excerpts showing each of the components from the IncidentManagement.wsdl file used by Service Manager.

---

### Notes.

Just as with SOAP, you won't be needing to write a WSDL file, nor even read one very often. Consider it more useful as a reference. The tags used are supposed to be informative, as in programming languages like Java where convention is to use phrases when constructing names for modules, packages, and procedures.

The components in a WSDL 1.1 file include

- types, describing the simple and complex data structures accompanying each request and response
- message, describes a single request or response message
- operation, describes a single remote procedure call, and combines input and output, being one message from client to server, and one message from server to client
- portType, being a collections of operations
- binding, being a specification of the transport protocols of the operations in a portType
- port, being the a service endpoint, typically a URL, used by a binding
- service, being a collection of ports

---

**Notes.**

```
<types>
  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" attributeFormDefault="unqualified"
    <xs:import namespace="http://www.w3.org/2005/05/xmlmime" schemaLocation="http://www.w3.org/2005/05/xmlmime" />
    <xs:import namespace="http://schemas.hp.com/SM/7/Common" schemaLocation="http://bp4sm.com/SM/7/Common" />
    <xs:complexType name="IncidentKeyType">
      <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:attribute name="updatecounter" type="xs:long" use="optional" />
    </xs:complexType>
    ...
  </xs:schema>
</types>
```

---

**Notes.**

A number of complexType tags will normally appear within a schema, even though we've only shown one.

```
<types>
  <xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    attributeFormDefault="unqualified"
    elementFormDefault="qualified"
    targetNamespace="http://schemas.hp.com/SM/7"
    version="2011-12-04 Rev 1"
    xmlns="http://schemas.hp.com/SM/7"
    xmlns:cmn="http://schemas.hp.com/SM/7/Common"
    xmlns:xmime="http://www.w3.org/2005/05/xmlmime">
    <xs:import namespace="http://www.w3.org/2005/05/xmlmime" schemaLocation="http://www.w
    <xs:import namespace="http://schemas.hp.com/SM/7/Common" schemaLocation="http://bp4sm
    <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:attribute name="updatecounter" type="xs:long" use="optional"/>
    </xs:complexType>
  </xs:schema>
</types>
```

---

**Notes.**

The schema includes a number of attributes.

```
<types>
  <xs:schema ...>
    <xs:import
      namespace="http://www.w3.org/2005/05/xmlmime"
      schemaLocation="http://www.w3.org/2005/05/xmlmime"/>
    <xs:import ...>
    <xs:complexType name="IncidentKeyType">
      <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:attribute
        name="updatecounter"
        type="xs:long"
        use="optional"/>
    </xs:complexType>
  </xs:schema>
</types>
```

---

**Notes.**

Imported namespaces provide some of the tags used to describe data structures. Attributes include an indication whether values can be explicitly empty, or whether their use is compulsory.



```
<types>
  <xs:schema ...>
    <xs:import ...>
    <xs:complexType name="IncidentKeyType">
      <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:attribute
        name="updatecounter"
        type="xs:long"
        use="optional"/>
    </xs:complexType>
  </xs:schema>
</types>
```

---

**Notes.**

Only the innermost data types can be predefined, like string. Almost everything else is complexType with internal structure.

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

---

**Notes.**

Here we intuit from the tag names used that RetrieveIncident uses one message type for the Request and another for the Response.

Typically, a message only contains one part.

## *WSDL operation and portType*

---

```
<portType name="IncidentManagement">
  <operation name="RetrieveIncident">
    <documentation/>
    <input message="ns:RetrieveIncidentRequest"/>
    <output message="ns:RetrieveIncidentResponse"/>
  </operation>
  <operation name="RetrieveIncidentKeysList">
    <documentation/>
    <input message="ns:RetrieveIncidentKeysListRequest"/>
    <output message="ns:RetrieveIncidentKeysListResponse"/>
  </operation>
</portType>
```

---

### **Notes.**

Here, an operation has an input and output message.

```
<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>
</binding>
```

---

**Notes.**

The binding includes information on how to transport data for each operation. In this case, "literal" means to transport the XML without compression or conversion to binary format.

*WSDL port and service*

---

```
<service name="IncidentManagement">
  <port binding="ns:IncidentManagement"
        name="IncidentManagement">
    <soap:address
      location="http://bp4smdemo.hpswededucation.com:13080/SM/7/ws"/>
    </port>
  </service>
```

Typically there is only one port entry within the service.

---

**Notes.**

## 2.4 How do I access the Service Manager SOAP service?

### *SOAP on HP Service Manager*

---

The SOAP service is presented via HTTP on a configurable TCP port of your SM machine (default 13080). A request contains some HTTP headers, followed by an XML body.

The response is returned in the same format.

Although web browsers have a mechanism for file upload, they typically have no general mechanism to allow passing content into a web service, so the alternatives are to use an existing program designed to pass XML to a web service, such as a REST client, or to write your own program which does this.

This course introduces programming in Perl using the SOAP::Lite module.

---

#### **Notes.**

The server TCP ports are configured in the `sm.ini` file, as in this excerpt

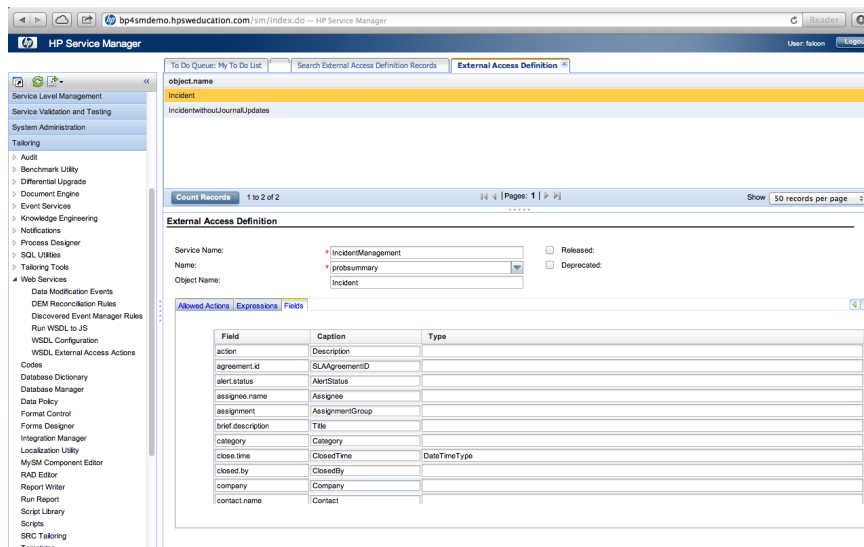
```
system:13080
httpPort:13080
sslConnector:0
httpsPort:13443
```

The web tier is often configured to listen on port 8080, and runs inside a java application server, such as Tomcat. A load balancer is often run in front of this.

The web services interface does not involve the web tier, but communicates directly a ServiceManager application server. The web tier is also configured to talk to the application servers.

Unfortunately the WSDL definitions provided by HP Service Manager do not make it easy to work out all the details of message structure. In particular, the language does not express which values SM will be happy with, and which will cause distress or misbehaviour.

You may find, as we have done, that a certain amount of trial and error is necessary, as is comparison with the results of using SM from a web browser.



**Notes.**

With SM, the WSDL file content is dynamically generated, as a result of the current settings in the extaccess table.

The Service Name is the name presented by web services and forms the basis of where the WSDL file is located.

`http://servername:port/SM/7/Service Name.wsdl`

“Name” is the name of the supporting Service Manager table.

“Object Name” is appended to the allowed actions so that it can be distinguished from other actions on the same Service Name.

Column 1 of the table lists the column names in the table. Column 2 represents their names as presented via web services.

## 2.5 Exercise

### *Exercise*

---

Exercise: Fetch the WSDL definition from HP Service Manager

Within `http://bp4smdemo.hpsweduction.com:13080/SM/7/` you'll find a number of `.wsdl` files, including

- `ChangeManagement.wsdl`
- `ConfigurationManagement.wsdl`
- `FSCManagement.wsdl`
- `IncidentManagement.wsdl`
- `ProblemManagement.wsdl`
- `ServiceDesk.wsdl`

Fetch at least one of these files, and inspect the content.

---

#### Notes.

Your easiest approach is to use a web browser. You'll need to type the full URL into the address field of your browser, since you won't be able to browse by directory search.

Command line alternatives available on most non-Windows systems include one or more of:

- `wget -O` (that's capital letter O)
- `curl -o`
- `ftp -o` (BSD)

If you have extra time, you might also want to compare with what you see using SM in a web browser.

Log in to `http://bp4smdemo.hpsweduction.com/`.

Navigate to `Tailoring > Web Services > WSDL Configuration`.

Ask the trainer for more information on values to try in the form.



3

**SOAP in Perl**

## 3.1 What options are there for SOAP/WSDL client-side in Perl?

### *SOAP modules*

---

Hundreds of Perl modules claim to implement or to use SOAP.

`SOAP::Lite` appears to provide the most popular method for writing SOAP clients and servers. Its dependency on WSDL is relatively 'Lite'. With the level of information provided by SM for WSDL, this appears to be a reasonable match.

`SOAP::WSDL` provides a heavier weight implementation, much more strongly tied to WSDL, and requiring configuration before use. The benefits gained do not appear to justify the costs of using this.

---

#### **Notes.**

In addition, `MIME::Base64` is used for basic password authentication. If you care about protecting this data in transit, you might want to configure your SM SOAP service, or a reverse proxy in front of it, to use SSL.

The Comprehensive Perl Archive Network is the standard mechanism for providing Perl modules not otherwise included with your original Perl installation. Whether the modules you need are included, or need to be added after market, will depend on decisions of your Perl vendor.

If you know how to use CPAN, please let your trainer know, and we can move on to the meat of this course.

ActiveState Perl provides its own package manager, PPM, for installing third party modules.

---

**Notes.**

### 3.1. WHAT OPTIONS ARE THERE FOR SOAP/WSDL CLIENT-SIDE IN PERL?

---

#### *ActiveState PPM*

---

Running `ppm` without specifying an argument should enable you to determine if the modules you need are installed. The ones we refer to explicitly are `MIME::Base64` and `SOAP::Lite`. Depending on the version of ActiveState Perl, you may find these packages already installed.

If you have ActiveState Perl installed, but `SOAP::Lite` is not installed, you should be able to install it by opening a command prompt window, and typing the command

```
ppm install SOAP-Lite
```

Similarly for `MIME::Base64`, packaged as `MIME-Base64`.

---

#### **Notes.**

Note that PPM package names use `-` instead of `::`.

## 3.2 Building a Perl client

*Perl client*

---

We'll go through the steps of building a script which retrieves contacts from HP Service manager. The exercise at the end of this part of the course will be polishing this to make it work.

---

**Notes.**

## 3.3 SOAP::Lite — putting stuff in

*Starting out*

---

```
a.pl
1  #!/usr/bin/perl
2
3  use strict;
4  use warnings;
5
6  use SOAP::Lite;
7
8  my $site = 'http://bp4smdemo.hpswededucation.com:13080/SM/7';
9
10 my $soap = SOAP::Lite->new(
11     proxy => "$site/ws",
12     service => "$site/ConfigurationManagement.wsdl",
13     default_ns => $site);
```

---

### Notes.

The main element is the `new` invocation on lines 10–13, which creates a new client object used for accessing the SOAP service.

Before we can use `use SOAP::Lite`, we have to import it, which we do on line 6.

The `proxy` argument on line 11 specifies the URL for the POST command.

The `service` argument on line 12 specified the URL for WSDL description of the SOAP service.

The `default_ns` argument on line 13 specifies the namespace used for XML tags within the request we will assemble.

Because the values for `proxy`, `service`, and `default_ns` share common content, we have used line 8 to factor it out.

The `strict` and `warnings` pragmata on lines 3–4 make Perl programming a little safer.

The `#!` comment on line 1 is appropriate for Linux, BSD, and most Unix editions.

Perl uses double quotes `"` around strings which require interpolation of `$` and `\` expressions, and single quotes `'` around strings which do not require such interpolation. You could of course use `"` around a strings which doesn't include `$` or `\` with the same result as using `'`.

Use of modules such as `Perl::Critic` can be used to impose coding style restrictions on your code. The sample code provided with this course does not necessarily follow all such recommendations.

```
b.pl
10 my $soap = SOAP::Lite->new(
11     proxy => "$site/ws",
12     service => "$site/ConfigurationManagement.wsdl",
13     default_ns => $site);
14
15 my $som = $soap->call('RetrieveContactKeysList');
16
17 print "Fault\n" if ($som->fault);
18
19 print $som->valueof('//RetrieveContactKeysListResponse/keys') . "\n";
```

---

**Notes.**

Now we add the call on line 15 and a first attempt at showing any error on line 17 or the result on line 19.

Unfortunately we receive a runtime error

Use of uninitialized value in concatenation (.) or string at soap-lite/b line 19.

which suggests that the call failed.

```
c.pl
1  #!/usr/bin/perl
2
3  use strict;
4  use warnings;
5
6  use SOAP::Lite +trace => 'debug';
7
8  my $site = 'http://bp4smdemo.hpswededucation.com:13080/SM/7';
9
10 my $soap = SOAP::Lite->new(
11     proxy => "$site/ws",
12     service => "$site/ConfigurationManagement.wsdl",
13     default_ns => $site);
14
15 my $som = $soap->call('RetrieveContactKeysList');
16
17 print "Fault\n" if ($som->fault);
18
19 print $som->valueof('//RetrieveContactKeysListResponse/keys') . "\n";
```

---

#### Notes.

To help determine what went wrong, we enable tracing within SOAP::Lite, by modifying line 6. The module will show the HTTP request and response, including raw XML.

The result of running the script reveals output

```
HTTP/1.1 401 Unauthorized
Connection: close
Date: Sun, 05 Aug 2012 08:15:35 GMT
Server: Apache-Coyote/1.1
WWW-Authenticate: Basic realm="CASM"
Content-Length: 40
Content-Type: text/html;charset=utf-8
Client-Date: Sun, 05 Aug 2012 09:22:18 GMT
Client-Peer: 74.50.56.155:13080
Client-Response-Num: 1
```

```
<HTML><BODY>Not Authorized</BODY></HTML>
```



```

d.pl
6  use SOAP::Lite +trace => 'debug';
7  use MIME::Base64;
8
9  my $site = 'http://bp4smdemo.hpswededucation.com:13080/SM/7';
10 my $user = 'falcon';
11 my $password = 'orange perfection';
12 my $auth = encode_base64("$user:$password", '');
13
14 my $soap = SOAP::Lite->new(
15     proxy => "$site/ws",
16     service => "$site/ConfigurationManagement.wsdl",
17     default_ns => $site);
18 $soap->transport->http_request->header('Authorization'
19     => "Basic $auth");
20
21 my $som = $soap->call('RetrieveContactKeysList');
22
23 print "Fault\n" if ($som->fault);
24 print $som->valueof('//RetrieveContactKeysListResponse/keys') . "\n";

```

**Notes.**

Service Manager uses basic HTTP authorization, which requires using base64 encoding of the user and password.

Line 7 imports the module which provides base64 encoding.

Line 18 adds the Authorization header to the request, as revealed by the tracing output.

The result of running the script now receives a SOM (SOAP Output Message). Unfortunately it indicates a fault, but that's far better than not executing the SOAP request at all. The output XML has been edited for readability.

```

HTTP/1.1 500 Internal Server Error
Connection: close
Date: Sun, 05 Aug 2012 08:58:11 GMT
Server: Apache-Coyote/1.1
Content-Type: text/xml;charset=utf-8
Client-Date: Sun, 05 Aug 2012 10:04:54 GMT
Client-Peer: 74.50.56.155:13080
Client-Response-Num: 1
Set-Cookie: JSESSIONID=19FDDDD3012A22CB923AB4BED1AD19CA; Path=/SM

```

```

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
      <faultcode>SOAP-ENV:Server</faultcode>
      <faultstring>A CXmlApiException was raised in native code : error 16 : scxmlapi(16) - Invalid
      <faultactor>Server</faultactor>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

**Fault**

Use of uninitialized value in concatenation (.) or string at soap-lite/d line 24.

```
e.pl
20 my $som = $soap->call('RetrieveContactKeysList');
21
22 if ($som->fault) {
23     my $code = $som->faultcode;
24     my $actor = $som->faultactor;
25     my $string = $som->faultstring;
26     my $detail = $som->faultdetail;
27     my $out = "Fault\n";
28     $out .= "\tCode\t\t$code\n" if ($code);
29     $out .= "\tActor\t\t$actor\n" if ($actor);
30     $out .= "\tString\t\t$string\n" if ($string);
31     $out .= "\tDetail\t\t$detail\n" if ($detail);
32     die $out;
33 }
34
35 print $som->valueof('//RetrieveContactKeysListResponse/keys') . "\n";
```

---

**Notes.**

Now that we can recognize a fault as reported by SOAP, we extract the fault information from the SOAP response and print it, lines 22–33.

If there was a fault, we no longer attempt to print useful results, but rather exit the script with error indication on line 32.

This removes the error message about uninitialized value.

The fault we currently see is

Fault

```
Code    SOAP-ENV:Server
Actor   Server
String  A CXmlApiException was raised in native code : \
        error 16 : scxmlapi(16) - Invalid or missing file \
        name in XML request
```

Unfortunately this is insufficient to suggest the direction to proceed.

Two modifications are needed to make progress, discovered by reading the .wsdl file, and by comparing the SOAP request with a working non-Perl implementation.

---

**Notes.**

```
f.pl
14 my $soap = SOAP::Lite->new(
15     proxy => "$site/ws",
16     service => "$site/ConfigurationManagement.wsdl",
17     default_ns => $site);
18 $soap->transport->http_request->header('Authorization'
19     => "Basic $auth");
20 $soap->on_action(sub { '"RetrieveKeysList"'; } );
21
22 my $keys = SOAP::Data->name('keys' => '')->type('');
23 my $instance = SOAP::Data->name('instance' => '')->type('');
24 my $model = SOAP::Data->name('model' =>
25     \SOAP::Data->value($keys, $instance));
26 my $som = $soap->call('RetrieveContactKeysList', $model);
```

**Notes.**

Looking at ConfigurationManagement.wsdl shows

```
<operation name="RetrieveContactKeysList">
  <soap:operation soapAction="RetrieveKeysList" style="document"/>
  ...
</operation>
```

where the soapAction attribute does not include Contact. On line 19, we have modified the SOAPAction header to use this value, rather than the incorrect SOAP::Lite default.

On lines 21–23 we have constructed data which modifies the RetrieveContactKeysList part of the request XML from

```
<RetrieveContactKeysList
  xmlns="http://bp4smdemo.hpswededucation.com:13080/SM/7"
  xsi:nil="true" />
```

to

```
<RetrieveContactKeysList
  xmlns="http://bp4smdemo.hpswededucation.com:13080/SM/7">
  <model>
    <keys />
    <instance />
  </model>
</RetrieveContactKeysList>
```

These modifications change the output to

```
HASH(0x7fe1eaaedb0)
```

meaning that there is now a result, even if it doesn't yet look informative.

The use of SOAP::Data will be explained more later.

## 3.4 SOAP::SOM — getting stuff back out

*SOM*

---

SOM stands for SOAP Object Model.

What remains now is to process the `SOAP::SOM` object received from `$soap->call`.

We use the `valueof` method on `$som` to extract data, using expressions similar to those used with DOM (describing HTML documents for manipulation using Javascript, etc.).

---

### Notes.

The beginning and end of the XML response are

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <RetrieveContactKeysListResponse message="Success" query="" returnCode="0" schemaRevisionDate="2002-01-01" >
      <keys><ContactName type="String">AARON, JIM</ContactName></keys>
      <keys><ContactName type="String">ACRE CORNER, ROB</ContactName></keys>
      ...
      <keys><ContactName type="String">ZAHN, HAROLD</ContactName></keys>
    </RetrieveContactKeysListResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

```
g.pl
30     my $detail = $som->faultdetail;
31     my $out = "Fault\n";
32     $out .= "\tCode\t$code\n" if ($code);
33     $out .= "\tActor\t$actor\n" if ($actor);
34     $out .= "\tString\t$string\n" if ($string);
35     $out .= "\tDetail\t$detail\n" if ($detail);
36     die $out;
37 }
38
39 foreach my $k
40     ($som->valueof('//RetrieveContactKeysListResponse/keys/ContactName')) {
41     print "$k\n";
42 }
```

---

**Notes.**

We changed line 39 to a loop over the appropriate elements of the SOM, and the result begins with

```
AARON, JIM
ACRE CORNER, ROB
ADAMS, IRENE
ADMIN HARVEY
ADMIN, CONFIG
```

You may notice some blank lines in the output. These are due to some `ContactName` values containing multiple lines.

```
h.pl
39  soapdata($som, '//RetrieveContactKeysListResponse/keys', 1);
40
41  sub soapdata {
42      my ($som, $root, $num) = @_;
43      my $ref = $som->valueof($root);
44      my @keys = sort(keys(%$ref));
45      foreach my $k (@keys) {
46          my $count = 0;
47          foreach my $v ($som->valueof("$root/$k")) {
48
49              }
50          }
51      }
52  }
```

---

**Notes.**

Now we've replaced lines 39–41 with a more general routine which knows how to show the internal structure of a SOM node. This may not make much difference to a key listing, but it will help when we retrieve individual records with a larger number of fields.

On line 42, the first parameter gives the SOM to extract data from, the second parameter indicates which part of the SOM to show, while the third parameter indicates whether the elements are believed to be part of an array. In this case numbers are added.

The `soapdata` function is not specific to this script, so it could be separated into a separate file, and included using `use`. We'll do this shortly.

```
h.pl
47     foreach my $v ($som->valueof("$root/$k")) {
48         my $kk = $k;
49         $count += 1;
50         if (! defined $v) {
51             $kk .= "[$count]" if ($num);
52             print "$kk: (undefined)\n";
53         } elsif (ref($v) eq 'HASH') {
54             soapdata($som, "$root/$k", 1);
55         } elsif ($v eq '') {
56             $kk .= "[$count]" if ($num);
57             print "$kk: (empty)\n";
58         } else {
67         }
68     }
```

---

**Notes.**

Line 47 allows the possibility of repetition.

Lines 51–52 handle the case where there is no data.

Line 54 handles the case where there is nested structure.

Lines 56–57 handle the case where there is data, but it's an empty string.



```
h.pl
58         } else {
59             $kk .= "[$count]" if ($num);
60             my @ls = split(/\n/, $v);
61             my $lcount = 0;
62             foreach my $l (@ls) {
63                 $lcount += 1;
64                 $kk .= "($lcount)" if ($#ls > 0);
65                 print "$kk: $l\n";
66             }
67         }
```

---

**Notes.**

Lines 59–66 handle the case where the data is a string, including the possibility of it spanning multiple lines.

A useful step here is to make soapdata a separate module.  
HPSMSOAP.pm

```
1  #!/usr/bin/perl
2
3  use SOAP::Lite;
4
5  package HPSMSOAP;
6  require Exporter;
7  @ISA = qw(Exporter);
8  @EXPORT = qw(soapdata);
9
10 sub soapdata {
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29 }
30
31
32
33
34
35
36
37
38
39 }
40
41 1;
```

---

#### **Notes.**

We put the function into a separate file, called HPSMSOAP.pm, with a small amount of decoration.

The extra lines around the soapdata function make the containing file a module. The filename must have the .pm suffix. It may be easiest to keep this file within the same directory as the scripts which use it.

The 1; line at the end is a quirk of needing file inclusions to return with success.

*Using the module*

---

i.pl To use the new module, we add line 8 to our script and remove the body of the soapdata function from the end of it.

```
6 use SOAP::Lite +trace => 'debug';
7 use MIME::Base64;
8 use HPSMSOAP;
```

---

**Notes.**

## 3.5 SOAP::Data — calling parameters

### *Calling parameters*

---

With the `RetrieveContactKeysList` script we've written so far, we haven't needed to pass any parameters to the script. Once we have the name of a `Contact`, we may wish to write a script which shows more information about a particular contact, say `RetrieveContact`, in which case we need to get the name into the script somehow, and pass it in the SOAP request.

One approach is to have the script read the argument, while another is to pass it on the command line. The second approach is more usual in scripting on Unix and Linux systems because it enables multiple arguments to be used even if some take more than one line.

---

#### **Notes.**

Reading multiple arguments from the terminal or standard input would make it difficult to handle a mixture of multiple arguments and multiple line arguments, while the POSIX shell is happy to allow quoted strings to cover multiple lines.

Now let's look at the `SOAP::Data` lines in the script.

```
21 my $keys = SOAP::Data->name('keys' => '')->type('');
22 my $instance = SOAP::Data->name('instance' => '')->type('');
23 my $model = SOAP::Data->name('model' =>
24     \SOAP::Data->value($keys, $instance));
```

---

**Notes.**

`SOAP::Data->name` takes arguments like (`'thing' => 'one'`) which produces `<thing>one</thing>`. `SOAP::Lite` attempts to deduce the type of the value (e.g., `'one'` looks like a string) and will typically add an attribute to the opening `<thing>` tag to indicate this. This often produces unwanted or even incorrect verbiage, hence the use of `->type('')` to remove the attribute.

If the tag value is to be empty, we can use `SOAP::Data->name('thing' => '')->type('')` which produces `<thing/>`, which is equivalent to `<thing></thing>`.

`SOAP::Data->value` provides a way to specify nesting of tags, taking a number of arguments for the contained content. Specifying this type of value as the right hand part of a `SOAP::Data->name` invocation requires creating a reference, hence the `\`.

## 3.6 Exercise

### *Exercise*

---

- Exercise: Retrieving contacts from HP Service Manager

---

#### Notes.

Ensure that your copy of `RetrieveContactKeysList.pl` works. (On systems other than Windows, you may be able to omit the `.pl` extension.)

Copy `RetrieveContactKeysList.pl` to `RetrieveContact.pl` (Omit `.pl` if both desired and possible).

Edit in the necessary changes

- Remove `KeysList` wherever it occurs.
- The first command line argument is the first element of `@ARGV`, namely `$ARGV[0]`.
- The `<instance/>` field in the request needs to become `<instance><ContactName>Whatever<`
- The part of the response to print is within `//RetrieveContactResponse/model/instance`.
- When you think it's ready, try with a contact who exists (say, `XINTIAN`), and one who doesn't (say, `FRED`).

If your argument contains spaces or special characters, you'll need to quote it. On Windows, you'll need to use double quote `"`. On POSIX you can choose between `'` and `"`.

You should find `RetrieveContact.pl` works identically whether you put the `<ContactName>` inside `<instance>` or inside `<keys>`.

If you try to put a reference to the same `SOAP::Data` into both, you'll have a failure, but you could make two copies with the same string value, and it'll work.

4

## HP Service Manager Tickets

## 4.1 Contacts and operators

### *Contacts and operators*

---

A ticketing system such as HP Service Manager involves several different kinds of people.

**operator** a person who operates the SM web interface to record, modify, or otherwise act upon tickets, usually on behalf of somebody else; this may include service technicians and other professionals

**contact** a person who communicates with an operator, by telephone or email, to address an issue on behalf of themselves or someone else

**self-service** a person (other than an operator) who uses SM, perhaps with restricted access, to record tickets on their own behalf

In addition computer programs may create tickets reflecting issues which need to be addressed, such as outages, or modify tickets when other situations occur, such as timeouts.

---

#### **Notes.**

Other people involved with SM include supervisors and those whose work is less directly driven by SM tickets than level 1 operators.

A person may be both an operator and a contact.



## *HP Service Manager ticket types*

---

**interaction** When a client communicates with an operator, the operator normally creates an interaction ticket to record this event.

**incident** When an interaction is complex enough not to be fully dealt with during the call, the interaction is escalated, and an incident ticket is created from the interaction.

**change** Sometimes the issue causing communication via SM is that a software or hardware requires (re)installation or upgrading. A change ticket is created to track this. This ticket type includes fields for build, testing, and deployment.

---

### **Notes.**

**interaction** An interaction ticket is always given a category, namely incident, or a request for administration, information, or change.

Some interaction tickets are closed almost immediately after opening, during the initial conversation, as the operator satisfies the contact.

An ongoing issue is typically escalated to an incident, which is a separate kind of ticket related to the interaction.

**incident** The incident is an issue which remains current until explicit steps are taken to close or abandon it. The incident may have related interactions, incidents, or other ticket types..

**change** A change ticket is not identical to the “request for change” category for interaction tickets. An interaction in that category will escalate to a change ticket.

*Interaction tickets*

---

When a contact calls an operator, an interaction ticket is created. The interaction ticket must be categorized as one of **incident** something has gone wrong and needs to be fixed **request for** one of

**administration** grant access or password reset

**change** emergency, normal, or standard

**information** complaint, how to, status, etc.

---

**Notes.**

Some amount of diagnosis and classification can be undertaken by the operator during the call, while further processing by operators or analysts may be necessary to progress the ticket further.

## 4.2 Interaction tickets and their lifecycle

### *Interaction tickets*

---

**create** When a client talks with an operator, the operator creates an interaction ticket to record the issue.

**escalate** If the issue cannot be fully dealt with during the call, and is not a request with its own category, the operator will escalate the issue, which creates a related incident ticket.

**close** When the client is fully satisfied the issue has been handled, the operator will close the issue.

---

#### **Notes.**

If the interaction is marked for email callback, closing other ticket types may close related interaction tickets, as the email notification is automated.

If the interaction is marked for telephone callback, the operator is expected to call the contact back before closing the interaction.

## 4.2. INTERACTION TICKETS AND THEIR LIFECYCLE

---

### *Interaction: Incident*

---

The screenshot displays the HP Service Manager web interface for an interaction with ID SD10797. The interface is divided into two main sections: 'Interaction Details' on the left and a classification section on the right. The 'Interaction Details' section includes fields for Interaction ID, Phase, Entitlement, Status (Registered), Owner (falcon), Primary Contact (FULTON, LOUIS), Full Name (Louis Fulton), Contact Number, Email (fulton.louis@advantage.com), Location (Africa), Alternate Contact Number, Medium of Request, Call Back Method (E-mail), Service Recipient (FULTON, LOUIS), Full Name (Louis Fulton), Assignment Group (Hardware), Assignee Name (Deidre Kirk), SRF Manager (BP SRF Manager), Expected Resolution Date and Time, Service (adv-aff-desk-101), and Affected CI (adv-aff-desk-101). The classification section includes Category (Incident), Sub Category (hardware), Classification (hardware failure), Impact (4 - User), Urgency (4 - Low), Priority (4 - Low), Approval Status, SLA Target Date, Knowledge Source, and an Autoclose Time field with an 'Exclude AutoClose' checkbox. Red stars are visible next to several fields, indicating they are compulsory. The browser address bar shows 'hp4smdemo.hpsweducation.com/sm/index.do' and the user is logged in as 'falcon'.

---

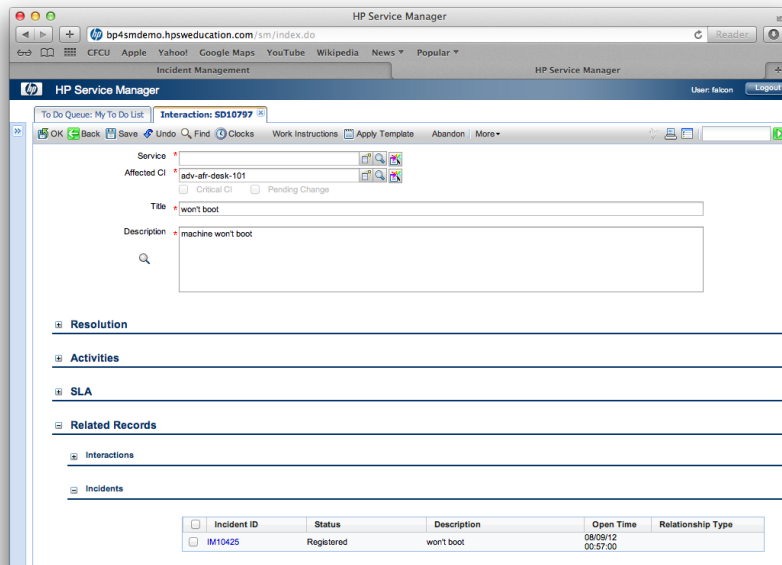
### **Notes.**

This screenshot was taken after an incident-category interaction was created and escalated. Both the interaction and the related incident were saved.

Note the red stars indicating compulsory fields.

*Scroll down*

---



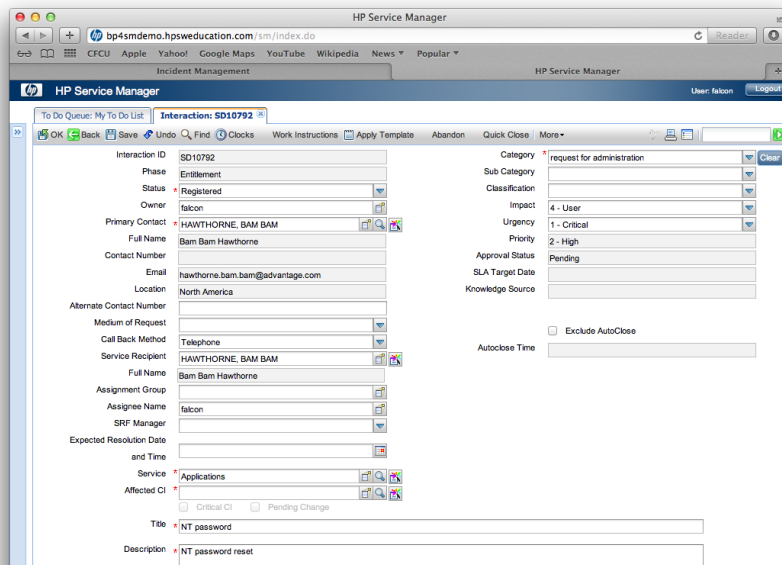
---

**Notes.**

Scrolling down the previous web page shows a link to the related incident.

## 4.2. INTERACTION TICKETS AND THEIR LIFECYCLE

### *Interaction: Request for Administration*



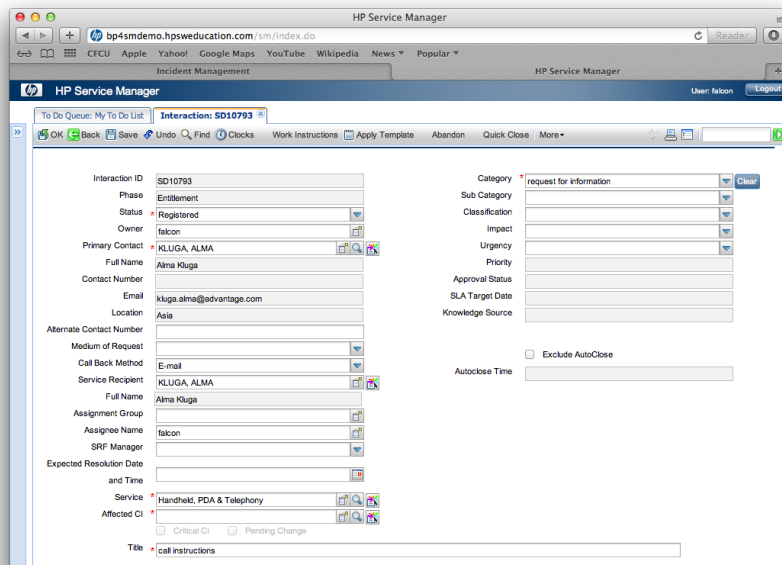
The screenshot displays the HP Service Manager web interface for interaction SD10792. The interface is organized into several sections:

- Header:** Shows the user 'falcon' and a 'Logout' button.
- Navigation:** Includes 'To Do Queue: My To Do List' and 'Interaction: SD10792'.
- Left Panel (Form Fields):**
  - Interaction ID: SD10792
  - Phase: Enrollment
  - Status: Registered
  - Owner: falcon
  - Primary Contact: HAWTHORNE, BAM BAM
  - Full Name: Bam Bam Hawthorne
  - Contact Number: [Empty]
  - Email: hawthorne.bam.bam@advantage.com
  - Location: North America
  - Alternate Contact Number: [Empty]
  - Medium of Request: [Empty]
  - Call Back Method: Telephone
  - Service Recipient: HAWTHORNE, BAM BAM
  - Full Name: Bam Bam Hawthorne
  - Assignment Group: [Empty]
  - Assignee Name: falcon
  - SRF Manager: [Empty]
  - Expected Resolution Date and Time: [Empty]
  - Service: Applications
  - Affected CI: [Empty]
  - Title: NT password
  - Description: NT password reset
- Right Panel (Metadata):**
  - Category: request for administration
  - Sub Category: [Empty]
  - Classification: [Empty]
  - Impact: 4 - User
  - Urgency: 1 - Critical
  - Priority: 2 - High
  - Approval Status: Pending
  - SLA Target Date: [Empty]
  - Knowledge Source: [Empty]
  - Autoclose Time: [Empty]
  - Exclude AutoClose:

#### **Notes.**

This screenshot was taken after a request for administration interaction was created.

*Interaction: Request for Information*



**Notes.**

This screenshot was taken after a request for information interaction was created.

### *Interaction: Request for Change*

---

The screenshot displays the HP Service Manager interface for an interaction with ID SD10799. The interface is divided into two main sections: Interaction Details and a right-hand configuration panel.

**Interaction Details:**

- Interaction ID: SD10799
- Phase: Entitlement
- Status: Registered
- Owner: falcon
- Primary Contact: NGUI, REX
- Full Name: Rex Ngui
- Contact Number: ngui.rex@advantage.com
- Email: ngui.rex@advantage.com
- Location: Europe
- Alternate Contact Number: (empty)
- Medium of Request: (empty)
- Call Back Method: E-mail
- Service Recipient: NGUI, REX
- Full Name: Rex Ngui
- Assignment Group: Intranet / Internet Support (North America)
- Assignee Name: BP\_Change Owner
- SRF Manager: BP SRF Manager
- Expected Resolution Date and Time: (empty)
- Service: VPN Client
- Affected CI: VPN Client

**Right-hand Configuration Panel:**

- Category: request for change
- Sub Category: normal
- Classification: Minor
- Impact: 4 - User
- Urgency: 4 - Low
- Priority: 4 - Low
- Approval Status: (empty)
- SLA Target Date: (empty)
- Knowledge Source: (empty)
- Exclude AutoClose:
- Autoclose Time: (empty)

At the bottom of the form, there are radio buttons for "Critical CI" and "Pending Change", with "Pending Change" selected.

---

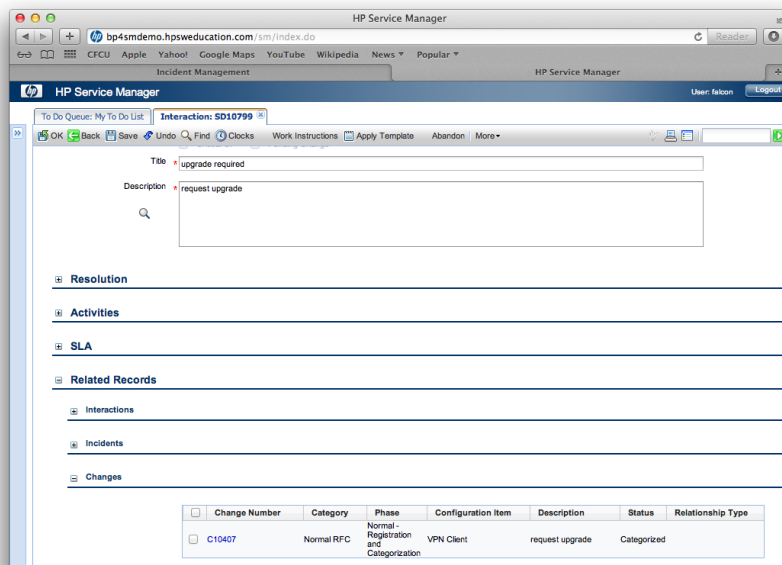
### **Notes.**

This screenshot was taken after a request for change interaction was created and escalated. Both the interaction and the related change were saved.



*Scroll down*

---



---

**Notes.**

Scrolling down the previous web page shows a link to the related change.

## 4.3 Incident tickets and their lifecycle

### *Incident tickets*

---

**categorize and prioritize** The operator will attempt to describe the incident, and determine urgency and impact.

**analyse and diagnose** The operator attempts to find a cause and/or documented workaround.

**relate** The operator will attempt to find related incidents, and mark them as such in Service Manager.

**reassign** If the incident is beyond the operator's skill, the ticket may be reassigned to more experienced staff.

**raise a change ticket** If a software or hardware change is required, the operator may raise a change ticket.

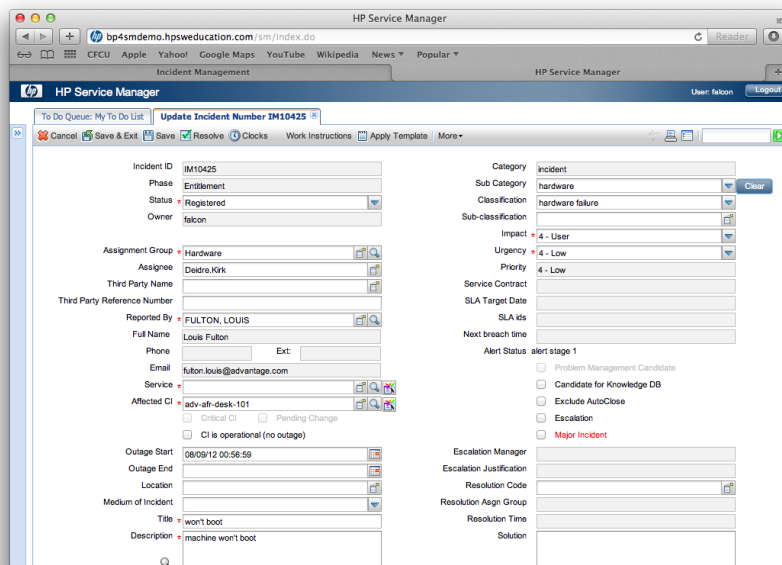
**resolve** If the issue has been resolved to the operator's understanding, it may be marked as resolved.

**close** When the contact is also satisfied with the resolution of the incident, it may be closed.

**abandon** If the contact loses interest in the incident, and nobody else believes it to be an issue, the incident may be marked as abandoned.

---

Notes.



**Notes.**

This ticket was created by escalating an interaction ticket.

It is possible to create an incident ticket directly, rather than from escalation of an interaction.

## 4.4 Change tickets and their lifecycle

### *Change tickets*

---

**log** An operator creates the ticket. Fields may be copied from a request for change interaction.

**assess and evaluate** Assess what needs to be done, and what it will take.

**analyse risk and impact** Analyse what services will be affected by the change, and the relative risk of doing or not doing the change.

**approve** Includes assigning resources to do the work.

**build and test** Assigned resources build and test as required.

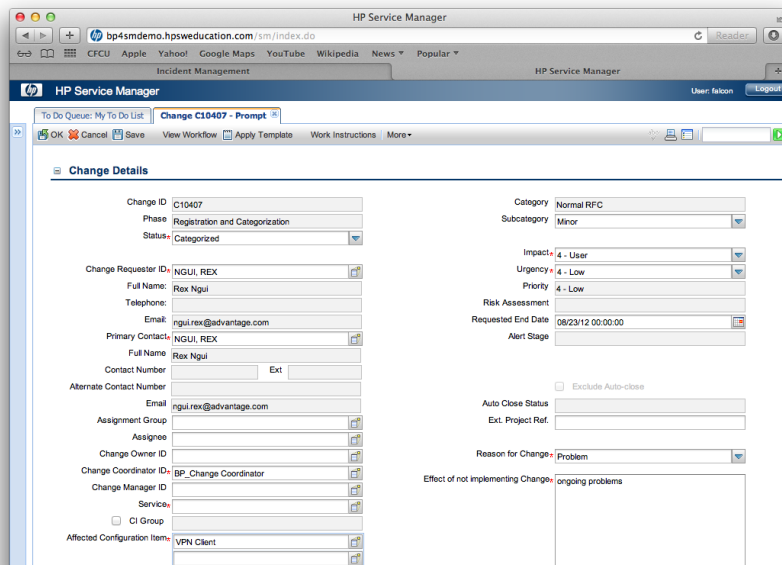
**schedule** Schedule deployment, including notifications.

**implement** Deploy the hardware and/or software.

**review** Post-Implementation Review.

---

**Notes.**



**Notes.**

This ticket was created by escalating an interaction ticket.

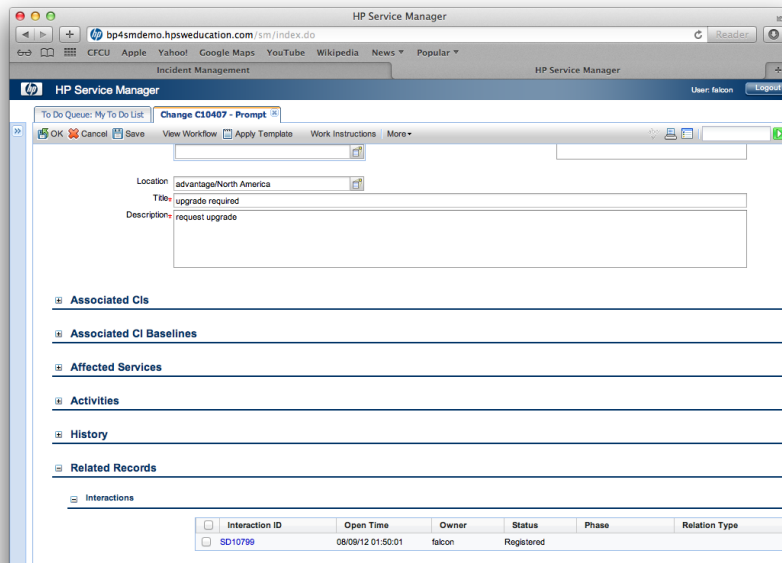
It is possible to create an change ticket directly, rather than from escalation of an interaction.

## 4.4. CHANGE TICKETS AND THEIR LIFECYCLE

---

*Scroll down*

---



---

### Notes.

Scrolling down shows the related interaction.

## 4.5 Exercise

### *Exercise*

---

Exercise: Using the HP Service Manager web interface to see tickets

---

#### **Notes.**

Log in to <http://bp4smdemo.hpsweduction.com/bp4sm93/>.

Visit each of

- Service Desk > Interaction Queue
- Incident Management > Incident Queue
- Change Management > Changes > Change Queue

5

## Creating Tickets



## 5.1 Common fields

### *Creation*

---

Creating tickets requires modifying existing information or adding new information.

As this is more destructive than merely fetching existing information, your script should check additional sources of information about things which might not be quite right.

We suggest including

```
soapfault($som);  
soapstatus($som);  
soapdata($som, '//XResponse/messages', 1);
```

near the end of your script, just before you show the result.

You will need to replace X with the appropriate request type.

---

#### **Notes.**

The `soapfault` and `soapstatus` routines are included in the provided `HPSMSOAP.pm` package.

If there was a SOAP error preventing execution of your request, `soapfault` will indicate what the fault was, and stop the script. Errors tend to be because of missing or erroneous parts in the request XML.

If there was no SOAP error, in that there was a response received from the server, `soapstatus` will indicate a summary of the status of your request. A status other than success can still result in ticket creation, in which case some fields will have default or empty values.

The `soapdata` invocation with `/messages` will show any messages sent by the server to indicate why status might not be an unambiguous Success. For ticket creation, these messages should indicate what fields are missing or fail validation.

Most tickets require the following fields to be given valid values

**Contact** Who wants this done, and/or to be informed about it

**Title** One line summary of the ticket

**Description** Multi-line description of the problem, issue, etc.

**AffectedService** What software service is at issue

**AffectedCI** What hardware is at issue

**Impact** Number 1–4 expands to one of 1 Enterprise, 2 Site/Dept, 3 Multiple Users, 4 User

**Urgency** Number 1–4 expands to one of 1 Critical, 2 High, 3 Average, 4 Low

---

**Notes.**

Our sample program checks for certain fields to ensure they are specified.

Only one of **AffectedService** and **AffectedCI** is required.

The **Contact**, **AffectedService** or **AffectedCI**, **Impact**, **Urgency**, and several other fields must match values already known to Service Manager.

For the Change ticket, you need to use **InitiatedBy** and **bp.primary.contact** instead of **Contact**.

## 5.2 Create skeleton

*Create skeleton 1*

---

```
m.pl
1  #!/usr/bin/perl
2
3  use strict;
4  use warnings;
5
6  BEGIN {
7      if ($0 =~ m|^(.*)/[^\s/]+$/) {
8          push(@INC, $1);
9      }
10 }
11
12 use SOAP::Lite +trace => 'debug';
13 use MIME::Base64;
14 use HPSMSOAP;
15
```

---

**Notes.**

Line 1 should work for any Unix-like OS including Linux and BSD. For Windows, the extension `.pl` should be enough.

Lines 6–10 set the include path for Perl modules to include the directory containing the script you’re running so that `HPSMSOAP.pm` can be found if it is in the same directory. If you install `HPSMSOAP.pm` in one of the directories already on your Perl include path, these lines can be omitted.

Line 12 enables tracing of SOAP messages. Once you’re satisfied your scripts work, you could remove the `+trace => 'debug'` if you like.

```
m.pl
15
16 my $site = 'http://bp4smdemo.hpsweduction.com:13080/SM/7';
17 my $user = 'falcon';
18 my $password = 'orange perfection';
19 my $auth = encode_base64("$user:$password", '');
20
21 my %argument = ();
22 foreach my $a (@ARGV) {
23     if ($a =~ s/^(\[^\=]+\)=//) {
24         $argument{$1} = $a;
25     } else {
26         die "invalid argument [$a]\n";
27     }
28 }
29 my @required = qw(Contact Title Description Impact Urgency);
```

---

**Notes.**

Line 16 needs to be corrected to refer to your actual Service Manager URL.

Lines 17–18 need to be adjusted for the user you want your scripts to raise tickets as.

Lines 21–28 check command line arguments are of the form Name=Value, and puts them into a the dictionary %argument. The value needs to be quoted or escaped if it includes anything which your shell might misinterpret, such as space, newline, or special characters.

On Windows, you may have to (double) quote the entire command line argument including the Name= part.

```
m.pl
29 my @required = qw(Contact Title Description Impact Urgency);
30 my @missing = ();
31 my $misscount = 0;
32 foreach my $r (@required) {
33     if (! exists $argument{$r}) {
34         $misscount += 1;
35         push(@missing, $r);
36     }
37 }
38 die "$misscount required args missing:\n\t" . join(' ', sort(@missing)) . "\n"
39     if ($misscount);
40
```

---

**Notes.**

Lines 29–39 check whether the required arguments (as listed on line 29) are present in %argument, and prints and error message if any are missing.

You probably want to add fields to 29 as appropriate for your request type as you discover them.

```
m.pl
40
41 my @xargs = ();
42 foreach my $k (keys %argument) {
43     push(@xargs, SOAP::Data->name($k => $argument{$k}));
44 }
45 my $x = SOAP::Data->value(@xargs);
46
47 my $soap = SOAP::Lite->new(
48     proxy => "$site/ws",
49     service => "$site/XX.wsdl",
50     default_ns => $site);
51 $soap->transport->http_request->header('Authorization' => "Basic $auth");
52 $soap->on_action(sub { '"Create"'; } );
53
```

---

#### Notes.

Lines 41–45 encode the command line arguments into SOAP data for use on line 55 below.

Lines 47–52 create the `SOAP::Lite` object to be used for your request.

You will have to modify the `XX` on line 49 to refer to the correct `.wsdl` file for your request type.

```
m.pl
53
54 my $keys = SOAP::Data->name('keys' => '')->type('');
55 my $instance = SOAP::Data->name('instance' => \$x)->type('');
56 my $model = SOAP::Data->name('model' => \SOAP::Data->value($keys, $instance));
57 my $som = $soap->call('CreateX', $model);
58
59 soapfault($som);
60 soapstatus($som);
61 soapdata($som, '//CreateXResponse/messages', 1);
62 soapdata($som, '//CreateXResponse/model', 0);
```

---

**Notes.**

Lines 54–57 collect the arguments required for the call, and invokes the SOAP service.

Line 55 uses the SOAP data encoded on lines 41–45 above.

Line 59 indicates whether there was a fault in the SOAP syntax, or any failure to contact the SOAP server, and terminates the script if there was no useful SOAP response.

Line 60 shows the SOAP status of the request.

Line 61 shows any warnings or informational messages associated with the SOAP status. This may include notification of missing or invalid fields not caught above in lines 29–39.

Line 62 shows the return data from your request.

You'll need to modify X for your request type on lines 57 and 61–62.

## 5.3 What an interaction ticket needs

### *Interaction ticket*

---

**Category** must be one of

- incident
- request for change
- request for administration
- request for information

**Area** Acceptable values (and whether required) depend on Category

**Subarea** Acceptable values (and whether required) depend on Category

---

**Notes.**

Remember that incident and request for change categories both result in separate ticket types upon escalation, and that the resulting incident and change tickets can actually be created separately without requiring an interaction ticket to start from. Your work processes will probably specify whether to create the interaction ticket or not.



## 5.4 What an incident ticket needs

### *Incident ticket*

---

In addition to the common fields mentioned previously, an incident ticket requires

**AssignmentGroup** Who will handle the incident.

---

#### **Notes.**

Additional fields are available for categorizing and classifying the incident further, but a ticket can be created without them, or they could be added later with an Update request if you wish.

## 5.5 What a change ticket needs

### *Change ticket*

---

In addition to the common fields mentioned previously, a change ticket requires

**Category** Permitted values depend on whether BP4SM is in use, and/or what customization has been done.

**Reason** Why is the change needed (menu)?

**bp.effect.not.implementation** What is the effect if the change is not implemented?

**InitiatedBy** Who started this change?

**bp.primary.contact** Who is responsible to see this change through?

**AssignmentGroup** Who will handle the incident?

**RequestedEndDate** When should the change be done by?

**bp.status** Similar to Status.

---

#### **Notes.**

The possible values of Category are defined in the category table. The area and subarea tables define permissible subcategories and areas.

For users of HP Service Manager without BP4SM, the standard categories of change include Hardware, Software, Maintenance and several others. Most sites customize these.

For users of BP4SM, Category must be one of Emergency RFC, Normal RFC and Standard RFC unless further site customisation has been done.

## 5.6 What ticket should I use?

*What ticket should I use?*

---

As described previously, each ticket type has a distinct usage.  
We have discussed

- Change
- Incident
- Interaction

We have not discussed

- Configuration Management
- Knowledge Management
- Problem Management
- Release Management
- Request Management
- Service Catalog(ue)

---

**Notes.**

## 5.6. WHAT TICKET SHOULD I USE?

---

### *Change ticket*

---

If you're recording a need for something to be changed, hardware or software, a Change ticket may be appropriate. As these require specific justification, they are probably best raised using the browser interface. Some changes may be routine enough to justify scripting.

---

#### **Notes.**

If you're recording an issue, other than a change, which requires future or ongoing attention, and which is well understood, an Incident ticket is reasonable. These may be routine enough to script.

For scripting, we may usually expect the ticket raised to be an Incident which can be auto-recognized by monitoring software. In this case, it may even be possible for the monitoring software to close some of the tickets it has raised.

---

**Notes.**

If you're recording an interaction between a contact and an operator, which is complete during the conversation, or which is a request for administration or information, an Interaction ticket is reasonable. Given the presence of an operator in the conversation, the browser interface is often appropriate. Scripting these may be useful if you're attempting to tie multiple Interactions into a single Incident. We have not discussed how to script such relationships.

---

**Notes.**

## 5.7 Exercise

### *Exercise*

---

Exercise: Creating an incident ticket

---

#### **Notes.**

Create an incident ticket using the browser interface.

Now attempt to create a script to do the same, referring back to your browser ticket as needed to fill in values.

You may find a Retrieve script useful to determine the field names used by the SOAP interface.

# 6

## Other Tickets and Tools



## 6.1 Service Catalogue

### *Service Catalogue*

---

This provides access to information and things configured to be available for the user to order or request. Depending on customization, this may include things like telephone line provisioning, laptop purchase, or allocation of some resources.

Knowledge Management is included, as knowledge documents grow over time.

---

#### **Notes.**

Actually, HP Service Manager spells it *Service Catalog*.

## 6.2 Configuration Items

### *Configuration Items*

---

Many different types of Configuration Items are available to classify or group people, physical items, and abstract things in ways which help document relationships relevant to the ticket types used within HP Service Manager, and to inventory individuals within each classification.

- Company, Computer, Contact, Department, Device, DisplayDevice, Furnishing, HandHeldDevice, InstalledSoftware, Location, MainFrame, Model, NetworkDevice, OfficeElectronic, SoftwareLicense, StorageDevice, TelecommunicationDevice, Vendor

Note that many of these are specializations of others listed.

---

#### **Notes.**

Each of these types can be fetched using the `RetrieveKeysList` methods you've already used, and then fetched individually using the `Retrieve` methods you've also used. Your `Create` experience is also useful, but you'll need to be aware of the way things are grouped and classified to make it work for you.

## 6.3 Creating a new web service interface

### *Creating a new web service interface*

---

Up to now, this course has been about writing SOAP client programs.

We'll briefly discuss now providing SOAP service.

1. First preference is to use the ITIL-based features already provided by Service Manager Web Services.
2. Second preference is to modify an existing extaccess record to expose access to additional fields of database records already in use for SM Web Services.
3. Third preference is do something new.

---

#### **Notes.**

You might want to consult the HP Service Manager Web Services Guide (SM9.30\_Web\_Services.p from <http://support.openview.hp.com/selfsolve/manuals/>

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

---

**Notes.**

These processes are available out-of-the-box.

Tailoring > Web Services > WSDL Configuration shows externally exposed tables and fields.

Tailoring > Database Dictionary shows tables and fields. Knowing what is connected to what can take some time to learn, and some ferreting around to discover.

The difference between them may show that the data you're after is present in the SM database, but not yet accessible.

You may modify or add *extaccess* record information to expose additional fields or tables.

---

**Notes.**

The *extaccess* tool uses the same file values as the Document Engine.

Browsing WSDL Configuration will help you determine which tables provide information for each *.wsdl*. From there, browsing Database Dictionary will help you determine what fields are used, and perhaps what the table is actually used for.

If you want Service Manager to use an externally provided service, you will need to run *WSDL2JS*.

## 6.4 Creating a mailer interface

*eventout*

---

- Notifications send messages to the eventout table
- These can be **logs**, **email**, **page** or many others.
- Other programs retrieve and delete from this table

---

### Notes.

A quick way of identifying all possible “out events” available in your system is to create a new notification and look at the “Notify Method” pull down.

Typically it will contain:

- connect
- email
- fax
- log
- mail
- msg
- noticenter
- noticenterappr
- page
- print
- tso

Many of these are obsolete, or almost never used – for example, far fewer sites are using “mail” (HP ServiceManager internal email) for notifications compared to “email”. tso is even rarer as it is many, many years since HP ServiceManager ran on a mainframe.

An out-of-the-box BP4SM or ServiceManager server will only have notifications that trigger email, msg and mail.

- Notifications send messages to the eventout table
- These can be **logs**, **email**, **page** or many others.
- Other programs retrieve and delete from this table.
- Usually done by Connect-IT

---

**Notes.**

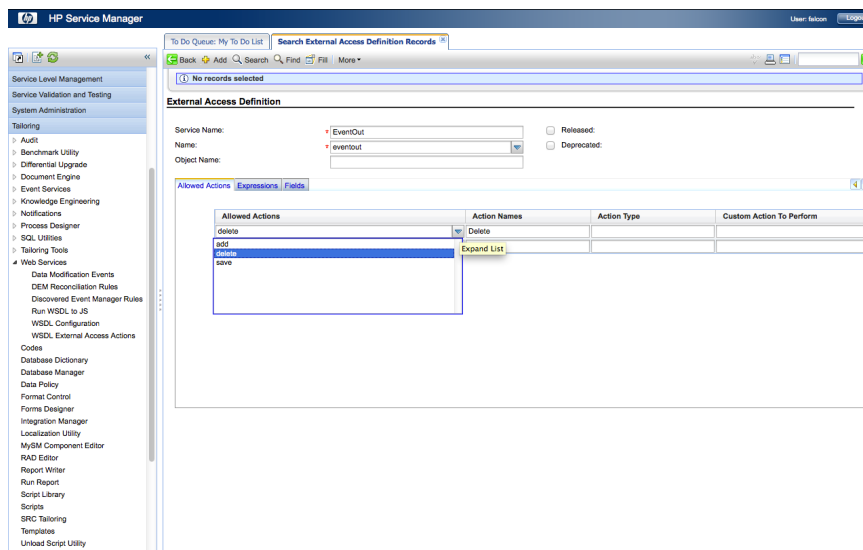
msg notifications appear at the top of the screen in the user's session if they are logged in, and not at all if they are not. ServiceManager handles mail internally, and version 9.3 introduced an SMTP sender. But other than these, ServiceManager has no built-in support for processing out events.

Connect-IT (also known as CIT) is the tool used by most customers for this job. Its strength is in handling web services; it can send emails and write log messages. Complicated processing is difficult since any programming has to be done in BASIC.

Connect-IT can connect on proprietary, unpublished APIs which give it access to all tables regardless of whether they are exposed via WSDL.

## 6.4. CREATING A MAILER INTERFACE

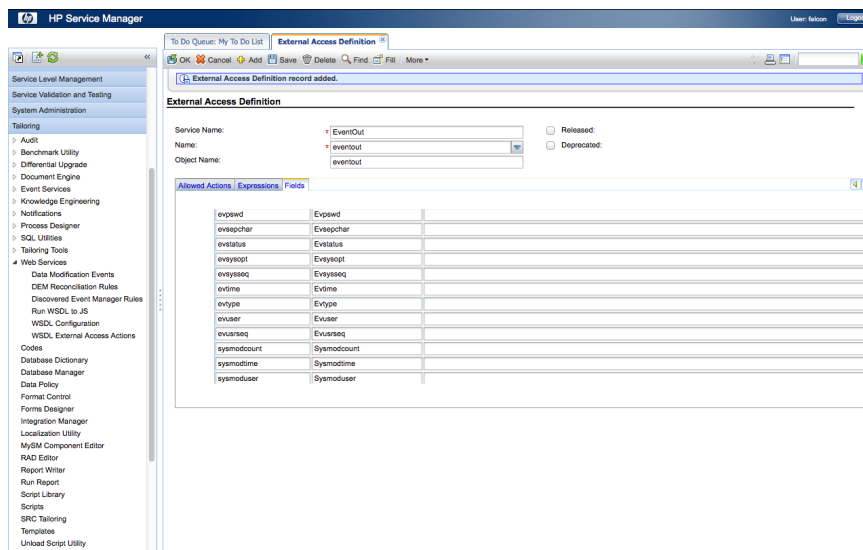
*extaccess actions for eventout*



### Notes.

There is no WSDL definition for the eventout table, so one needs to be created. There is no need to create an add or save action as event out records are only viewed and deleted.



**Notes.**

There is no established standard or convention for eventout web services, nor is there likely ever to be one. There is no reason to hide any fields from view either, so in this example we have mapped every field.

More importantly, it is hard to imagine any upgrade which would affect this extaccess record, so even sites with concerns about stepping away from an out-of-the-box solution need not be too worried.

Notice that the automatically-created sysmodcount, sysmodtime and sysmoduser can also be made available via web services.

Of course, for the eventout table, every record will have a sysmodcount of 1 since after creation they only ever get deleted!

```

- <definitions targetNamespace="http://schemas.hp.com/SM/7" xsi:schemaLocation="http://schemas.xmlsoap.org/wsdl/ http://schemas.xmlsoap.org/wsdl/">
- <types>
- <xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified" targetNamespace="http://schemas.hp.com/SM/7" version="2012-08-22 Rev 0">
- <xs:import namespace="http://www.w3.org/2005/05/xmlmime" schemaLocation="http://www.w3.org/2005/05/xmlmime/">
- <xs:import namespace="http://schemas.hp.com/SM/7/Common" schemaLocation="http://hp4smdemo.hpsweducation.com:13080/SM/7/Common.xsd"/>
- <xs:complexType name="eventoutKeyType">
- <xs:sequence>
- <xs:element minOccurs="0" name="Evsysseq" nillable="true" type="cmn:StringType"/>
- </xs:sequence>
- <xs:attribute name="query" type="xs:string" use="optional"/>
- <xs:attribute name="updatecounter" type="xs:long" use="optional"/>
- </xs:complexType>
- <xs:complexType name="eventoutInstanceType">
- <xs:sequence>
- <xs:element minOccurs="0" name="Evtype" nillable="true" type="cmn:StringType"/>
- <xs:element minOccurs="0" name="Evtime" nillable="true" type="cmn:DateTimeType"/>
- <xs:element minOccurs="0" name="Evsysseq" nillable="true" type="cmn:StringType"/>
- <xs:element minOccurs="0" name="Evsusreq" nillable="true" type="cmn:StringType"/>
- <xs:element minOccurs="0" name="Evsysopt" nillable="true" type="cmn:StringType"/>
- <xs:element minOccurs="0" name="Evuser" nillable="true" type="cmn:StringType"/>
- <xs:element minOccurs="0" name="Evpwd" nillable="true" type="cmn:StringType"/>
- <xs:element minOccurs="0" name="Evsqchar" nillable="true" type="cmn:StringType"/>
- <xs:element minOccurs="0" name="Evfields" nillable="true" type="cmn:StringType"/>
- <xs:element minOccurs="0" name="Evstatus" nillable="true" type="cmn:StringType"/>
- <xs:element minOccurs="0" name="Evexpire" nillable="true" type="cmn:DateTimeType"/>
- <xs:element minOccurs="0" name="Evcount" nillable="true" type="cmn:DecimalType"/>
- <xs:element minOccurs="0" name="Evnumber" nillable="true" type="cmn:StringType"/>
- <xs:element minOccurs="0" name="Evid" nillable="true" type="cmn:StringType"/>
- <xs:element minOccurs="0" name="Sysmodcount" nillable="true" type="cmn:DecimalType"/>
- <xs:element minOccurs="0" name="Sysmoduser" nillable="true" type="cmn:StringType"/>
- <xs:element minOccurs="0" name="Sysmodtime" nillable="true" type="cmn:DateTimeType"/>
- <xs:element minOccurs="0" name="attachments" nillable="true" type="cmn:AttachmentsType"/>
- </xs:sequence>
- <xs:attribute name="query" type="xs:string" use="optional"/>
- <xs:attribute name="uniquequery" type="xs:string" use="optional"/>
- <xs:attribute name="recordid" type="xs:string" use="optional"/>
- <xs:attribute name="updatecounter" type="xs:long" use="optional"/>
- </xs:complexType>
- <xs:complexType name="eventoutModelType">
- <xs:sequence>

```

**Notes.**

The resulting WSDL is available as soon as the definition is saved.

And of course, with Perl, there is no compilation of WSDL to code required, so you are good to write your application immediately!

Note that the evsysseq field (which is a key on the eventout table is automatically used as an index for queries.

*But what do these fields mean...?*

---

- Find a *format* which contains these fields (e.g. with Database Manager)
- Try the context help (F1)
- Try the help generally
- Try the HP ART and EUT content
- Reverse engineer
- Guess

---

**Notes.**

Sometimes fields are obvious. Many times, the most obscure fields were used once many versions ago and are only maintained for backward compatibility. Sadly, guessing is often the only option. Looking at established solutions in Connect-IT are often inadequate as it will usually use non-published APIs.

**evsysseq** Sequence number. Primary key

**evtype** Email, mail, page, etc.

**evsyssep** Separator character for the information field

**evfields** Details of message to send; five fields: email, operator, name, subject, body.

---

**Notes.**

For example, if evsyssep is the ^ character, evfields might be:

```
caffrey.aaron@advantage.com^falcon^Aaron.Caffrey^Your HP Service
Manager password has been updated by falcon. It was changed on
07/20/12 04:44:00.^Your HP Service Manager password has been updated
by falcon. It was changed on 07/20/12 04:44:00.
```

Exercise: fetching and deleting event out records

---

**Notes.**

Use the content in the last few pages to:

- Create a WSDL definition for the eventout table
- Write a Perl program to:
  1. Fetch a list of eventout sequence numbers
  2. Retrieve the event out record associated with that sequence number
  3. If it is an email, split the evfields value using the evsyssep character.  
Delete the eventout record.

For a bonus, use `Net::SMTP` or equivalent to send the email that you have split. Add some fancy HTML as well.

For an extra bonus, update the Notifications in Service Manager to send a “page”. Modify your program to turn these “page” records into an SMS. Ask your instructor for an SMS gateway you can use.

7

**Techniques**

## 7.1 Performance

### *Performance*

---

Always constrain your search on the following tables:

**probsummary** Bad – you will have many incidents.

**problem** If paging is turned on, very, very bad – you will have much incident history. (If paging is off, you will get nothing)

**device** You will have many computers

**contact** You will have many contacts

**incidents** Unless you don't use the Service Desk module

**activity**

---

#### **Notes.**

Unconstrained searches on these tables are costly and very slow.

Depending on the size of the organisation, the operator table may be very large as well – particularly if the classic Employee Self-Service interface has been deployed.

*Searching prefixes and tricks*

---

SOAP-initiated searches use the same code path as user-initiated searches.

Exact match (not prefix only)

> Search for a numeric field greater than supplied model value.

\* Anywhere in the field

---

**Notes.**



*Being gentle with big tables*

---

- `sysmodtime` is your friend. Search for changes since your last call.
- Search for newer incidents – e.g. `>IM123425`
- Create a trigger or format control to log changes to another table
- Query the activity table for BP4SM audit information.

---

**Notes.**

The mobility client access the activity table – look at `MobilityIncidentJournal1`.

## 7.2 Tables

*A brief list of common tables*

See notes for standard out-of-the-box tables.

### Notes.

A few unimportant tables have been left off for brevity. The most important are in bold.

Object Name	Table Name	Description
ActiveSLA	slaactive	Current active service level agreements
Approval	Approval	Pending approvals (e.g. for normal changes)
ApprovalLog	ApprovalLog	History of approvals given
Cart	svcCart	Service Catalogue carts. Used by the Flash-based self-service portal
CartItem	svcCartItem	Service Catalogue carts items. Used by the Flash-based self-service portal
CatalogByLanguage	svcCatLanguage	Languages supported by the service catalogue
CatalogDetail	svcCatDetail	Catalogue contents. Used by the Flash-based self-service portal
CatalogItem	joinsvcDisplay	Catalogue contents. Used by the Flash-based self-service portal
Change	cm3r	All changes on the system.
ChangeIIA	cm3r	All changes, alternate API.
ChangeOperatorInformation	cm3operatorinfo	Information about operator profiles for change
ChangeRC	cm3r	Change API as used by release control
ChangeTask	cm3t	All change tasks.
ChangeTaskOperatorInformation	cm3operatorinfo	Information about operator profiles to change tasks
ChangeTaskRC	cm3t	Change API as used by release control
<b>Company</b>	company	Access to the company table. Especially important for organisations running in multi-company mode.

Object Name	Table Name	Description
<b>Contact</b>	contacts	Access to the contents table. Used in almost all organisations to populate from HR databases.
Currency	currency	The service catalogue and request modules (not the service require module) can operate in multiple currencies.
CurrencyConvert	curconvert	The service catalogue and request modules (not the service require module) can operate in multiple currencies. This provides an API for updating exchange rates.
Delegation	ApprovalDelegation	Who has delegated their approvals, to whom, and for how long.
<b>Department</b>	dept	The departmental table. Very important for identifying who is subscribed to what service.
<b>Device</b>	device	All configuration items, including business services.
DisplayDevice	joindisplaydevice	Information about desktop monitors.
GlobalLists	globallists	You can modify global lists externally!
Inbox	inbox	The to-do list that operators see on login.
<b>Incident</b>	probsummary	Incident table
IncidentwithoutJournalUpdates	probsummary	Access to create and update incidents without having to supply a journal update.
InstalledSoftware	pcsoftware	Installed software.
<b>Interaction</b>	incidents	Service Desk tickets, including service requests.
InteractionApprovalControl	svcInteractionApprovalControl	Service Catalogue Items can have approvals.
InteractionInbox	svcInteractionInbox	Service Catalogue items can have approvals.
InteractionInfo	incidents	Alternate incidents API.
KMAttachments	kmattachments	Knowledge management attachments.
Knowledge	kmdocument	Knowledge management documents
KnowledgebaseErrors	kmknowledgebaseerrors	Knowledge management documents derived from known errors.

7.2. TABLES

Object Name	Table Name	Description
KnowledgebaseUpdates	kmknowledgebaseupdates	Documents which have been submitted for indexing but have not yet been processed.
Location	location	Contact locations
MainFrame	joinmainframe	Database of mainframes in the organisation
Model	model	The models of computer, furnishings, etc. in the system.
<b>Operator</b>	operator	All operators
Operator700	operator	Backward-compatibility interface
Operators	svcCatOperatorList	Used by Flash-based self service catalogue.
<b>Problem</b>	rootcause	Access to the problems table
Relationship	cirelationship	Access to the relationships between Configuration Items (e.g. this CI runs on this other CI).
SLA	sla	All service level agreements
SLAResponse	slaresponse	SLA response success levels
SRGlobalLists	globallists	Used by SRC.
SRCInteraction	incidents	Used by SRC.
Vendor	vendor	Companies that we buy from. See also "model"
bpreleaseinfo	bpreleaseinfo	To be able to identify the current BP4SM version. (BP4SM only).
civisualizationcat	civisualizationcat	Visualiation information
ucmdbApplication	joinapplication	Interface for UCMDB
ucmdbBusinessService	joinbizservice	Interface for UCMDB
ucmdbComputer	joincomputer	Interface for UCMDB
ucmdbNetwork	joinnetworkcomponents	Interface for UCMDB
ucmdbPrinter	joinofficeelectronics	Interface for UCMDB

Even if the mobility client is not licensed, these APIs are available and are unlikely to change in any future version. Most of the fields are self-explanatory.

**Notes.**

Object Name	Table Name
MobilityApproval1	Approval
MobilityChange1	cm3r
MobilityChangeByFields1	getrecordfieldsbykey
MobilityChangeGroups1	cm3groups
MobilityChangeJournal1	activitycm3r
MobilityChangeJournalType1	activitytype
MobilityChangeOperatorInformation1	cm3operatorinfo
MobilityChangeTask1	cm3t
MobilityChangeTaskCategory1	cm3tcategory
MobilityChangeTaskOperatorInformation1	cm3operatorinfo
MobilityChangesAwaitingApproval1	mobilitychmwtap
MobilityChangesAwaitingApprovalCount1	mobilitycountone
MobilityChangesAwaitingApprovalQuery1	mobilityquery
MobilityChangesInAssignmentGroups1	changesinassignmentgroup
MobilityChangesInAssignmentGroupsCount1	mobilitycounttwo
MobilityChangesInAssignmentGroupsQuery1	mobilityquery
MobilityContact1	contacts
MobilityGlobalList1	wsdogloballist
MobilityIncident1	probsummary
MobilityIncidentAssignmentGroups1	assignment
MobilityIncidentByFields1	getrecordfieldsbykey
MobilityIncidentClosureCode1	probcause
MobilityIncidentJournal1	activity
MobilityIncidentJournalType1	activitytype
MobilityIncidentOperatorInformation1	incidentoperatorinfo
MobilityIncidentsAssignedToMeCount1	mobilitycounttwo
MobilityIncidentsInAssignmentGroups1	incidentsinassignmentgroups
MobilityIncidentsInAssignmentGroupsCount1	mobilitycounttwo
MobilityIncidentsInAssignmentGroupsQuery1	mobilityquery
MobilityLogin1	mobilityoperatorinfo
MobilityLogout1	mobilitylogout
MobilityMassChangeOperatorInformation1	mobilityrecopinfo
MobilityMassChangeTaskOperatorInformation1	mobilityrecopinfo
MobilityMassIncidentOperatorInformation1	mobilityrecopinfo
MobilityOperator1	operator

**Index**

- authorization, 36, 37
- base64, 30, 37
- Blueprint for Service Manager, *see* BP4SM
- BP4SM, 78
- BPEL, 16
- Business Process Execution Language, *see* BPEL
- change, 53, 61, 62, 64, 65, 67, 70, 76, 78–80
- CIT, 91
- Connect-IT, 91
- Content-Length, 10
- default ns, 34
- Document Object Model, *see* DOM
- DOM, 41
- Email from Service Manager, 90
- eventout, 90–94
- evsysseq, 94
- exercise, 28, 50, 67, 83, 97
- extaccess, 27, 87, 89, 93
- eXtensible Markup Language, *see* XML
- HP Service Manager, *see* SM
- HTML, 41
- HTML POST action, *see* POST
- HTTP, 4, 8–11, 37
- HyperText Markup Language, *see* HTML
- HyperText Transfer Protocol, *see* HTTP
- incident, 13, 16, 22, 28, 53, 56, 62, 63, 67, 77, 79, 81–83, 97
- interaction, 53–56, 58–60, 63–65, 67, 76, 79, 82
- ITIL, 87, 88
- Javascript, 41, 89
- load balancer, 26
- mainframe, 90
- MIME::Base64, 30
- module, 11, 15, 26, 30–32, 34, 36, 37, 46, 47, 71
- Net::SMTP, 97
- notification, 90
- Notify Method, 90
- Perl, 11
- Perl module, *see* module
- Perl package, *see* module
- Perl::Critic, 34
- port, 26
- Port 13080, 26
- port 8080, 26
- POST, 8–10, 34
- pragma, 34
- proxy, 34
- Remote Procedure Call, *see* RPC
- Representational State Transfer, *see* REST
- REST, 26
- RPC, 4, 5, 10, 35
- service, 34
- Service Manager, *see* SM
- Simple Object Access Protocol, *see* SOAP
- SM, 7, 10, 11, 14, 16, 26, 28, 33, 37, 50, 52, 53, 62, 67, 78, 85–87, 89
- SOAP, 4, 7, 9–16, 24, 26, 30

SOAP Object Model, *see* SOM  
SOAP::Data, 40, 48–50  
SOAP::Lite, 26, 30, 32, 34, 36, 40  
SOAP::SOM, 41  
SOAP::WSDL, 30  
SOAPAction, 9–11, 24, 40  
SOM, 41  
strict, 34  
sysmodcount, 93  
sysmodtime, 93  
sysmoduser, 93  
  
tomcat, 26  
tso, 90  
  
warnings, 34  
Web Services Definition Language, *see*  
    WSDL  
Web Services Description Language, *see*  
    WSDL  
web tier, 26  
WSDL, 4, 16–26, 28, 30, 34, 39, 40, 74,  
    89  
  
XML, 4, 6, 7, 9–11, 16  
XML namespace, *see* xmlns  
xmlns, 12, 13, 15