Oracle® Communications Network Charging and Control

Data Access Pack User's and Technical Guide

Release 12.0.3

December 2019

Copyright

Contents

About This Document

Scope

The scope of this document includes all the information required to install, configure and administer the Oracle Communications Network Charging and Control Data Access Pack application.

Audience

This guide was written primarily for system administrators and persons installing and administering the DAP application. The documentation assumes that the person using this guide has a good technical knowledge of the system.

Prerequisites

Although there are no prerequisites for using this guide, familiarity with the target platform would be an advantage.

A solid understanding of Unix and a familiarity with IN concepts are an essential prerequisite for safely using the information contained in this guide. Attempting to install, remove, configure or otherwise alter the described system without the appropriate background skills, could cause damage to the system; including temporary or permanent incorrect operation, loss of service, and may render your system beyond recovery.

This manual describes system tasks that should only be carried out by suitably trained operators.

Related documents

The following documents are related to this document:

- *Advanced Control Services User's Guide*
- *Control Plan Editor User's Guide*
- *Service Management System Technical Guide*
- *Service Management System User's Guide*
- *Service Logic Execution Environment Technical Guide*
- *Data Access Pack Protocol Implementation Conformance Statement*
- *RFC 2616: Hypertext Transfer Protocol – HTTP/1.1*

Document Conventions

Typographical Conventions

The following terms and typographical conventions are used in the Oracle Communications Network Charging and Control (NCC) documentation.

Formatting Convention	Type of Information
Special Bold	Items you must select, such as names of tabs. Names of database tables and fields.
Italics	Name of a document, chapter, topic or other publication. Emphasis within text.
Button	The name of a button to click or a key to press. **Example:** To close the window, either click **Close**, or press **Esc**.
Key+Key	Key combinations for which the user must press and hold down one key and then press another. Example: **Ctrl+P** or **Alt+F4**.
`Monospace`	Examples of code or standard output.
`Monospace Bold`	Text that you must enter.
variable	Used to indicate variables or text that should be replaced with an actual value.
menu option > menu option >	Used to indicate the cascading menu option to be selected. Example: **Operator Functions > Report Functions**
hypertext link	Used to indicate a hypertext link.

Specialized terms and acronyms are defined in the glossary at the end of this guide.

Chapter 1

System Overview

Overview

Introduction

This chapter provides a high-level overview of the application. It explains the basic functionality of the system and lists the main components.

It is not intended to advise on any specific Oracle Communications Network Charging and Control (NCC) network or service implications of the product.

In this Chapter

This chapter contains the following topics.

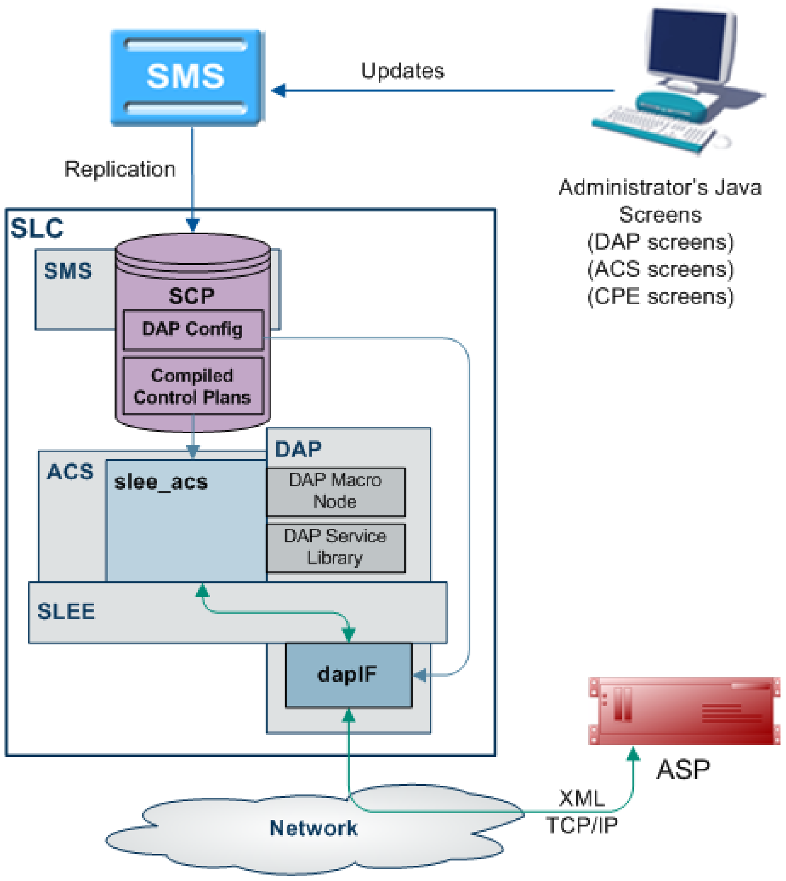
What is Data Access Pack?

Introduction

Oracle Communications Network Charging and Control Data Access Pack (DAP) provides the capability to send requests to external Application Service Providers (ASP) and optionally receive responses for further processing by the IN platform. The protocol that the system uses is determined by the ACS service library (libacsService).

Diagram

The following diagram shows the architecture of the DAP solution.



Synchronous and asynchronous connections

Responses over a connection are expected to be asynchronous if the <!--CORRELATE--> or <!--CORRELATE-ID--> tag is included in the template which specifies the request. In this case, only the ACK is checked in the (initial) synchronous response, and any later responses (using the same <!--CORRELATE--> tag) are not checked. Asynchronous mode is not supported for HTTPS connections. The listening port only supports HTTP connections.

Responses over a connection are expected to be synchronous if the <!--CORRELATE--> tag is not included in the template which specifies the request. In this case, DAP parses the synchronous response for component fields.

"Synchronous" mode for HTTPS is supported. This is done using *openssl* (on page 92) SSL sockets to encrypt a request/response pair to a remote HTTPS server.

Note: The server does not request any client-side authentication.

For more information about:

- Configuring operations, see *Operations* (on page 26).
- The <!--CORRELATE--> tag, see *Correlation* (on page 27)

Correlation

Correlation is a way to ensure that an asynchronous response is associated with an originating request. Requests that require an asynchronous response are identified by the setting of a correlated flag in the template.

HTTP and HTTPS Connections

DAP supports concurrent connections to multiple ASPs, using either HTTP or HTTPS. More than one of each type of connection can be open at once, including multiple HTTPS connections.

Supported protocols

This table describes the function of each field.

Protocol	Description
SOAP	Over HTTP or HTTPS. For more information about this protocol, see *SOAP* (on page 10).
XML	Over HTTP or HTTPS.
HPSA	HP-SA formatted XML messages over TCP. For more information about HP-SA handling, see *HP-SA* (on page 14).
PIXML	PI commands using XML. For more information about this protocol, see DAP and the PI.
LDAP	Enables DAP to send LDAP requests to the LDAP server and receive response.

Message flow

This table describes the message flow for a standard DAP message.

Stage	Description
1	When run in a control plan, the DAP Send Request feature node (*dapMacroNodes* (on page 89) and *libdapChassisActions* (on page 91)) sends a request over the SLEE to the interface specified by the *InterfaceHandle* (on page 64) parameter for the specified protocol (usually *dapIF* (on page 88)). The message is populated from the macro node configuration with the: • Protocol • Template ID • List and number of request parameters • "Wait flag" **Note:** FAST_KEY parameters are replaced with the actual run-time parameter. For example, the value <aCN> is replaced with the calling number. Other parameters are simply used "as is".
2	dapIF receives a DAP request event over the SLEE, and extracts the request template ID. dapIF uses the ID to query its cache or the SLC database for the XML request template details. For each tag name/value pair found in the request, parameter substitution is performed. If dapIF determines a response is required, it searches the script for a correlation parameter. dapIF replaces the correlation tag with a unique ID. The correlation tag is set in the XML template using the XML tag with the string defined by *correlationTagName* (on page 53) (usually <!--CORRELATION-->). This id will be used to assign the incoming response.

Stage	Description
	For more information about: • Substitution, see *Parameter Substitution* (on page 10) • Substitution for HP-SA connections, see *Parameter substitution* (on page 14)
3	If a primary TCP/IP connection is already established with the ASP but no ASP interaction is associated with that socket (that is, dapIF is idle), then this connection can be re-used immediately and dapIF sends the message. If no connection is available (and the limit has not yet been reached), dapIF opens a TCP/IP connection to the ASP specified by the host and (optional) port number found in the destination URL.
4	dapIF wraps the XML in an HTTP header and footer. For more information about how the headers and footers are constructed, see *Message Header Construction* (on page 11).
5	If the wait flag of the incoming DAP request event was set to false, dapIF immediately sends a DAP response event to slee_acs. The DAP response has Operation Status set to true to indicate success. For HTTP and HTTPS connections, if no response is expected and the request was sent on a secondary connection, it is closed unless there is no primary connection, in which case it becomes the primary connection. For HP-SA connections, the connection is closed.
6	dapIF starts a timer with the sooner of the value specified in either the template or the socket timeout (*connectionTimeout* (on page 53)).
7	dapIF checks whether there is any incoming traffic on any of the TCP/IP sockets it has previously opened with ASPs. If a TCP/IP connection is already established with the ASP and there is an ASP interaction currently associated with that socket (that is, we are waiting for a response), then the new request from the ASP is queued. If activity comes from an ASP connection and dapIF is waiting for a response from the ASP, then the incoming TCP/IP traffic is accepted and data is read. The tags are parsed and a DAP response request is constructed. For more information about parsing messages from ASPs, see *ASP Message Parsing* (on page 12). For more information about parsing HP-SA messages, see *HP-SA response messages* (on page 14). dapIF sends the message back to libdapChassisActions over the SLEE.
8	When the DAP response event is received by libdapChassisActions, and the node exits based on whether the response was a Success or an Error.

DAP and the PI

The DAP provides the ability to automatically log in to the PI when using the PIXML protocol. This feature enables PI commands to be triggered from a control plan using the DAP Send Request feature node. The DAP interface will process the PI response before returning it to the DAP Send Request feature node.

For more information on DAP feature nodes, see *Feature Nodes Reference Guide*.

Introduction to LDAP interface for DAP

Introduction

The LDAP interface for DAP (LDAP IF) is an extension to DAP that introduces support for the LDAP protocol. Specifically, it provides the capability to send DAP requests to external ASPs that provide their services using LDAP. This is an enhancement to DAP's existing ability to communicate with ASPs using XML, SOAP, HSPA and PIXML.

LDAP requests are made from within ACS control plans by using the DAP Request macro node located within the Data Access Pack feature node group.

Functionality

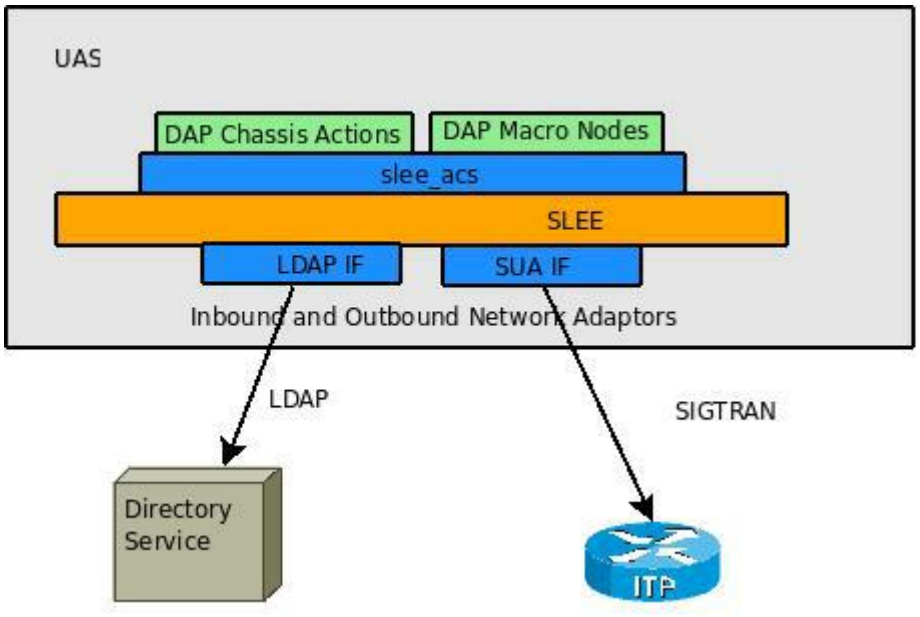
The LDAP IF is a protocol translator. The DAP Request node collects the configured request parameters and passes them to the LDAP Interface via the Service Logic Execution Environment (SLEE). The LDAP Interface embeds these per-request parameters into a standard template (stored in XML format). Then the completed template is translated into an ASN.1 format LDAP v2 or v3 request. The request is sent to the LDAP server, and the response is returned to ACS by the same mechanism.

The LDAP Interface then is responsible for:

- Managing LDAP connections (bind/unbind).
- Translating DAP requests into LDAP search requests.
- Relaying valid LDAP search requests to the appropriate LDAP ASP.
- Handling overdue responses.
- Translating LDAP search responses to DAP responses.
- Relaying valid ASP responses to SLEE ACS.

Diagram

Here is an example of the main components of the DAP service with LDAP Interface.



DAP Template Language

Introduction

DAP uses a template language to describe the format of the messages (requests) that are sent to ASPs. This template language controls variable substitution and repetition of subtrees.

Variables

There are two formats that variables can take in the XML document.

1 An empty XML element:
`<phone_number></phone_number>`
This is interpreted as a variable called **phone_number**

2 A specific format of text string:
`<<$phone_number>>`
This is interpreted as a variable called **$phone_number**.

In addition to user defined variables, there are several other variables which are substituted automatically by the DAP interface.

Variable	Description
<!--CORRELATE-->	This is substituted with the correlation ID wrapped with an element specified by the `correlationTagName` (on page 53) configuration value.
<!--CORRELATION_ID-->	This is substituted with the correlation ID. The value of the correlation ID should be treated as an opaque, variable length string.
<!--TIMESTAMP-->	This is substituted with the current time, in the format *YYYYMMDDHHmmSS*, wrapped with an element specified by the `timestampTagName` (on page 63).

Repetition of subtrees

Using profile fields contained in array profile fields, it is possible to repeat sections of a DAP request template.

This is done through the use of the **dap_main_key** attribute. When a tree has the dap_main_key specified in the root, the subtree will be duplicated for each instance of the variable in dap_main_key.

For example:

The variable **FF_numbers** is configured to point to an array of three elements (121, 122, 123), the following template stub:

```
<number_list>
    <phone_number dap_main_key="FF_numbers"><<$FF_numbers>></phone_number>
</number_list>
```

This will result in the following template being sent to the ASP:

```
<number_list>
    <phone_number>121</phone_number>
    <phone_number>122</phone_number>
    <phone_number>123</phone_number>
</number_list>
```

Multiple variables

It is possible to have multiple variables in a repeated subtree. If there are not enough elements to provide each subtree with a different value, the first value in the array will be repeated for the remaining values.

Example:

$FF_number = (121,122,123)

$FF_shortCode = (555,666)

$FF_enabled = "Yes"

```
<number_details dap_main_key="FF_number">
    <phone_number><<$FF_number>></phone_number>
    <short_code><<$FF_shortCode>></short_code>
    <enabled><<$FF_enabled>></enabled>
</number_details>
```

Will result in the following:

```
<number_details>
    <phone_number>121</phone_number>
    <short_code>555</short_code>
    <enabled>Yes</enabled>
</number_details>
<number_details>
    <phone_number>122</phone_number>
    <short_code>666</short_code>
    <enabled>Yes</enabled>
</number_details>
<number_details>
    <phone_number>123</phone_number>
    <short_code>555</short_code>
    <enabled>Yes</enabled>
</number_details>
```

Detailed example

This detailed example shows what is sent to the ASP given the variables and template used.

Variables:

CustomerName	"Bill"
$CallTo	"5551212"
$CallFrom	"5557399"

$FF_list(5550000,5550001,5550002)

Template:

```
<ProvideDiscount>
    <RequestType>Regular Call</RequestType>
    <CustomerName></CustomerName>
    <Destination><<$CallTo>></Destination>
    <Source><<$CallFrom>></Source>
    <FriendsAndFamily>
        <PhoneNumber dap_main_key="$FF_list"><<$FF_list>></PhoneNumber>
    </FriendsAndFamily>
</ProvideDiscount>
```

The following will be sent to the ASP:

```
<ProvideDiscount>
    <RequestType>Regular Call</RequestType>
    <CustomerName>Bill</CustomerName>
    <Destination>5551212</Destination>
    <Source>5557399</Source>
    <FriendsAndFamily>
        <PhoneNumber>5550000</PhoneNumber>
        <PhoneNumber>5550001</PhoneNumber>
        <PhoneNumber>5550002</PhoneNumber>
    </FriendsAndFamily>
</ProvideDiscount>
```

Template contents

This is a list of the various DAP Templates used for real time notifications and their data contents:

Wallet Expiry

- TIMESTAMP
- NOTIFICATION_NAME
- WALLET_NAME
- CLI
- PRODUCT_TYPE
- OLD_STATE
- NEW_STATE

Wallet State Change

- TIMESTAMP
- NOTIFICATION_NAME
- WALLET_NAME
- CLI
- PRODUCT_TYPE
- OLD_STATE
- NEW_STATE

Charging

- TIMESTAMP
- NOTIFICATION_NAME
- WALLET_NAME
- CLI
- PRODUCT_TYPE
- BALANCE_TYPE
- BALANCE_UNIT
- COST
- OLD_BALANCE
- NEW_BALANCE

Recharging

- TIMESTAMP
- NOTIFICATION_NAME
- WALLET_NAME
- CLI
- PRODUCT_TYPE
- BALANCE_TYPE
- BALANCE_UNIT
- AMOUNT
- OLD_BALANCE

- NEW_BALANCE

Balance Expiry

- TIMESTAMP
- NOTIFICATION_NAME
- WALLET_NAME
- CLI
- PRODUCT_TYPE
- BALANCE_TYPE
- BALANCE_UNIT
- EXPIRED_AMOUNT
- OLD_BALANCE
- NEW_BALANCE

RAR Detailed Example

```
<RAR>
 <instance></instance>
 <session></session>
 <origin_host></origin_host>
</RAR>
```

Profile Tag Formats

Introduction

The profile block values need to be converted to the receiving application's expected format, so that requests can be transmitted to other systems, and for them to communicate back.

This is impossible without a set of supported types and detailed information about what format the data is sent and received will be in.

Supported tag types

This table describes the formats and meanings of the supported ACS profile tags.

Format	Description
STRING	Any character string.
NSTRING	String containing only digits, the letters A-F, and the characters # and *.
INTEGER	Signed base 10 integer, range -2147483648 to 2147483647 inclusive.
UINTEGER	Unsigned base 10 integer, 0 to 4294967295 inclusive.

Format	Description
DATE	Supported DATE formats include: • ISO 8601 time date-time format (YYYYMMDDTHHHMMSS) • Explicit UTC timezone specifier (YYYYMMDDTHHMMSSZ) ▪ Offset from UTC with : (YYYYMMDDTHHMMSS[+-]hh[:mm]) ▪ Offset from UTC without : (YYYYMMDDTHHMMSS[+-]hh[mm]) • Extended ISO format with - and : delimiters (YYYY-MM-DDTHH:MM:SS) • Explicit UTC timezone specifier (YYYY-MM-DDTHH:MM:SSZ) ▪ Offset from UTC timezone specifier with : (YYYY-MM-DDTHH:MM:SS[+-]hh[:mm]) ▪ Offset from UTC timezone specifier without : (YYYY-MM-DDTHH:MM:SS[+-]hh[mm]) • Date only with midnight time of T000000 is added in all cases to make a date time (YYYYMMDD and YYYY-MM-DD) ▪ Explicit UTC timezone (YYYYMMDDZ and YYYY-MM-DDZ) ▪ UTC offset with : (YYYYMMDD[+-]hh[:mm]) ▪ UTC offset without : (YYYYMMDD[+-]hh[mm]) • Time only with current system date UTC added in all cases to make a date time (HHMMSS) ▪ Explicit UTC timezone (HHMMSSZ) ▪ UTC offset with : (HHMMSS[+-]hh[:mm] ▪ UTC offset without : (HHMMSS[+-]hh[mm]
BOOLEAN	String containing "1" for true or "0" for false.
DISCOUNT	String of the following format: *maxCharge*,*period1Discount*,*period2Discount*
VXMLANN	A plain text string, it is opaque to ACS. ACS expects (but does not enforce) that it is a valid URL that can be used to retrieve a VXML document. For example: **http://example.org/ExampleDocument.vxml**

XML and SOAP over HTTP/HTTPS

SOAP

When creating an XML template through the *Resources* (on page 21), the XML is parsed at a simple level for syntactical validity against the XML standard. Simple Object Access Protocol (SOAP) messages are formatted XML messages. DAP does not use syntactical parsing to check for properly formatted SOAP messages.

Parameter Substitution

When *dapIF* (on page 88) is requested to send a message to an ASP, it resolves any variables in the template as follows:

- If the tag name starts with a \ and is then followed by a $, dapIF assumes the $ is part of an existing expression that does not require substitution.
- If the tag name starts with a $, every occurrence of the tag name is replaced by the tag value.
- If the tag name does not start with a $, dapIF looks for XML tags with the given name and sets their value accordingly.
- <date> fields are populated with the date in one of the supported formats listed in the table above.

- If dapIF determines a response is required, it searches the script for a correlation parameter and replaces it with a unique ID. This substitution results in a user-specific version of the template. The correlation tag is set in the XML template using the XML tag defined by *correlationTagName* (on page 53) (usually <!--CORRELATION-->). This ID is used to assign the incoming response. The ID the CORRELATION takes place on has the form:

  ```
  <CORRELATION>HHHHHHHHTTTTTTTTSSSSSSSS</CORRELATION>
  ```

  Where:

  - H is the hostID, 8 hex characters in length.
  - T is the current time with accuracy of seconds, 8 hex characters in length.
  - S is a sequence number from 0 to FFFFFFFF, generated by the macro node each time it is invoked which ensures that the resulting string is unique.

Example: For a correlated message with tag/value pairs of: $1 and 999, and MSISDN and 00441473289900:

```
<!--CORRELATION-->
<emergencyNumber>$1</emergencyNumber>
<msisdn></msisdn>
```

This results in:

```
<CORRELATION>abcdef121234561212345678</CORRELATION>
<emergencyNumber>999</emergencyNumber>
<msisdn>00441473289900</msisdn>
```

For more information about overall message handling, see *Message flow* (on page 3).

Message Header Construction

When dapIF is constructing a message to send to an ASP it constructs a message from an initial HTTP request line, followed by HTTP headers:

```
POST path HTTP/1.1
Host: host[:port]
SOAPAction: url
User-Agent: Oracle DAP
Cache-Control: no-cache
Pragma: no-cache
Content-Type: text/html; charset=utf-8
Content-Length: length
clientUrl: http://<listenHost>:<listenPort>/ACK
```

Where:

- *path* has been extracted from the destination URL
- *host* has been extracted from the destination URL.
- *port* (if any) has been extracted from the destination URL.
- *url* is the full destination URL.
- *length* is the length (in bytes) of the template body

Notes:

- The SOAPAction header line is only sent if the protocol associated with the ASP is set to S (meaning SOAP).
- The clientUrl header is only sent if the *listenHost* (on page 55) parameter is configured.
- The request line and each of the above HTTP headers is terminated by a CRLF sequence as specified in 5.1 of *RFC 2616: Hypertext Transfer Protocol – HTTP/1.1*.

The HTTP headers are followed by an empty line, consisting only of the CRLF sequence.

The HTTP body (that is, the request template body), is sent to the ASP.

If the protocol associated with the ASP is set to S (meaning SOAP), the body is surrounded by the following SOAP header:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?><soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"><soapenv:Header/><soapenv:Body>
```

and the following SOAP footer:

```
    </soapenv:Body></soapenv:Envelope>
```

For more information about overall message handling, see *Message flow* (on page 3).

ASP Message Parsing

The first line of the response buffer contains the HTTP response status line (as defined in 6.1 of *RFC 2616: Hypertext Transfer Protocol – HTTP/1.1*). The HTTP status code is extracted from the HTTP response status line. If it is not 200 (indicating success), an alarm is raised and a DAP response event is returned to *libdapChassisActions* (on page 91).

If the original DAP request event contained response tag names, these will have been stored in the ASP queue. dapIF parses the HTTP response body to retrieve the value associated with these tags. The DAP response event will have its Operation Status set to true to indicate success. The DAP response event will be sent on the SLEE dialog which the original DAP request event arrived from.

If there is correlation and the response is the:

- First response, then the socket is still closed but the request is queued using the key of the correlation ID and with a timeout of the timeout value for a response.
- Second response (that is, it contains a correlation ID), then that correlation ID is searched for in the queue. If found, a response is generated using the parameters found in the response XML.

Note: Correlation data is received on a ‘listening’ socket (these are defined by *listenPort* (on page 55)).

For more information about overall message handling, see *Message flow* (on page 3).

HTTPS Connections

A new SSL connection is made whenever there is a queued request to be sent to the remote HTTPS server (that is, whenever an ASP connection is made on a secure connection). New connections start as HTTP, and are moved to HTTPS if a secure connection is established.

When the connection is opened:

- *openssl* (on page 92) (if not already initialized):
  - Loads our keys
  - Seeds the random number generator
  - Verifies the location of the certificates directory
- An SSL socket is created
- An SSL handshake is performed
- *Server Authentication* (on page 13) is optionally performed

Note: An error is reported if a secure connection cannot be made, or server authentication fails. However, *dapIF* (on page 88) does not abort on these errors and continues to run for other response/request pairs on other ASPs.

On a database cache reread (if the ASP has been deleted or modified) the secure socket is shut down, and if needed, restarted.

When dapIF exits normally it sends close_notify messages to the server for each open socket.

Certificate Checking

Certificate checking, when performed, is done by checking the hostname from the URL in the ASP against the common name field in the public certificate from the remote server. This check ensures that more than just the names match, by establishing that the server is who it says it is by encrypting something with its private key that matches the locally-held public key in the public certificate. This protects against spoofing-style attacks.

You can configure DAP to verify the full certificate chain. DAP can check ASP certificates against lists of certificates that have been revoked by Certificate Authorities (CAs). These lists are called Certificate Revocation Lists (CRLs). Each CA maintains their own CRL list and publishes it for customers as data files.

When DAP verifies an ASP certificate, it checks whether the certificate appears in any CRL data file. When a match is found, verification fails.

A CA can revoke a certificate for a variety of reasons, such as:

- The CA issued it in error
- The entity it certifies no longer exists
- The certificate is fraudulent

Verifying ASP Certificates

Follow these steps to configure DAP to check ASP certificates against CRLs:

Step	Action
1	Set the `CARevocationListChecking` parameter to true in the **eserv.config** file. See *CARevocationListChecking* (on page 51).
2	Establish a process to regularly obtain the CRL data file from each CA.
3	Load the CRL data files into the same directory as the ASP certificates. This directory is specified in the `certificatePath` parameter. See *certificatePath* (on page 51).
4	Run the **dapReadyCertificates.sh** utility. See *dapReadyCertificates.sh* (on page 95). **Result:** The utility processes the standard certificates and ASP certificates in the directory.

Server Authentication

Server authentication against a public certificate provided by the remote HTTPS server is available on a per-ASP basis. If not configured, the SSL connection will only have handshaking performed which ensures a minimum of session keys are used for encrypting the traffic to the HTTPS server. This protects against snoop-style attacks.

Response Validation

Checking of the response from the HTTPS server is limited to checking whether the ACK is returned as HTTP 200. Any further lines of the response are read, but are not parsed.

Certificate Management

The certificates are stored as ***.pem** files in the directory specified by *certificatePath* (on page 51).

The *dapReadyCertificates.sh* (on page 95) tool prepares the certificates into the form required by *openssl* (on page 92).

Note: There is no need to have any certificates if server authentication is not turned on.

SOAP Support Over HTTP

DAP supports SOAP by allowing the use of HTTP 1.1 as a container protocol. The basic HTTP implementation only accepts HTTP/200 as a success response, treating other success messages such as "`204 - No Content`" as error conditions.

HP-SA

Introduction

HP-SA messages are generally handled the same way as XML and SOAP messages, though the contents of the messages are different, and HP-SA is not supported over HTTPS.

For more information about overall message handling, see *Message flow* (on page 3).

Parameter substitution

When *dapIF* (on page 88) is requested to send a message to an ASP, it resolves any variables in the template as follows:

- The message_id field (in the header of each request) will be generated as a number in the range 0 to MAXINT. The message_id field will be incremented for each successive request.
- The system time is used at the time of request construction to populate the date_time field.
- All other fields are either hard-coded in the message template, or populated using the profile field values provided as DAP parameters. For more information about how these parameters are populated, see *Parameter Substitution* (on page 10).

HP-SA response messages

Response messages will be received on a new connection to the port defined by *listenPort* (on page 55). Once received, the connection will be closed. Two responses will be received in the following order for each request:

1 Command Received Acknowledgment

2 Command Processed Acknowledgment

Both have the same form (activation response).

Each response has a response ID that correlates with the request's message ID.

Response status/details command received

The following response status/details are possible for the command received acknowledgment:

```
OK/<no details>
NOK/Invalid XML
```

Response status/details command processed

The following response status/details are possible for the command processed acknowledgment:

```
OK/<no details>
NOK/Workload Failure
NOK/<platform name>:Network Problem
RB_OK/<platform name1>:OK | <error code>; …;<platform nameN>:OK | <error code>
RB_NOK/<platform name1>:OK | <error code>; …;<platform nameN>:OK | <error code>
```

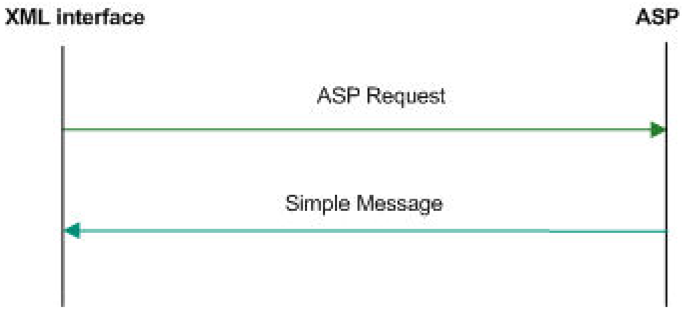
XML Interface

Description

The XML interface is a dedicated DAP interface that allows an XML script to be sent to an ASP as a request and receive another XML script as a response in order to be parsed.

Synchronous request

This diagram shows a simple example of a synchronous message flow.



Synchronous message flow

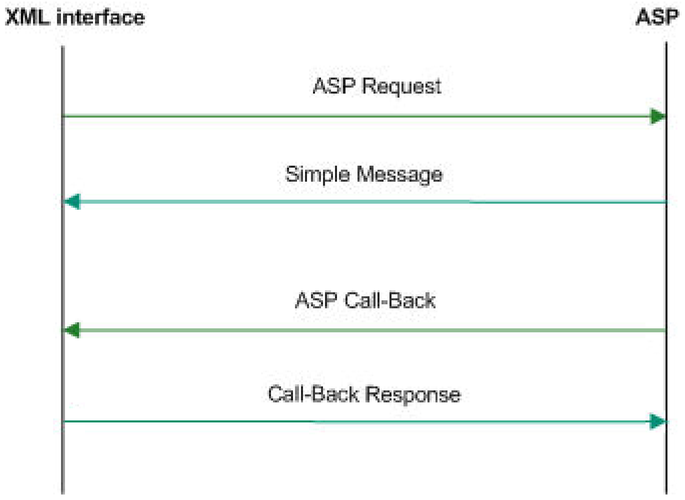
The following table describes the message flow between the XML Interface and ASP.

Step	Action
1	The XML interface sends a request to the ASP.
2	ASP returns an ACK and response on the same socket.

Note: In this example transaction, a response value is required, however a callback is not needed.

Asynchronous request

This diagram shows a simple example of an asynchronous message flow.



Asynchronous message flow

The following table describes the message flow between the XML Interface and ASP.

Step	Action
1	The XML Interface sends the request to the ASP. The request contains the `clientUrl` information.
2	The ASP returns an ACK on the same socket.
3	The ASP initiates a new request back to the XML Interface using the initial `clientUrl` information, on a new socket.
4	The XML Interface returns an ACK on the same socket.

Message contents

The XML interface is responsible for issuing a TCP/IP based message to an ASP when it receives an XML request message from a client. The message contains the specified XML string and any substituted parameters.

The response from the ASP is parsed to retrieve any requested parameter strings and these are sent back to the client in the form of an XML response.

WSDL

SOAP bindings

DAP supports WSDL 1.1 and Simple Object Access Protocol (SOAP) bindings. Since WSDL is a complex specification, some parts of the specification do not match with the capabilities of the DAP SOAP implementation.

The current SOAP implementation only allows templates to be created that make use of the `soap:body` and the `soap:fault` element. Therefore, the `soap:header` and `soap:headerfault` elements in the WSDL SOAP bindings are not supported. For more information on DAP support for SOAP bindings, see *Data Access Pack Protocol Implementation Conformance Statement*.

Note: OSD will report an error when soap:header or soap:headerfault is encountered in the WSDL file.

XSD support

XSD is supported by DAP as a type definition language.

WSDL provides an extension format allowing several different type languages to be used to describe the format of the messages used by the services, however it recommends the use of XSD. Since the XSD standard is 300 pages long, DAP only supports the use of XSD as the type definition language.

Note: DAP will report an error if a type definition language other than XSD is encountered in the WSDL file.

WSDL styles

Web Services Description Language <WSDL) allows several different encodings to be specified, each resulting in the message being presented with a different style.

DAP attempts to be as permissive as possible with the style and use declarations in accordance with the WSDL document.

DAP supports the following styles:

- style=rpc
- style=document

DAP supports the following use declarations:

- use=literal
- use=encoded

Other encodings

Since the SOAP specification allows different encodings other than XML to be used to transfer the information, it is necessary to specify which encodings DAP will support. DAP only supports the soap-encoding.

However, even in soap-encoding, there are issues with the protocol. The portions of the standard that define how low-level types are encoded are supported but encoding data as references is not supported.

Note: DAP will report an error if an encodingStyle other than soap-encoding is encountered in the WSDL file.

Transmission services

DAP supports the One-way transmission primitive as well as the Request-response transmission primitive.

WSDL supports the definition of services where the view of the service is from the client. However, SOAP does not typically support these definitions. To determine which side of the protocol is being defined, DAP only supports WSDL files generated for a server in the following cases where the server receives a request:

- With no response
- And provides a response

Statistics

Introduction

The DAP macro-node (*dapMacroNodes* (on page 89)) collects statistics using the standard Service Management System statistic mechanism and stores them to the SMF database. Refer to *SMS Technical Guide* for details on how the statistics are collected.

Statistics collected

This table describes the statistics that are collected.

Statistic	Description
OP-SENT	Count of the number of new requests sent to an ASP. This statistic is incremented each time a new request is sent to an ASP. The name of the template used is put in the statistics “details” field.
OP-SUCCESS	This statistic is incremented each time an ASP returns a successful response. The name of the template used is put in the statistics “details” field.
OP-FAIL	This statistic is incremented each time an ASP returns a failure response. The name of the template used is put in the statistics “details” field.

Note: These statistics have a period of 300 seconds.

ASP based statistics

ASP is recorded in the detail field for ASP based statistics.

The table describes the ASP statistic generated for each statistic, if the ASP column indicates "Y". The error values shown here returned by the XML interface to the node.

ASP	Statistic	Error
Y	Success	-
Y	Unable to connect to ASP	connectconnect
Y	Unable to send request to ASP	cannotwrite
Y	ASP Protocol Failure (HTTP Error or ASP closed the connection prior to initial HTTP/ response)	asperror
Y	ASP closed the connection while the response is being read	noresponse
Y	An operation has timed out waiting for a response	responsetimeout
Y	An expected parameter is missing from the response	missingparam
Y	A valid correlation ID is missing from the response	malformeddata
Y	A reply to the request has been received after the call back is received	sequenceerror
Y	A request has been resent	retryattempt
Y	Too many requests are waiting for callbacks from an ASP	maxqueuesizereached

Notes:

- Only external errors will be reported as statistics.
- The total number of requests generated by the system are recorded as a statistic.
- The total number of requests for each ASP are recorded as a statistic.

LDAP IF Statistics

The following table lists and describes the statistics generated by the LDAP IF.

Note:

- These statistics are recorded under the “DAP” application.
- The LDAP IF generates only these LDAP-specific statistics. It does not generate any of the other LDAP statistics.
- These statistics are included in the SMF_STDEF_DAP replication group. If DAP is installed and configured for a SLC node, then these LDAP statistic definitions will automatically also be available on that SLC node with no operation action required.

Statistic	Description
MAX_LDAP_LATENCY	Maximum latency of LDAP responses in milliseconds during the period.
MEAN_LDAP_LATENCY	Average latency of LDAP responses in milliseconds during the period.
MIN_LDAP_LATENCY	Minimum latency of LDAP responses in milliseconds during the period.
NUM_LDAP_DAP_SEARCH_REQUESTS	Total number of DAP search requests received from ACS during the period.

NUM_LDAP_MISMATCHED_RESPONSES	Number of unmatched LDAP responses received. These are responses for which we have no matching request recorded. This will typically indicate a late response, where we have timed-out, returned failure to ACS, and discarded the original request.
NUM_LDAP_NEGATIVE_RESPONSES	Number of unsuccessful LDAP responses, where the server has not returned subscriber data. This may be due to a malformed request, or to a valid request referencing a subscriber unknown to the ASP.
NUM_LDAP_SUCCESS_REQUESTS	Number of requests where processing was successful.
NUM_LDAP_TIMED_OUT_RECEIVE	Total time-outs waiting for message reply from the ASP.
NUM_LDAP_TIMED_OUT_SEND	Total time-outs received while sending messages. This occurs when the connection set-up process takes so long that the request has timed-out before it is sent.

LDAP IF Reports

A single SMF report is installed for the LDAP IF. This is a text format report which counts the average LDAP request rate grouped by hour.

Report Name: “DAP LDAP Requests per Second”
Parameter #1: Start Date (YYYYMMDD)
Parameter #2: End Date (YYYYMMDD)

Example Output:

```
AVERAGE NUMBER OF LDAP REQUESTS PER SECOND
==========================================

20 August 2016, 21:16:00
Period: 01 January 2016 to 01 January 2017

Date/Time        Requests/s
---------------- ----------
...
19/08/2016 05:00 0
19/08/2016 06:00 0
19/08/2016 07:00 0
19/08/2016 08:00 0
19/08/2016 09:00 < .01
19/08/2016 10:00 .01
19/08/2016 11:00 < .01
20/08/2016 12:00 < .01
20/08/2016 01:00 < .01
20/08/2016 02:00 < .01
20/08/2016 03:00 < .01
20/08/2016 04:00 0
20/08/2016 05:00 .87
20/08/2016 06:00 .33
20/08/2016 07:00 0
20/08/2016 08:00 0
20/08/2016 09:00 0
```

```
20/08/2016 10:00 0
...
```

Notes:

- For the period, the end date is up to midnight at the start of the day, so, for example, asking for '2016-08-28' to '2016-08-28' will give you data for the whole day.
- The Requests/s number is the average requests per second for the hour up to the given hour, for example 6:00, is for the hour between 5:00 and 6:00.
- The average is calculated by dividing the total for the hour by 3600. 0.01 means about 36 requests were sent in that hour. < 0.01 means less than that number were sent (but more than 0 requests).

Accessing the DAP application

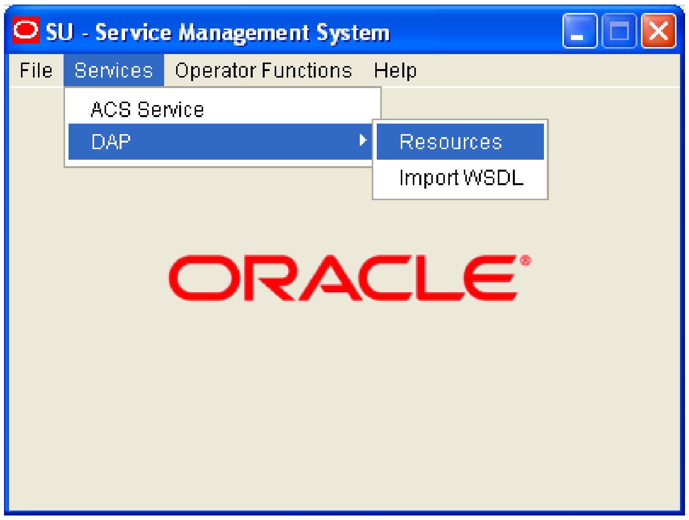
Introduction

You access the Data Access Pack (DAP) application screens from the Service Management System (SMS) UI.

To begin configuring the DAP application, the SMS screens must first be configured and running. For more information about how to set up the SMS screens, see *Service Management System User's Guide*.

SMS main menu

Here is an example of the Service Management System main menu showing the DAP menu options.



DAP screens

There are two DAP screens:

- *Resources* (on page 21)
- *Import WSDL* (on page 37)

Chapter 2

Resources

Overview

Introduction

This chapter explains how to use the Resources screen to configure Application Service Providers (ASP) and make operations available to DAP.

In this chapter

This chapter contains the following topics.

Resources Screen

Introduction

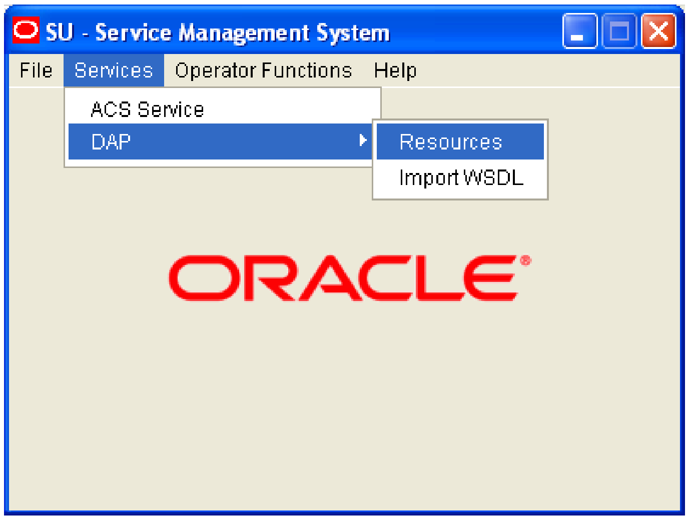
The DAP Resources screen is used to configure the Oracle Communications Network Charging and Control Data Access Pack (DAP) application. It contains these tabs:

- *ASP* (on page 22)
- *Operations* (on page 26)
- *Operation Sets* (on page 34)

Accessing the Resources screen

Follow these steps to open the DAP Resources screen.

Step	Action
1	Select the **Services** menu from the SMS main screen.
2	Select **DAP**.
3	Select **Resources**.

Step	Action
	 SU - Service Management System File Services Operator Functions Help ACS Service DAP Resources Import WSDL ORACLE **Result:** You see the Resources screen.

ASP

Introduction

You use the **ASP** tab to configure the ASPs that are available to the DAP application.

About Specifying URLs

When you configure an ASP you must specify the URL to which requests are sent. You can also specify an URL as the file location to use for importing WSDLs. The URL can contain either the host name of the ASP or its Internet Protocol (IP) address, and an optional port number. You can specify an IP version 6 (IPv6) or an IP version 4 (IPv4) address.

If you specify an IPv6 address and port number in the URL, then you must enclose the IPv6 address in square brackets []; for example: `[2001:db8:`*`n`*`:`*`n`*`:`*`n`*`:`*`n`*`:`*`n`*`:`*`n`*`]` where *`n`* is a group of 4 hexadecimal digits. The industry standard for omitting zeros is also allowed when specifying IP addresses. Note that square brackets are not required for IPv4 addresses or if the port number is not included in the URL.

Example IPv4 and IPv6 addresses

```
192.0.2.1:4000
[2001:db8:0000:1050:0005:0600:300c:326b]:3004
[2001:db8:0:0:0:500:300a:326f]:1234
2001:db8::c3
```

If the URL starts with "http:" or "https:", then you must append a trailing forward-slash, "/", after the host name and port. For example:

`http://`*`domain_name`*`:`*`port`*`/`

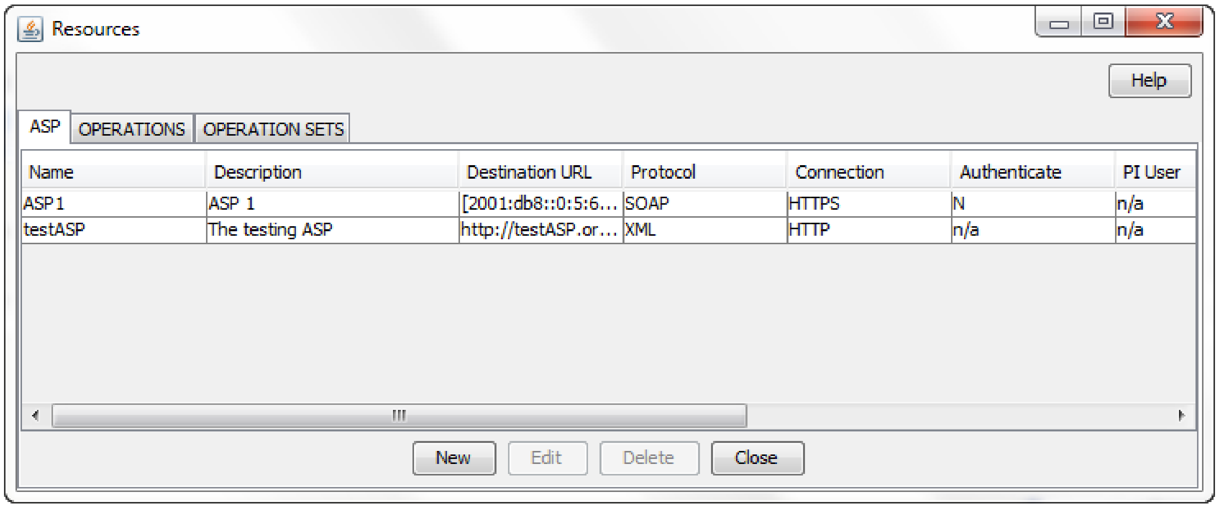
`http://`*`domain_name`*`:`*`port`*`/mydoc`

Where:

- *`domain_name`* is the URL domain name or IP address.
- *`port`* is the port number to use.

ASP tab

The **ASP** tab gives a view of all the ASPs that have been created for the service. Here is an example.



Name	Description	Destination URL	Protocol	Connection	Authenticate	PI User
ASP1	ASP 1	[2001:db8::0:5:6...	SOAP	HTTPS	N	n/a
testASP	The testing ASP	http://testASP.or...	XML	HTTP	n/a	n/a

ASP tab fields

This table describes the function of each field on the **ASP** tab. The records that display on the **ASP** tab are sorted by the **Name** field.

Field	Description
Name	(Required) The unique name for this ASP connection. The ASP name can be up to 64 alphanumeric characters in length. **Note:** When you save a new ASP, this field becomes read only and may not be edited.
Description	(Required) A description of the ASP that can be up to 64 characters in length.
Destination URL	(Required) The destination URL to which requests are sent. Specify either the host name of the ASP or the IP address, and an optional port number. The destination URL can be up to 256 characters in length. For more information about specifying the destination URL, see *About Specifying URLs* (on page 22).
Protocol	The protocol field contains the protocol that should be used when interacting with the ASP. The protocol can be one of: XML, SOAP, HPSA, PIXML, or LDAP. **Note:** You define which protocols are available to ASPs in the *Mapping parameters* (on page 64) configuration.
Connection	The type of the connection (normal/HTTP or encrypted/HTTPS). **Note:** This field is only available for the SOAP and XML protocols.
Authenticate	Whether to check server authentication. **Note:** This field is only available for HTTPS connectivity. For more information about server authentication, see *HTTP and HTTPS Connections* (on page 3).
PI User	The PI user for whom PI commands may be triggered by the DAP Send Request feature node. The PI logs in the specified PI user automatically when a PI

Field	Description
	command is first triggered. **Note:** This field is only available for the PIXML protocol. This field is populated by the **PI Users** tab in the PI Administration screen. For more information, see *PI User's Guide*.
Destination Auth User	The user name to authenticate with the ASP server using HTTP basic authentication. **Note:** This field is only available for the SOAP, XML, and LDAP protocols.
Max Secondary Connections	This field contains the maximum number of secondary connections that can be created by the DAP interface.

ASP configuration

The table on the **ASP** tab in the Resources screen displays the ASPs that are currently available to the DAP. Follow these steps to edit or create an ASP.

For more information about the fields on this screen, see *ASP tab fields* (on page 23).

Step	Action
1	On the ASP tab do one of the following: • If you want to create a new ASP, click **New**. • If you want to edit an existing ASP, select the ASP record in the table and click **Edit**. The *ASP Configuration screen* (See example on page 25) displays.
2	If you are creating a new ASP, enter the name of the ASP in the **Name** field. **Note:** When you save a new ASP, this field becomes read only and may not be edited.
3	Enter a description for the ASP in the **Description** field.
4	Select the protocol to use when interacting with the ASP from the **Protocol** list. The available protocols depend on the *Mapping parameters* (on page 64) configuration. The following table lists the additional fields that become enabled if you select one of the listed protocols.

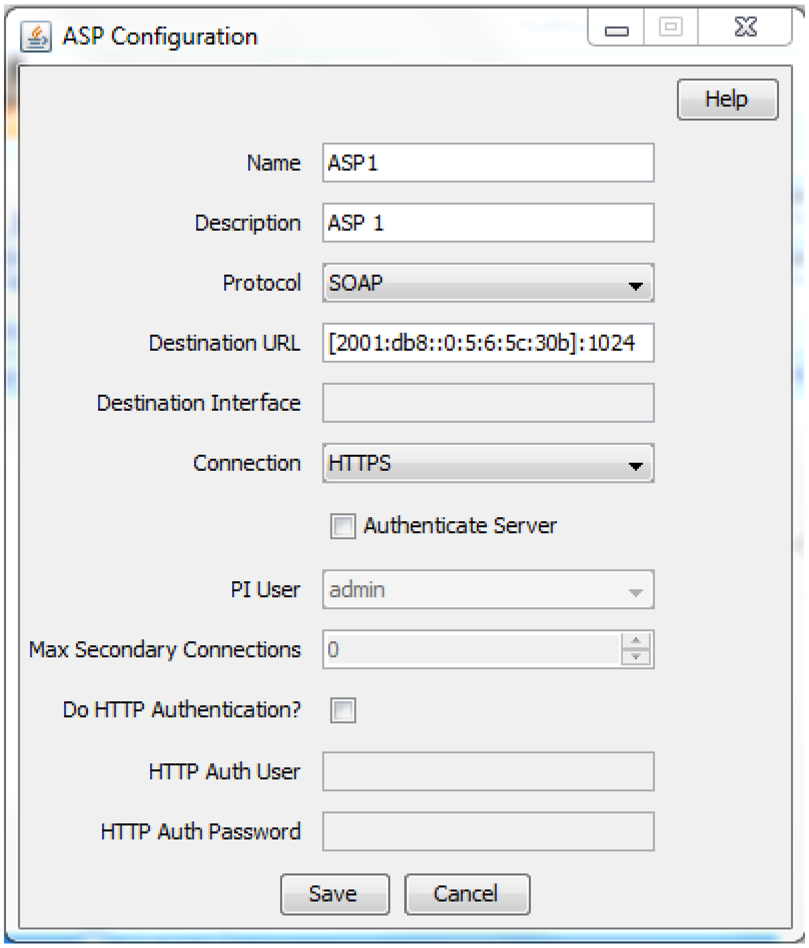
Protocols	Fields
XML, SOAP, and LDAP	Connection Do HTTP Authentication?
PIXML	PI User
SLEE	Destination Interface

Step	Action
5	Specify the destination URL to which requests should be sent in the **Destination URL** field. You can specify the host name of the ASP or the IP address, and an optional port number. For more information, see *About Specifying URLs* (on page 22).
6	If you selected `SLEE` in the **Protocol** field, then specify the SLEE interface that the DAP should use when sending requests in the **Destination Interface** field.
7	If the protocol is XML or SOAP, select whether the connection should use HTTP or HTTPS.
8	If you selected an HTTPS connection and you want to ensure that it matches the server name in the HTTPS certificate, select the **Authenticate Server** check box.

Step	Action
9	If the protocol is PIXML, select the required PI user from the **PI User** list.
10	If required, specify the maximum number of secondary connections that can be created by the DAP interface in the **Max Secondary Connections** field.
11	If the protocol is XML or SOAP and you want to configure HTTP authentication then perform the following steps: a. Select the **Do HTTP Authentication?** check box. b. In the **Destination Auth User** and the **Destination Auth Password** fields, enter the user name and password to authenticate with, .
12	Click **Save**. The ASP is saved to the database. **Note:** When editing an ASP description, the **Save** button is disabled if the new description is the same as the original description.

ASP Configuration screen

Here is an example ASP Configuration screen.



Name	ASP1
Description	ASP 1
Protocol	SOAP
Destination URL	[2001:db8::0:5:6:5c:30b]:1024
Destination Interface	
Connection	HTTPS
	☐ Authenticate Server
PI User	admin
Max Secondary Connections	0
Do HTTP Authentication?	☐
HTTP Auth User	
HTTP Auth Password	

Deleting an ASP

Follow these steps to delete an ASP from the service.

Step	Action
1	Select the ASP that you want to delete in the table on the **ASP** tab.
2	Click **Delete**. The Delete Confirmation prompt displays. **Note:** An error is raised if the ASP has any operations attached to it.
3	Click **OK**. The ASP record is removed from the database.

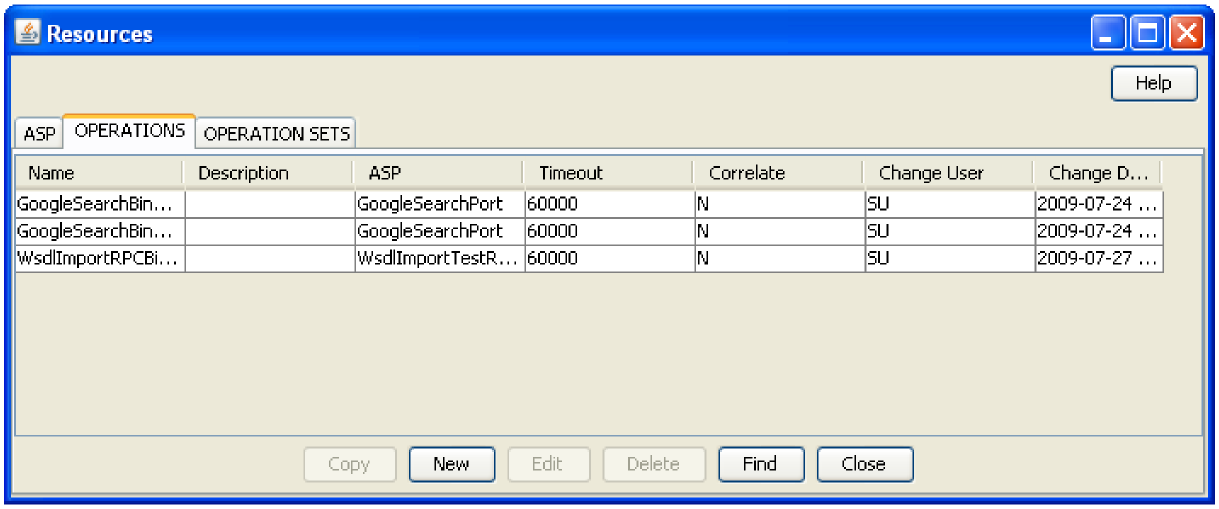
Operations

Introduction

You use the **Operations** tab on the Resources screen to configure the operations that are available to the DAP application. These operations are used by the Send Request or DAP Request feature nodes.

Operations tab

The **Operations** tab contains a view of all the operations created in the service. Here is an example.



Name	Description	ASP	Timeout	Correlate	Change User	Change D...
GoogleSearchBin...		GoogleSearchPort	60000	N	SU	2009-07-24 ...
GoogleSearchBin...		GoogleSearchPort	60000	N	SU	2009-07-24 ...
WsdlImportRPCBi...		WsdlImportTestR...	60000	N	SU	2009-07-27 ...

Operations tab fields

This table describes the function of each field in the **Operations** tab.

Field	Description
Name	(Required) The name of the operation. The operation name can be up to 64 alphanumeric characters in length. **Note:** When you save a new operation, this field becomes read only and may not be edited.
Description	(Required) A description of the operation. The description can be up to 64 characters in length.

Field	Description
ASP	The ASP associated with the operation. This field is populated by the records configured on the *ASP* (on page 22) tab.
Timeout	The number of milliseconds that *dapIF* (on page 88) should wait for a response from the ASP.
Correlate	Indicates return parameters are expected. See *Correlation* (on page 27).

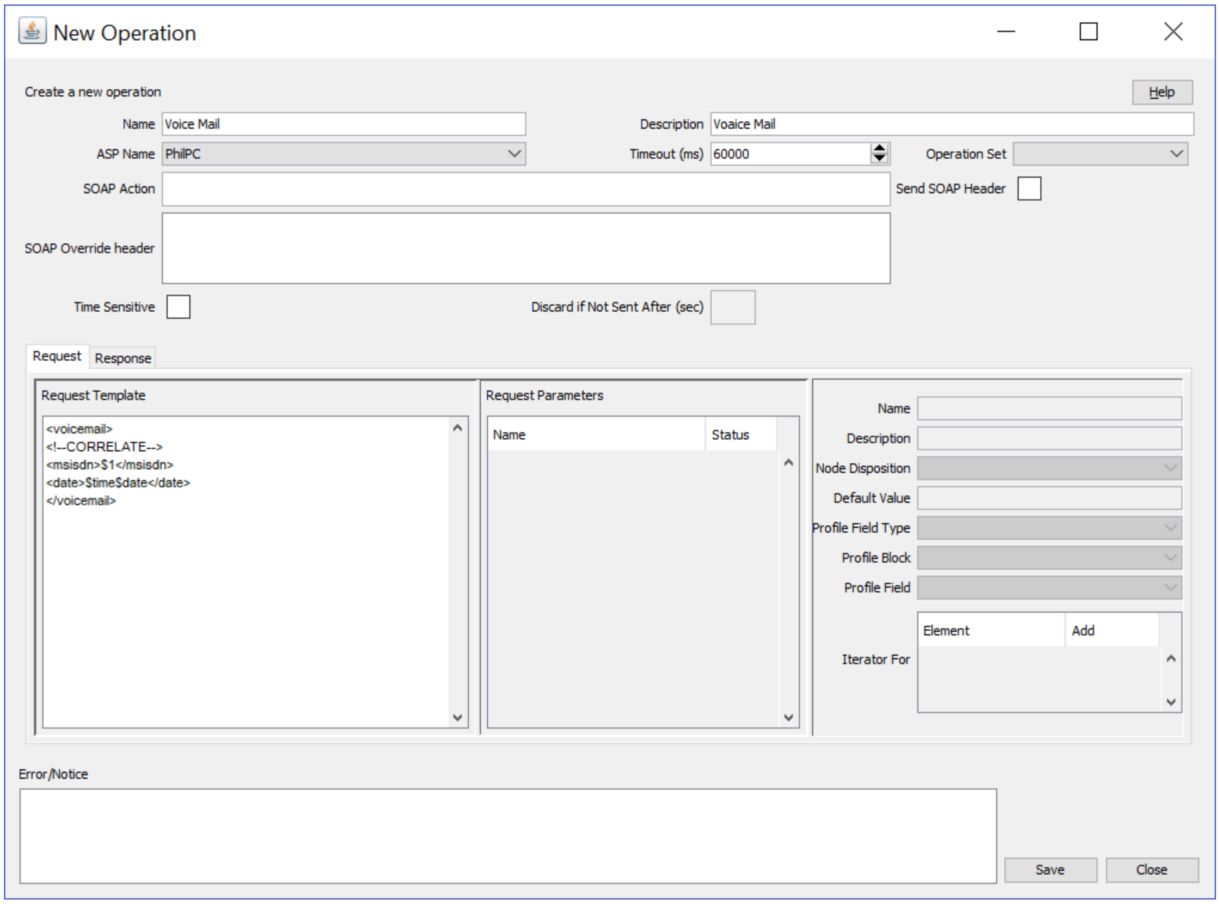
Correlation

When the correlation comment `<!--CORRELATE-->`, or `<!--CORRELATE-ID-->` is specified within the request script, it is implied that return parameters will be sent back by the ASP. The return parameters can be stored within ACS in a user defined profile block and field.

The correlation tag is only supported in asynchronous connections. For more information about asynchronous connections, see *Synchronous and asynchronous connections* (on page 2).

New Operation screen - Request tab

Here is an example **Request** tab in the New Operation screen.

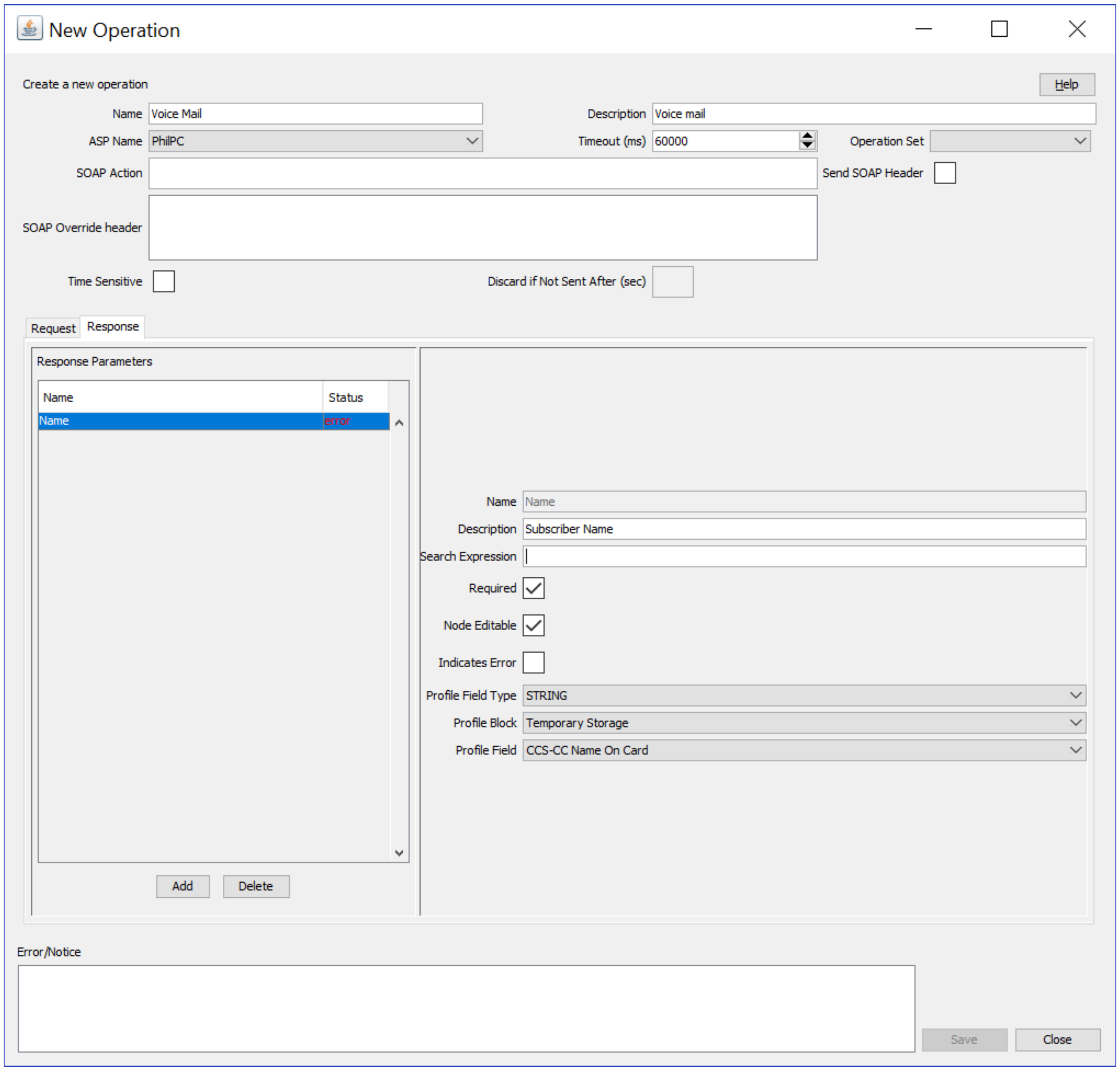


Name: Voice Mail
Description: Voaice Mail
ASP Name: PhilPC
Timeout (ms): 60000
Operation Set:
SOAP Action:
Send SOAP Header: ☐
SOAP Override header:
Time Sensitive: ☐
Discard if Not Sent After (sec):

```
<voicemail>
<!--CORRELATE-->
<msisdn>$1</msisdn>
<date>$time$date</date>
</voicemail>
```

New Operation screen - Response tab

Here is an example **Response** tab in the New Operation screen.



Name: Voice Mail
Description: Voice mail
ASP Name: PhilPC
Timeout (ms): 60000
Operation Set:
SOAP Action:
Send SOAP Header: ☐
SOAP Override header:
Time Sensitive: ☐
Discard if Not Sent After (sec):

Name	Status
Name	error

Name: Name
Description: Subscriber Name
Search Expression:
Required: ✓
Node Editable: ✓
Indicates Error: ☐
Profile Field Type: STRING
Profile Block: Temporary Storage
Profile Field: CCS-CC Name On Card

Adding an operation

Follow these steps to add a new operation and the associated request and response templates.

For more information about the fields on the **Operations** tab, see *Operations tab fields* (on page 26).

Step	Action
1	On the **Operations** tab, click **New**. The **Request** tab in the *New Operation screen* (See example on page 27) displays.
2	Type the name of the new operation in the **Name** field. **Note:** After you save an operation, the name is no longer editable.
3	Type the description for the operation template in the **Description** field.
4	Select the ASP to associate with the operation from the **ASP Name** list.
5	(Optional) Change the timeout specified for a response to the DAP interface from the ASP by selecting a different value in the **Timeout** field.

Step	Action
6	(Optional) Select the operation set for this operation from the **Operation Set** list. For more information, see *Operation Sets* (on page 34).
7	If the ASP uses the SOAP protocol then you can optionally configure the following: • To allow the SOAP action in the HTTP header field to be overridden, enter the SOAP action override in the **SOAP Action** field. By default, the DAP sends the destination URL that is configured for the ASP. • To send the SOAP Header tag select the **Send SOAP Header** check box. The DAP sends an empty SOAP Header tag that uses the form: `<soapenv:Header/>`. Select to send the SOAP Header tag only if the ASP is able to accept an empty SOAP Header tag. • To define SOAP headers that will be used only by this template, enter the SOAP header definitions in the **SOAP Override header** field. The specified SOAP header overrides the standard XML and SOAP header tags, or the override header tags defined for all DAP operations in the `soapHeaderOverride` parameter. See *soapHeaderOverride* (on page 61) for details.
8	Select the **Time Sensitive** check box to discard SOAP requests after a certain period of time if the destination ASP cannot be reached. By default, this check box is unchecked. In the **Discard if Not Sent After (sec)** field, enter the total number of seconds which the request will be retried for before it is discarded. **Note**: The **Discard if Not Sent After (sec)** field is enabled only if the **Time Sensitive** check box is selected. If set, **Discard if Not Sent After (sec)** will take priority over *discardPendingQueueRequestsAfterSeconds* (on page 54) parameter.
9	In the **Request Template** section create the template for the request by entering a valid script in XML format. For more information about script formats, see *Script Format* (on page 29).
10	Configure the variables listed in the **Request Parameters** area that have an error status and corresponding message in the **Error/Notice** area. A variable is any item within an element that is enclosed in double angle brackets <<>> and that has a $ prefix, or an empty element. For information about variable configuration, see *Request Parameter Configuration* (on page 30). For more information about variables, see *Variables* (on page 6).
11	Add and configure the response parameters that you require on the **Responses** tab. See *Response Parameter Configuration* (on page 31).
12	Click **Save.** The **Save** button is available when there are no errors listed in the **Error/Notice** area.

Script Format

The script entered in the script text field of a template forms the body of the request template. The expected format is XML.

A parameter is defined either by using the $ prefix or by using an empty element:

```
<voicemail>
    <!--CORRELATE-->
    <msisdn>$1</msisdn>
    <language></Language>
    <date>$time$date</date>
</voicemail>
```

The parameters specified in the example are $1, language, $date and $time.

DAP requests sent through the SLEE have a limitation of 10 parameters per script. This limit is enforced at run time. A parameter can be up to 32 characters in length.

Note: The "$" character will not be treated as a parameter within the text of a tag if it is escaped, that is, "\$".

Request Parameter Configuration

Follow these steps to configure operation request parameters.

Step	Action
1	Click on the parameter to configure. The parameter name is displayed in the **Name** field on the **Request** tab and the editable parameter fields are made available. The **Iterator For** panel is populated with all the request template elements.
2	In the **Description** field type a description for the parameter.
3	Select the feature node behavior for the parameter from the **Node Disposition** list. **Note:** The option you select determines whether or not you are required to configure a run-time parameter in a DAP feature node that sends XML requests. Select one of the following options: • `Hidden` – If the parameter will not be visible in the feature node configuration window and the parameter must be configured in this screen • `Text` – If the parameter will be completed in the feature node configuration window as a text field • `Profile Block –` If the parameter will be completed in the feature node configuration as a profile block location • `Either` – If the parameter will be completed in the feature node configuration either as a text field or as a profile block location • Transient – If you do not want to provide a run-time parameter and you also want to send the text defined in the XML template without any further processing. **Note:** The Transient option enables the DAP to process XML requests containing attributes where the parameters are not recognized.
4	If you set **Node Dispostition** to `Hidden`, then specify a value for the parameter by configuring either, or both, the following fields: • (Optional) Enter the default value for the parameter in the **Default Value** field. • (Optional) Select the parameter location from the following lists: **Profile Field Type**, **Profile Block** and **Profile Field.** **Note:** You can configure any missing profile fields in the ACS Configuration window in the ACS UI. For information about configuring profile fields, see *Advanced Control Services User's Guide*.
5	If you want to use this parameter as an iteration value for an element in a request template, scroll through the **Iterator For** list to find the element, and select the **Add** check box. The parameter is added to the request template element as an iteration.
6	Repeat steps **1** to **5** for each parameter that you need to configure.

Response Parameter Configuration

Follow these steps to add and configure any required response parameters.

Step	Action
1	Select the **Response** tab in the *New Operation screen* (See example on page 27).
2	To add a new parameter, click **Add**. A new editable row is added to the table in the **Response Parameters** area.
3	Type the parameter name in the empty **Name** field.
4	Press Enter, or click on the **Name** field heading. The status field displays `error`, and the parameter configuration fields are enabled. The error is also listed in the **Error/Notice** area.
5	Type a description for the parameter in the **Description** field.
6	Type a valid XPath search expression in the **Search Expression** field.
7	If this parameter must be returned, select the **Required** check box.
8	If this parameter can be edited in the feature nodes, select the **Node Editable** check box.
9	If the return of this parameter is an error condition, select the **Indicates Error** check box. **Note:** The **Required** and **Indicates Error** check boxes are mutually exclusive.
10	Select the profile field location for this parameter from the **Profile Field Type**, **Profile Block** and **Profile Field** lists. **Note:** If an expected profile field is missing, it can be added via the ACS Configuration screens. For information about configuring profile fields, see *Advanced Control Services User's Guide*.
11	Repeat steps 2 to 10 for all the response parameters that you want to add.
12	Click **Save**. **Note:** The **Save** button is enabled when all errors indicated in the **Error/Notice** area are resolved.

Editing an Operation

Follow these steps to edit an operation on the **Operations** tab.

For more information about the fields on the **Operations** tab, see *Operations tab fields* (on page 26).

Step	Action
1	Select the operation that you want to edit on the **Operations** tab and click **Edit**. The **Request** tab for the operation displays in the Edit Operation screen. The **Name** field for the operation is read only.
2	(Optional) Change the description for the operation in the **Description** field.
3	(Optional) Select a different ASP to associate with the operation from the **ASP** list.
4	(Optional) Change the timeout specified for a response to the DAP interface from the ASP by selecting a different value in the **Timeout** field.
5	(Optional) Select a different operation set from the **Operation Set** list.

Step	Action
6	If the ASP uses the SOAP protocol then you can optionally configure the following: • To allow the SOAP action in the HTTP header field to be overridden, enter the SOAP action override in the **SOAP Action** field. By default, the DAP sends the destination URL that is configured for the ASP. • To send the SOAP Header tag select the **Send SOAP Header** check box. The DAP sends an empty SOAP Header tag that uses the form: `<soapenv:Header/>`. Select to send the SOAP Header tag only if the ASP is able to accept an empty SOAP Header tag. • To define SOAP headers that will be used only by this template, enter the SOAP header definitions in the **SOAP Override header** field. The specified SOAP header overrides the standard XML and SOAP header tags, or the override header tags defined for all DAP operations in the `soapHeaderOverride` parameter. See *soapHeaderOverride* (on page 61) for details.
7	(Optional) Change the **Request Template** script and update the configuration for any parameters listed in the **Error/Notice** area. See *Request Parameter Configuration* (on page 30) for more information.
8	If required, select the **Response** tab and update the parameters configured on the tab by adding, editing and deleting response parameters: • To add a new response parameter or edit an existing response parameter, see *Response Parameter Configuration* (on page 31). • To delete a response parameter select the parameter you want to delete and click **Delete**. **Tip:** If you delete a parameter by mistake, then add a new parameter with the same name as the deleted parameter.
9	Click **Save**. **Note:** The **Save** button is enabled only when there are no errors listed in the **Error/Notice** panel.

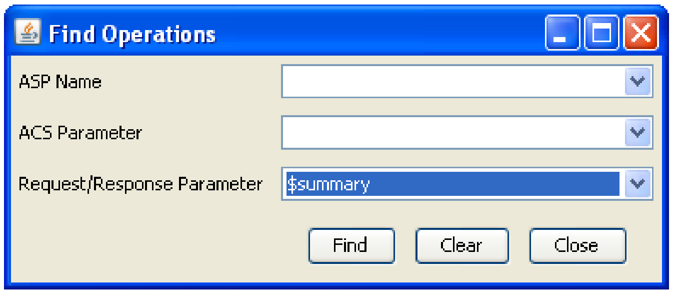
Finding a DAP template

Follow these steps to search for a DAP operation.

Step	Action
1	On the **Operations** tab, click **Find**. **Result:** You see the *Find Operations screen* (See example on page 32).
2	Enter the search criteria in one of the query fields using the drop down list and click **Find**. **Result:** When you click **Find** a query is triggered and the first 100 records matching the value in the query field will be returned. The results appear in the table on the Operations tab. **Note:** If you select a value in more than one query field then the **Find** button will be disabled.

Find Operations screen

Here is an example Find Operations screen.



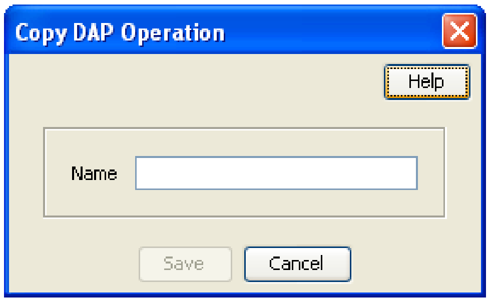
Copying DAP templates

Follow these steps to copy a defined DAP operation.

Step	Action
1	On the **Operations** tab, select the operation to copy.
2	Click **Copy**. **Result:** You see the *Copy DAP Operation screen* (See example on page 33).
3	In the **Name** field, type a unique name for the new operation.
4	Click **Save**. **Result:** The operation will be saved to the database under the new name. **Note:** This operation will have exactly the same details as the original operation. See *Editing an Operation* (on page 31) to change the operation details.

Copy DAP Operation screen

Here is an example Copy DAP Operation screen.



Deleting an operation

Follow these steps to delete an Operation from the service.

Step	Action
1	In the table on the **Operations** tab, select the operation to delete.
2	Click **Delete**. **Result:** You see the Delete Confirmation prompt. **Note:** An error is raised if the operation exists in an active control plan.
3	Click **OK**. **Result:** The operation is removed from the database.

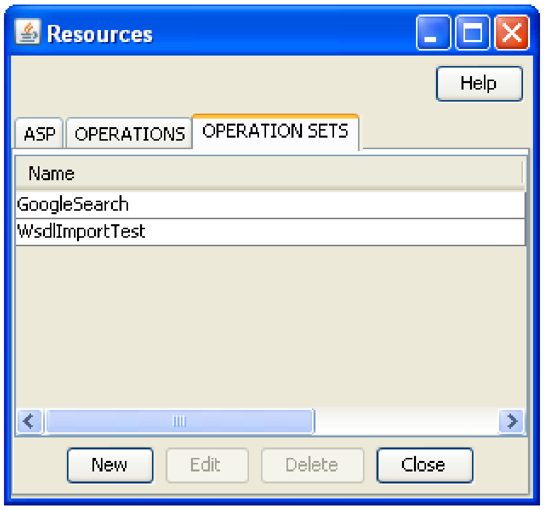
Operation Sets

Introduction

Operation sets are used to limit the operations to selected users.

Operation Sets tab

Here is an example **Operation Sets** tab.



GoogleSearch
WsdlImportTest

Add or edit an operation set

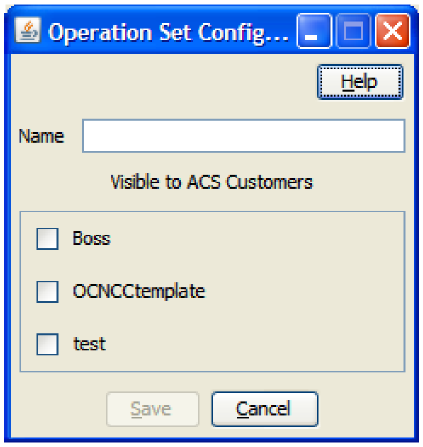
The table in the **Resources** screen **Operation Sets** tab displays the operation sets that are currently available in the system.

Follow these steps to add a new operation set or edit an existing set.

Step	Action
1	On the **Operation Sets** tab: • To create a new set, click **New** • To edit an existing set, select the set, then click **Edit** **Result:** You see the *Operation Set Configuration screen* (See example on page 34).
2	In the **Name** field, enter or edit the name of the operation set.
3	Select the check box for all the listed users you want in this operation set.
4	Click **Save.**

Operation Set Configuration screen

Here is an example Operation Set Configuration screen.



Name

Visible to ACS Customers

- [ ] Boss
- [ ] OCNCCtemplate
- [ ] test

Deleting an operation set

Follow these steps to delete an operation set.

Step	Action
1	In the table on the **Operation Sets** tab, select the operation set to delete.
2	Click **Delete**. **Result:** You see the Delete Confirmation prompt.
3	Click **OK**. **Result:** The operation set is removed from the database.

Chapter 3

Import WSDL

Overview

Introduction

This chapter explains how to use the DAP Import WSDL screen.

In this chapter

This chapter contains the following topics.

Import WSDL Screen

Introduction

The Oracle Communications Network Charging and Control Data Access Pack DAP Import WSDL screen allows you to import and configure predefined web services from ASPs.

It contains these functions:

- *Import WSDL files* (on page 38)
- *ASP Configuration* (on page 22)
- *Operation Request Configuration* (on page 40)
- *Operation Response Configuration* (on page 42)

What is WSDL?

Web Services Description Language (WSDL) is a XML based language that provides a model for describing web services.

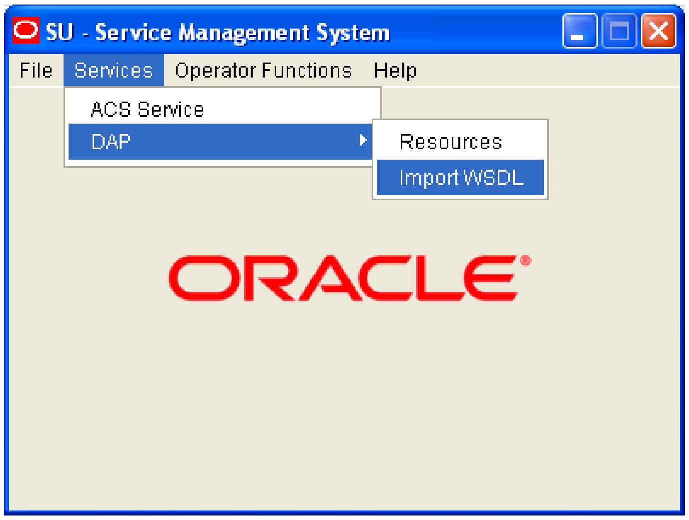
The reason for using WSDL is to import predefined web services from ASPs, thereby speeding up configuration of DAP messages.

For restrictions on what parts of the specification are supported, see *NCC Data Access Pack PICS Guide*.

Accessing the Import WSDL screen

Follow these steps to open the DAP Import WSDL screen.

Step	Action
1	Select the **Services** menu from the SMS main screen.
2	Select **DAP**.
3	Select **Import WSDL**.

Step	Action
	 SU - Service Management System File Services Operator Functions Help ACS Service DAP Resources Import WSDL ORACLE **Result:** You see the Import WSDL screen.

Import WSDL Files

Importing a WSDL file

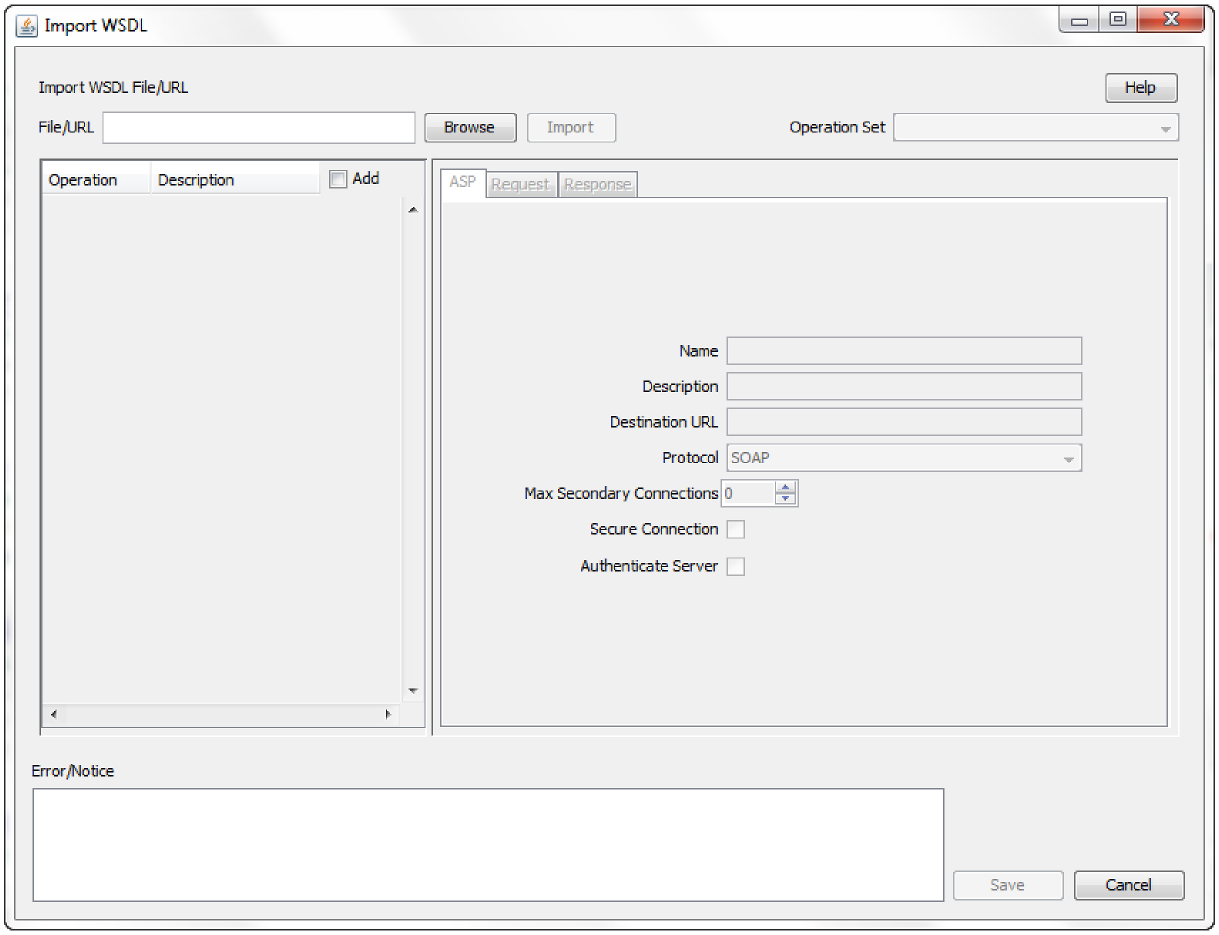
Follow these steps to import a WSDL file.

Step	Action
1	On the *Import WSDL screen* (See example on page 39), perform one of the following actions to select a file: • Enter the location and name of the **.wsdl** file in the **File/URL** field. The file location can be an URL. For information about specifing URLs, see *About Specifying URLs* (on page 22). • Click **Browse** to find the file The file has the suffix of **.wsdl** (for example **DAP.wsdl**) **Result:** The **Import** button becomes available.
2	Click **Import**. **Result:** The WSDL file is imported, then: • The **Operation/Description** panel is populated from the imported file • The *No operation has been added* message is displayed in the Notice/Error box.
3	To add: • *All* operations, select the **Add** check box • *Selected* operations, select the check box for each of the required operations **Result:** • Any configuration requirements are displayed in the Notice/Error box. • The configuration fields under the **ASP**, **Request**, and **Response** tabs are populated with data from the WSDL and become available for editing. • The **Operation Set** field becomes available. **Warning:** If you select multiple operations, you must configure all of them before the **Save** button is available. If there are any kind of fatal system problems, you will need to re-do

Step	Action
	any unsaved configuration.
4	Select the **Operation Set** to use from the drop down list.
	Note: See *Operation Sets* (on page 34) for details on configuring this list.
5	Click on the operation to configure.
6	Configure the ASP as required. See *ASP* (on page 22) for more information.
7	Configure the request as required. See *Operation Request Configuration* (on page 40) for more information.
8	Configure the response as required. See *Operation Response Configuration* (on page 42) for more information.
9	Repeat steps 4 to 8 for each operation that you want to configure.
10	When there are no errors listed in the **Error/Notice** area, the **Save** button becomes available. Click **Save**. The ASP and the configured operations are created. You can view and edit them on the DAP *Resources screen* (on page 21).

Import WSDL screen

Here is an example Import WSDL screen.



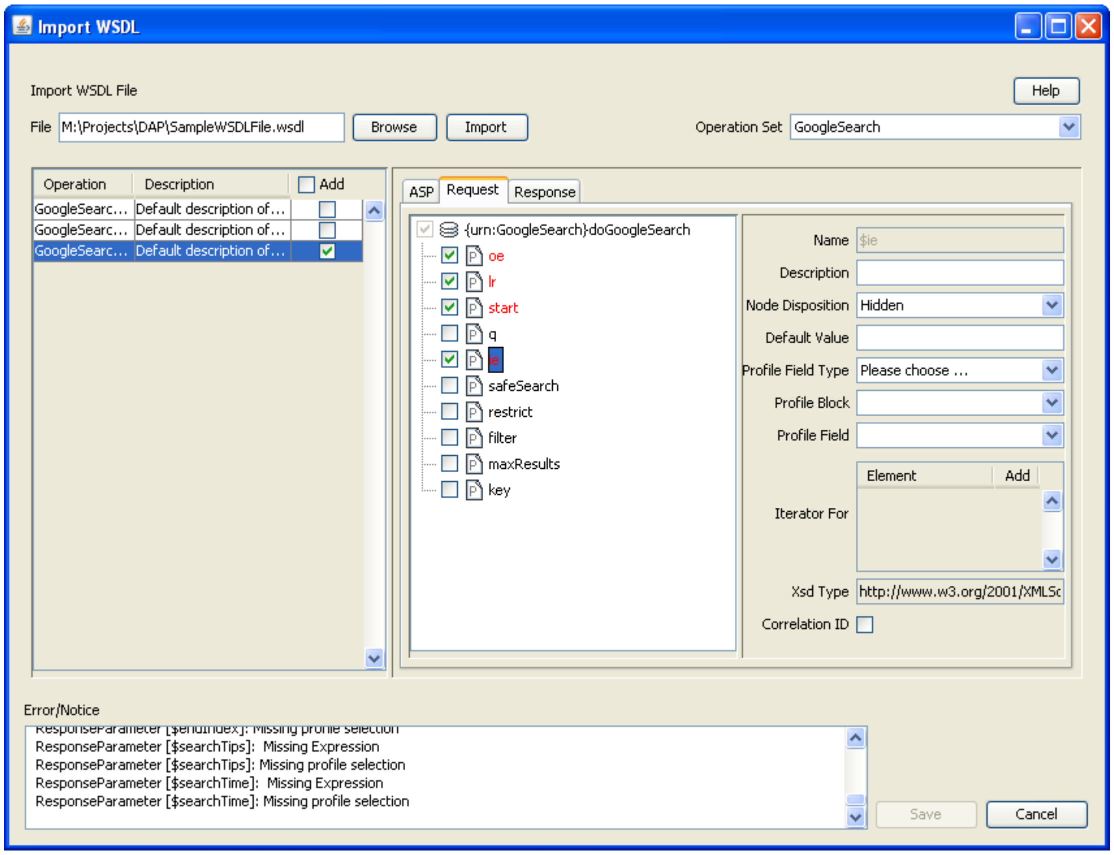
Operation Request Configuration

Introduction

You use the **Request** tab on the Import WSDL screen to configure the request parameters that are within the imported operation script. The imported operations are used by the DAP Request feature node.

Request tab

Here is an example of the **Request** tab on the Import WSDL screen after a WSDL file has been imported.



Import WSDL

Import WSDL File

File: M:\Projects\DAP\SampleWSDLFile.wsdl

Operation Set: GoogleSearch

Operation	Description	Add
GoogleSearc...	Default description of...	☐
GoogleSearc...	Default description of...	☐
GoogleSearc...	Default description of...	✓

{urn:GoogleSearch}doGoogleSearch
- ✓ oe
- ✓ lr
- ✓ start
- ☐ q
- ✓ ie
- ☐ safeSearch
- ☐ restrict
- ☐ filter
- ☐ maxResults
- ☐ key

Name	$ie
Description	
Node Disposition	Hidden
Default Value	
Profile Field Type	Please choose ...
Profile Block	
Profile Field	
Iterator For	Element / Add
Xsd Type	http://www.w3.org/2001/XMLSc
Correlation ID	☐

Error/Notice

```
ResponseParameter [$endIndex]: Missing profile selection
ResponseParameter [$searchTips]:  Missing Expression
ResponseParameter [$searchTips]: Missing profile selection
ResponseParameter [$searchTime]:  Missing Expression
ResponseParameter [$searchTime]: Missing profile selection
```

Configuring requests

Follow these steps to configure operation request parameters.

Step	Action
1	In the **Request** *tab* (See example on page 40), expand the parameter tree to see all the elements that can be configured (click **+** signs).

Note: Each element is preceded by a C, a P, an O, or an E:

- C – For a complex parameter that has more than one sub parameters.
- P – For the last element down a branch. This is the parameter that can be configured.
- O – For an optional parameter that may require configuration.
- E – For an empty sub-parameter within a complex parameter. This is not configurable.

Step	Action
2	Select the parameters to configure by selecting the check box preceding the parameter. All the selected parameter names are colored red to indicate that the parameters require configuration, and the error reasons appear in the **Error/Notice** area. Sometimes all parameters under a selected complex parameter are automatically selected because they are all required. **Tip:** Hover over a parameter to get a tip if a box cannot be selected or deselected.
3	Click a parameter name to configure. The parameter name appears in the **Name** field, the editable parameter fields are made available, and the **Iterator For** table is populated with all the request elements.
4	If this parameter is to be used as a unique id, select the **Correlation ID** check box and then start configuring a new parameter.
5	Type a description for the parameter in the **Description** field.
6	Select the feature node behavior for the parameter from the **Node Disposition** list. **Note:** The option you select determines whether or not you are required to configure a run-time parameter in a DAP feature node that sends XML requests. Select one of the following options: • `Hidden` – If the parameter will not be visible in the feature node configuration window and the parameter must be configured in this screen. • `Text` – If the parameter will be completed in the feature node configuration window as a text field. • `Profile Block –` If the parameter will be completed in the feature node configuration as a profile block location. • `Either` – If the parameter will be completed in the feature node configuration either as a text field or as a profile block location. • `Transient` – If you do not want to provide a run-time parameter and you also want to send the text defined in the XML template without any further processing. **Note:** The Transient option enables the DAP to process XML requests containing attributes where the parameters are not recognized.
7	If you selected the `Hidden` disposition, perform one, or both, of the following steps: a. Type the default value for the parameter in the **Default Value** field. b. Select the location of the parameter from the **Profile Field Type**, **Profile Block,** and **Profile Field** lists. **Tip:** The **Xsd Type** field indicates what type of data is expected in the DAP message for this parameter. **Note:** If an expected profile field is missing, it can be added via the ACS Configuration screens. For information about configuring profile fields, see *Advanced Control Services User's Guide*.
8	If this parameter is being used as an iteration value for a request element, scroll to find the element in the **Iterator For** list and select the **Add** check box. The parameter will be added to the request template element as an iteration. **Tip:** A parameter can be an iterator for more than one element, however one element registers just one parameter as the iterator.
9	Repeat steps **3** to **8** for all the parameters that require configuration.

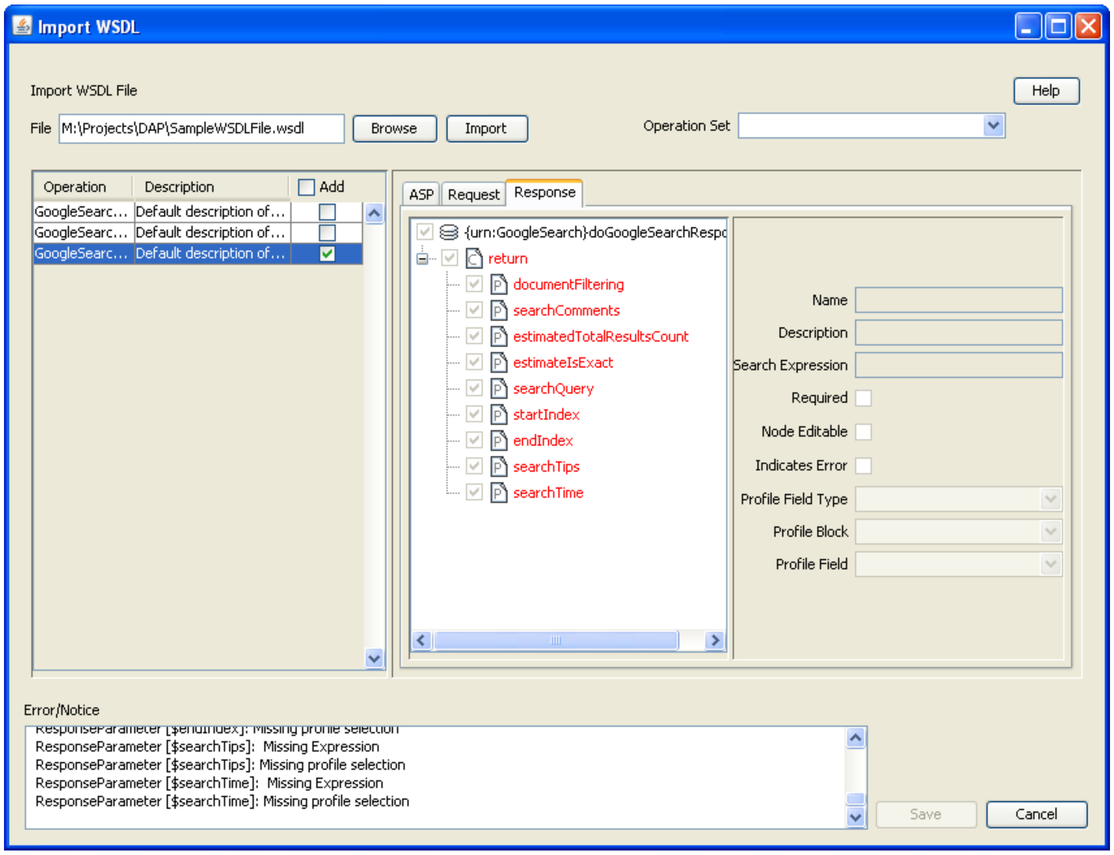
Operation Response Configuration

Introduction

You need to configure any responses required by the imported script. You perform this on the **Response** tab.

Response tab

Here is an example of the **Response** tab on the Import WSDL screen after a WSDL file has been imported.



File: M:\Projects\DAP\SampleWSDLFile.wsdl

Operation	Description	Add
GoogleSearc…	Default description of…	☐
GoogleSearc…	Default description of…	☐
GoogleSearc…	Default description of…	✓

```
{urn:GoogleSearch}doGoogleSearchRespo
  return
    documentFiltering
    searchComments
    estimatedTotalResultsCount
    estimateIsExact
    searchQuery
    startIndex
    endIndex
    searchTips
    searchTime
```

Name, Description, Search Expression, Required, Node Editable, Indicates Error, Profile Field Type, Profile Block, Profile Field

Error/Notice

```
ResponseParameter [$endIndex]: Missing profile selection
ResponseParameter [$searchTips]:  Missing Expression
ResponseParameter [$searchTips]: Missing profile selection
ResponseParameter [$searchTime]:  Missing Expression
ResponseParameter [$searchTime]: Missing profile selection
```

Configuring responses

Follow these steps to configure all response parameters.

Step	Action
1	In the **Response** *tab* (See example on page 42), expand the parameter tree to see all the elements that can be configured (click + signs). **Note:** Each element is preceded by a C, a P, an O, or an E: • C – For a complex parameter that has more than one sub parameters. • P – For the last element down a branch. This is the parameter that can be configured. • O – For an optional parameter that may require configuration. • E – For an empty sub-parameter within a complex parameter. This is not

Step	Action
	configurable.
2	Select the parameters to configure by selecting the check box that precedes each parameter. All the selected parameter names are colored red to indicate that the parameters require configuration.
3	Click a parameter name to configure. The parameter name appears in the **Name** field, the editable parameter fields are made available
4	In the **Description** field, type what the response is expected to be.
5	If this parameter must be returned, select the **Required** check box.
6	If this parameter can be edited in feature nodes, select the **Node Editable** check box.
7	If the return of this parameter is an error condition, select the **Indicates Error** check box. **Note:** The Required and Indicates Error check boxes are mutually exclusive.
8	Select the profile type and location for the parameter from the **Profile Field Type**, **Profile Block** and **Profile Field** lists. **Note:** If an expected profile field is missing, it can be added via the ACS Configuration screens. For information about configuring profile fields, see *Advanced Control Services User's Guide*. **Result:** The last error message for this parameter will disappear from the Error/Notices panel.
9	Repeat steps 3 to 8 to configure all the response parameters.

Chapter 4

Configuration

Overview

Introduction

This chapter explains how to configure the Oracle Communications Network Charging and Control (NCC) application.

In this chapter

This chapter contains the following topics.

Configuration Overview

Introduction

This topic provides a high level overview of how the Oracle Communications Network Charging and Control Data Access Pack (DAP) component is configured.

Configuration components

DAP is configured by the following components:

Component	Locations	Description	Further Information
eserv.config	All SLC machines	The DAP is configured in the `DAP` section of the **eserv.config** file.	**eserv.config** *Configuration* (on page 46)
SLEE.cfg	All SLC machines	This configures how the SLEE runs and manages *dapIF* (on page 88). SLEE configuration must include the DAP.	*Startup* (on page 88) *SLEE Technical Guide*
acs.conf	All SLC machines	The ACS framework must be configured to accept DAP calls.	*ACS Technical Guide*
SMF database	SMS	Statistics, profile block and EFM alarms configuration for DAP. Configured automatically when lcaSms is installed.	*Statistics* (on page 17)
DAP screens	SMS	The Resources screen configures ASP and DAP template records in SMF	*Resources* (on page 21)
ACS control	SMS	Specific control plans must be developed	*CPE User's*

plans		to handle DAP requests successfully.	*Guide*

eserv.config Configuration

Introduction

The **eserv.config** file is a shared configuration file, from which many Oracle Communications Network Charging and Control (NCC) applications read their configuration. Each NCC machine (SMS, SLC, and VWS) has its own version of this configuration file, containing configuration relevant to that machine. The **eserv.config** file contains different sections; each application reads the sections of the file that contains data relevant to it.

The **eserv.config** file is located in the **/IN/service_packages/** directory.

The **eserv.config** file format uses hierarchical groupings, and most applications make use of this to divide the options into logical groupings.

Configuration File Format

To organize the configuration data within the **eserv.config** file, some sections are nested within other sections. Configuration details are opened and closed using either { } or [ ].

- Groups of parameters are enclosed with curly brackets – { }
- An array of parameters is enclosed in square brackets – [ ]
- Comments are prefaced with a # at the beginning of the line

To list things within a group or an array, elements must be separated by at least one comma or at least one line break. Any of the following formats can be used, as in this example:

```
{ name="route6", id = 3, prefixes = [ "00000148", "0000473"] }
{ name="route7", id = 4, prefixes = [ "000001049" ] }
```

or

```
{ name="route6"
    id = 3
    prefixes = [
        "00000148"
        "0000473"
    ]
}
{ name="route7"
    id = 4
    prefixes = [
        "000001049"
    ]
}
```

or

```
{ name="route6"
    id = 3
    prefixes = [ "00000148", "0000473" ]
}
{ name="route7", id = 4
    prefixes = [ "000001049" ]
}
```

eserv.config Files Delivered

Most applications come with an example **eserv.config** configuration in a file called **eserv.config.example** in the root of the application directory, for example, **/IN/service_packages/eserv.config.example**.

Editing the File

Open the configuration file on your system using a standard text editor. Do not use text editors, such as Microsoft Word, that attach control characters. These can be, for example, Microsoft DOS or Windows line termination characters (for example, ^M), which are not visible to the user, at the end of each row. This causes file errors when the application tries to read the configuration file.

Always keep a backup of your file before making any changes to it. This ensures you have a working copy to which you can return.

Loading eserv.config Changes

If you change the configuration file, you must restart the appropriate parts of the service to enable the new options to take effect.

DAP eserv.config configuration

Introduction

The DAP **eserv.config** file example is installed by dapSms, dapScp and dapExtras in **/IN/services_packages/DAP/etc**.

The **eserv.config** file and `DAP` section is required on all nodes running a DAP client capable of sending DAP requests.

Example eserv.config DAP Section

Here is an example of the `DAP` section of the **eserv.config**.

```
DAP = {

    Mapping = [
        # XML protocol
        {
            Protocol = "H"
            InterfaceHandle = "dapIF"
        }
        # SOAP protocol
        {
            Protocol = "S"
            InterfaceHandle = "dapIF"
        }
        # HPSA protocol
        {
            Protocol = "A"
            InterfaceHandle = "dapIF"
        }
        # PIXML protocol
        {
            Protocol = "P"
            InterfaceHandle = "dapIF"
        }
        # LDAP protocol
        {
            Protocol = "L"
            InterfaceHandle = "ldapIF"
        }
    ]
```

```
templates = [
    {
        id = 83
        prefix = "xmlData="
        soapAction="PaymentService#OTPPayment"
        sendHeaderTag=false
    }
]

allowEmptyProfileValues = false
allowINSECURESSLv3 = false
allowLegacyServerConnect = false
allowBugWorkArounds = false

concatenate = false

listenHost = ""
listenPort = 4099

connectionTimeout = 0

disableTLS1_1 = false

correlationTagName = "CORRELATE"
uncorrelatedRequestDir = "/IN/service_packages/DAP/tmp/"
responseTagName = "CORRELATE"
hostnameInPost = true
timestampTagName = "TIMESTAMP"
DateTimeFormat = "YYYY-MM-DDThh:mm:ss"

sqlUseBusyHandler = true
sqlBusyWaitInteval = 20
sqlBusyRetryCount = 20

PollInterval = 500
PollCount = 100
PollServiceCounter = 2

cacheAgeSeconds = 60

maxRetries = 0
retryTimeout = 10

sessionTimeout = 86400

pendingFilename = "/IN/service_packages/DAP/tmp/pendingRequests.db"
pendingQueueInMemory = false

nonBlockingConnections = true
persistentConnections = false
persistentConnectionCheckTimeout = 0

maxQueueLength=500
maxQueueCheckTimeout=100

enableRetries=true
appendCRLFAfterBody=true

discardPendingQueueRequestsAfterSeconds = 60

timedConnectTimeout = 1

certificatePath = "/IN/service_packages/DAP/certificates/"
certificatesName = "CAfile.pem"
```

```
    cipherList = "ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-
    AES128-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES128-CBC-SHA256:ECDHE-RSA-
    AES256-CBC-SHA384:DHE-RSA-AES128-CBC-SHA256:DHE-RSA-AES256-CBC-SHA256:ECDH-RSA-
    AES128-GCM-SHA256:ECDH-RSA-AES256-GCM-SHA384:RSA-AES128-GCM-SHA256:RSA-AES256-
    GCM-SHA384:DH-RSA-AES128-GCM-SHA256:DH-RSA-AES256-GCM-SHA384:ECDH-RSA-AES128-CBC-
    SHA256:ECDH-RSA-AES256-CBC-SHA384:RSA-AES128-CBC-SHA256:RSA-AES256-CBC-SHA256:DH-
    RSA-AES128-CBC-SHA256:DH-RSA-AES256-CBC-SHA256"

    clientCertificateFile="client.pem"

    openSSLPath = "/usr/sfw/bin"
    sendRequestDateFormat = "%Y-%m-%d"
    sendRequestDateTZ = "US/Eastern"

    prefixTagName = "CUSTOM_PREFIX"
    suffixTagName = "CUSTOM_SUFFIX"

    connectionFailureRetryTime = 10

    soapHeaderOverride = "<?xml version=\"1.0\" encoding=\"ISO-8859-1\"
    standalone=\"no\" ?><soapenv:Envelope
    xmlns:soapenv=\"http://schems.xmlsoap.org/soap/envelope/\"
    xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\"
    xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"
    <soapenv:Header/><soapenv:Body>"
    useTemplateSOAPTags = false

    tracing = {
        enabled = true
        templates = [
            {
            id = 83
            msisdnParam = "$msisdn"
            msisdns = [
                "1234565"
                ]
            }
        ]
    }

}
```

DAP parameters

DAP accepts the following **eserv.config** parameters.

allowEmptyProfileValues

Syntax:	`allowEmptyProfileValues = true|false`
Description:	Whether or not the Send Request feature node should treat an empty or missing profile value as a failure. By default the Send Request feature node treats empty or missing profile values as a failure. Set to true to allow empty or missing values.
Type:	Boolean
Optionality:	Optional (default used if not set)
Allowed:	true or false
Default:	false
Notes:	
Example:	`allowEmptyProfileValues = true`

`allowINSECURESSLv3`

Syntax:	`allowINSECURESSLv3 = true|false`
Description:	Whether to allow use of SSLv3 in the SSL handshake for SSL enabled systems. For example, set this parameter to true for customers with an ASP that must use the SSLv3 protocol version. Use of SSLv3 and SSLv2 is disabled by default.
Type:	Boolean
Optionality:	Optional (default used if not set)
Allowed:	• true – Use of SSLv3 protocol version enabled. • false – Use of SSLv3 protocol version disabled.
Default:	false
Notes:	The `allowINSECURESSLv3` parameter can be set for the DAP, PI and OSD components. You should set `allowINSECURESSLv3` to true if the ASP is able to use only SSLv3 protocol version. Otherwise set `allowINSECURESSLv3` to false.
Example:	`allowINSECURESSLv3 = true`

`allowBugWorkArounds`

Syntax:	`allowBugWorkArounds = `*`true`*`	`*`false`*
Description:	Whether or not dapIF supports bug workarounds to cope with faulty SSL implementations on the ASP.
Type:	Boolean
Optionality:	Optional (default used if not set)
Allowed:	true – Bug workarounds are supported
	false – Bug workarounds are not supported
Default:	false
Notes:	Set this parameter to true only if it is required for dapIF to make successful SSL connections to an ASP.
Example:	`allowBugWorkArounds = true`

`allowLegacyServerConnect`

Syntax:	`allowLegacyServerConnect = `*`true`*`	`*`false`*
Description:	Whether or not dapIF allows connections to legacy servers that do not support secure renegotiation.
Type:	Boolean
Optionality:	Optional (default used if not set)
Allowed:	true – Allows connections to legacy servers that do not support secure renegotiation.
	false – Prohibits connections to legacy servers that do not support secure renegotiation.
Default:	false
Notes:	Set this parameter to true only if it is required for dapIF to make successful SSL connections to an ASP.
Example:	`allowLegacyServerConnect = true`

`appendCRLFAfterBody`

Syntax:	`appendCRLFAfterBody = `*`true|false`*
Description:	Set to true to append "\r\n\" (carriage return, line feed) after sending the message body.
Type:	Boolean
Optionality:	Optional (default used if not set).

Allowed:	true or false
Default:	true
Example:	`appendCRLFAfterBody = true`

cacheAgeSeconds

Syntax:	`cacheAgeSeconds =` *`seconds`*
Description:	The number of seconds before the template cache expires.
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	
Default:	60
Notes:	
Example:	`cacheAgeSeconds = 60`

CARevocationListChecking

Syntax:	`CARevocationListChecking =` *`true`* | *`false`*
Description:	Controls whether DAP checks ASP certificates against Certificate Revocation Lists (CRLs). For more information, see *Certificate Checking* (on page 13).
Type:	Boolean
Optionality:	Optional (default used if not set)
Allowed:	• true – DAP checks ASP certificates against CRLs. • false – DAP does not verify ASP certificates against CRLs.
Default:	false
Notes:	You must perform additional steps to configure DAP to verify ASP certificates against CRLs. For more information, see *Verifying ASP Certificates* (on page 13).
Example:	`CARevocationListChecking = true`

certificatePath

Syntax:	`certificatePath = "`*`dir`*`"`
Description:	The location of: • Server public certificates • Concatenated certificates file produced by *dapReadyCertificates.sh* (on page 95)
Type:	String
Optionality:	Optional (default used if not set)
Allowed:	
Default:	**/IN/service_packages/DAP/certificates**
Notes:	The name of the file containing the concatenated certificates is defined by *certificatesName* (on page 52). Any file in the `certificatePath` directory that does not have a filename matching `certificatesName` will be concatenated into the `certificatesName` file. For an overview of how certificates are handled, see *Certificate Management* (on page 13).
Example:	`certificatePath = "`**/IN/service_packages/DAP/certificates**`"`

certificatesName

Syntax: `certificatesName = "`*file*`"`

Description: The name of the file containing the concatenated servers' public certificates produced by **dapReadyCertificates.sh** (on page 95).

Type: String

Optionality: Optional (default used if not set)

Allowed:

Default: **CAfile.pem**

Notes: The directory location for the **dapReadyCertificates.sh** file is configured in the `certificatePath` (on page 51) parameter.

For an overview of how certificates are handled, see *Certificate Management* (on page 13).

Example: `certificatesName = "`**CAfile.pem**`"`

cipherList

Syntax: `cipherList = "`*string*`"`

Description: Specifies the ciphers allowed to be used for SSL. It is a list of one or more cipher names separated by colons.

Type: String

Optionality: Optional (default used if not set)

Allowed: May always be specified

Default: Default cipher list used is (Oracle product security recommended):

```
"ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-
AES128-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES128-CBC-
SHA256:ECDHE-RSA-AES256-CBC-SHA384:DHE-RSA-AES128-CBC-SHA256:DHE-RSA-
AES256-CBC-SHA256:ECDH-RSA-AES128-GCM-SHA256:ECDH-RSA-AES256-GCM-
SHA384:RSA-AES128-GCM-SHA256:RSA-AES256-GCM-SHA384:DH-RSA-AES128-GCM-
SHA256:DH-RSA-AES256-GCM-SHA384:ECDH-RSA-AES128-CBC-SHA256:ECDH-RSA-
AES256-CBC-SHA384:RSA-AES128-CBC-SHA256:RSA-AES256-CBC-SHA256:DH-RSA-
AES128-CBC-SHA256:DH-RSA-AES256-CBC-SHA256"
```

Notes: The cipher that is actually used is the result of the SSL handshake negotiation with the ASP. For the handshake to be successful, both DAP and the ASP must have a common cipher in the list DAP offers (set by cipherList) and the list the ASP supports (controlled by the ASP configuration). You can obtain the list of available ciphers for an OpenSSL installation by running the **`openssl ciphers`** command.

Oracle recommends choosing the most secure cipher that the ASP supports.

Example:

```
cipherList = "ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384"
```

clientCertificateFile

Syntax: `clientCertificateFile = "`*client_file*`"`

Description: The name of the file that contains the ASP-provided X.509 client certificate on the SLC. The DAP uses the specified file for SSL client authentication of dapIF on the ASP. If `clientCertificateFile` is not defined in **eserv.config**, then client certificates are not used for SSL client authentication. You configure the directory location of the certificate file in the `certificatePath` (on page 51) parameter.

Type: String

Optionality:	Optional
Allowed:	A valid SSL client certificate filename.
Default:	Not set
Notes:	Use the `certificatesName` (on page 52) parameter when configuring the filename for non-SSL client certificates. For an overview of how certificates are handled, see *Certificate Management* (on page 13).
Example:	`clientCertificateFile = "client.pem"`

`concatenate`

Syntax:	`concatenate =` *`true`*`|`*`false`*
Description:	Whether to concatenate the body of the XML requests.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	true, false
Default:	false
Notes:	
Example:	`concatenate = false`

`connectionFailureRetryTime`

Syntax:	`connectionFailureRetryTime =` *`seconds`*
Description:	How long to wait in seconds between connection attempts after a failed connection attempt.
Type:	Integer
Optionality:	Optional (default used if missing)
Allowed:	
Default:	10
Notes:	
Example:	`connectionFailureRetryTime = 20`

`connectionTimeout`

Syntax:	`connectionTimeout =` *`mseconds`*
Description:	The number of milliseconds before a connection to an ASP times out.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	0 (wait indefinitely)
Notes:	Can be overridden by the timeout for correlation objects.
Example:	`connectionTimeout = 0`

`correlationTagName`

Syntax:	`correlationTagName = "`*`name`*`"`
Description:	The correlation tag in the XML messages.
Type:	String
Optionality:	Optional (default used if not set)

Allowed:	
Default:	"CORRELATE"
Notes:	For more information about correlation, see *Correlation* (on page 27).
Example:	`correlationTagName = "CORRELATE"`

`DateTimeFormat`

Syntax:	`DateTimeFormat = "`*`dateformat`*`"`
Description:	Indicates the format for date variables sent in a DAP notification where the parameter value has been read from a DATE type profile field.
Type:	String
Optionality:	Optional
Allowed:	"YYYY-MM-DDThh:mm:ss" "YYYY-MM-DDThh:mm:ssZ" "-YYYY-MM-DDThh:mm:ss" "YYYYMMDDThhmmss"
Default:	YYYYMMDDThhmmss
Notes:	
Example:	`DateTimeFormat = "YYYYMMDDThhmmss"`

`disableTLS1_1`

Syntax:	`disableTLS1_1 = true|false`
Description:	Sets whether or not TLS (Transport Layer Security) 1.1 is enabled or disabled in the SSL context options when dapIF connects. Disable TLS 1.1 for connections to ASPs that do not support TLS 1.1 or that do not fully support SSL secure renegotiation.
Type:	Boolean
Optionality:	Optional (default used if not set)
Allowed:	false (TLS 1.1 is enabled) true (TLS 1.1 is disabled)
Default:	false
Notes:	Setting `disableTLS1_1` to true also disables use of TLS 1.2. Because of the way OpenSSL works, if there is an available lower version, such as TLS 1.0 (which is always enabled), disabling a higher version also disables the version above that one. Because of this behavior, do not set `disableTLS1_1` to true if TLS 1.2 is supported by the ASP and can be negotiated.
Example:	`disableTLS1_1 = true`

`discardPendingQueueRequestsAfterSeconds`

Syntax:	`discardPendingQueueRequestsAfterSeconds = `*`value`*
Description:	The number of seconds which the DAP request will be retried for before it is discarded. Using this parameter, DAP is configured to discard requests after a certain period of time if the destination ASP cannot be reached.
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	
Default:	-1 (no global discarding of old DAP requests)
Notes:	**Discard if Not Sent After (sec)** field value if set will always take priority over `discardPendingQueueRequestsAfterSeconds` parameter.

Example:	`discardPendingQueueRequestsAfterSeconds = 60`

`enableRetries`

Syntax:	`enableRetries =` *`true`*`|`*`false`*
Description:	Sets if DAP will attempt to resend failed requests
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	true, false
Default:	true
Notes:	
Example:	`enableRetries = true`

`hostnameInPost`

Syntax:	`hostnameInPost =` *`true`*`|`*`false`*
Description:	Include the full or shortened version of destination in the POST command in requests.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	true, false
Default:	true
Notes:	
Example:	`hostnameInPost = true`

`listenHost`

Syntax:	`listenHost = "`*`host`*`"`
Description:	Specifies the host name to put in the outgoing clientUrl HTTP header.
Type:	String
Optionality:	`listenHost` and `listenPort` are both required if the ASP returns parameters in a separate request.
Allowed:	
Default:	"" (none)
Notes:	
Example:	`listenHost = ""`

`listenPort`

Syntax:	`listenPort =` *`port`*
Description:	Specifies the port to put in the outgoing clientUrl HTTP header.
Type:	Integer
Optionality:	`listenHost` and `listenPort` are both required if the ASP returns parameters in a separate request.
Allowed:	
Default:	4099
Notes:	This allows for the situation when the ASP initiates requests to the XML Interface. For more information about message flows, see *Message flow* (on page 3).
Example:	`listenPort = 4099`

`Mapping`

Mapping between protocols and SLEE interface handles. Refer to *Mapping parameters* (on page 64).

`maxQueueCheckTimeout`

Syntax:	`maxQueueCheckTimeout =` *`seconds`*
Description:	The number of seconds between each diagnostic check of the pending queue size for each ASP.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	A valid number
Default:	100
Notes:	A warning will be logged if an ASP queue size is larger than the value specified for maxQueueLength.
Example:	`maxQueueCheckTimeout = 100`

`maxQueueLength`

Syntax:	`maxQueueLength =` *`size`*
Description:	The maximum size for the message queue. When set to greater than zero, any new requests to an ASP will be rejected if the request queue size for the ASP exceeds the defined value.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	A valid number
Default:	500
Notes:	A warning will be logged if an ASP queue size is larger than `maxQueueLength`.
Example:	`maxQueueLength = 500`

`maxRetries`

Syntax:	`maxRetries = number`
Description:	If a COMMAND_ACK is not received following a request, maxRetries defines the number of subsequent requests to attempt before expiring the request.
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	
Default:	0
Notes:	This parameter applies only to HPSA.
Example:	`maxRetries = 0`

`nonBlockingConnections`

Syntax:	`nonBlockingConnections =` *`true`*`|`*`false`*
Description:	Specifies whether or not to use non-blocking sockets for connections to ASPs.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	true, false
Default:	true
Notes:	

Example:	`nonBlockingConnections = false`

openSSLPath

Syntax:	`openSSLPath = "`*`path`*`"`
Description:	The location of *openssl* (on page 92). openssl is used by **dapReadyCertificates.sh** to concatenate and rehash certificate files to create fast lookup tables for DAP certificates.
Type:	String
Optionality:	Optional (default used if not set)
Allowed:	
Default:	**`/usr/sfw/bin`**
Notes:	
Example:	`openSSLPath = "`**`/usr/sfw/bin`**`"`

pendingFilename

Syntax:	`pendingFilename = "`*`path/file`*`"`
Description:	Location in which to store pending requests in the event that redelivery is required.
Type:	String
Optionality:	Optional (default used if not set)
Allowed:	
Default:	**`/IN/service_packages/DAP/tmp/pendingRequests.db`**
Notes:	The pendingRequests file contains one line per message in the format: `<msg_id>::<asp_url>::<port>::<protocol>::<timeout>::<xml>::[return _parameter_name::]+`
Example:	`pendingFilename = "`**`/IN/service_packages/DAP/tmp/pendingRequests.db`**`"`

pendingQueueInMemory

Syntax:	`pendingQueueInMemory = `*`true`*`|`*`false`*
Description:	Sets whether or not to boost performance by holding the queue of pending DAP requests in non-persistent memory.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	true, false
Default:	false
Notes:	When set to true, then no persistent records are stored in the location defined in the `pendingFilename` parameter.
Example:	`pendingQueueInMemory = true`

persistentConnections

Syntax:	`persistentConnections = `*`true`*`|`*`false`*
Description:	Specifies whether the primary connection to an ASP will remain open, or persist, between requests.
Type:	Boolean
Optionality:	Optional (default used if not set).

Allowed:	true, false
Default:	false
Notes:	When set to true, secondary connections will persist.
Example:	`persistentConnections = true`

`persistentConnectionCheckTimeout`

Syntax:	`persistentConnectionCheckTimeout =` *`seconds`*
Description:	Specify whether to check persistent socket connection for closure before sending new request on it. Value is number of seconds connection has been idle before check is to be performed. Zero value means no checking.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	0
Notes:	
Example:	`persistentConnectionCheckTimeout = 0`

`PollCount`

Syntax:	`PollCount =` *`number`*
Description:	Defines the number of zero wait polls to perform during idle periods, after which the `PollInterval` (on page 58) timeout is re-applied.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	A valid number
Default:	5000
Notes:	Enables tight polling in periods of high traffic
Example:	`PollCount = 2000`

`PollInterval`

Syntax:	`PollInterval =` *`mseconds`*
Description:	Number of milliseconds that *dapIF* (on page 88) will sleep before processing SLEE events.
Type:	
Optionality:	Optional (default used if not set)
Allowed:	
Default:	500
Notes:	
Example:	`PollInterval = 500`

`pollServiceCounter`

Syntax:	`PollServiceCounter =` *number*
Description:	Defines when to process backlogged requests during idle periods. DAP waits the specified number of idle polls before checking for pending timeout requests.
Type:	Integer
Optionality:	Optional (default used if not set)

Allowed:	Positive integer
Default:	2
Notes:	
Example:	`PollServiceCounter = 2`

`prefixTagName`

Syntax:	`prefixTagName = "`*`name`*`"`
Description:	Identifies the tag encapsulating the custom prefix text.
Type:	String
Optionality:	Optional
Allowed:	
Default:	Not set
Notes:	The tag itself is discarded before sending.
Example:	`prefixTagName = "CUSTOM_PREFIX"`

`responseTagName`

Syntax:	`responseTagName = "`*`name`*`"`
Description:	The name of the expected correlation tag in the XML response.
Type:	String
Optionality:	Optional (default used if not set).
Allowed:	A valid correlation tag name.
Default:	*correlationTagName* (on page 53) value
Notes:	If not set, then the value specified for the `correlationTagName` parameter will be used.
Example:	`responseTagName = "CORRELATE"`

`retryTimeout`

Syntax:	`retryTimeout = `*`seconds`*
Description:	After a request is issued, this defines the time to wait for a COMMAND_ACK before it attempting a new request.
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	
Default:	10
Notes:	This parameter applies only to HPSA.
Example:	`retryTimeout = 10`

`sendRequestDateFormat`

Syntax:	`sendRequestDateFormat = "`*`dateformat`*`"`
Description:	Indicates the date format to use for the `System date` (formatted) option while setting variables.
Type:	Date
Optionality:	Optional (default used if not set).
Allowed:	

Default:	`"%Y-%m-%d"`
Notes:	
Example:	`sendRequestDateFormat = "%Y-%m-%d"`

sendRequestDateTZ

Syntax:	`sendRequestDateTZ = "`*`name`*`"`
Description:	Alternative timezone abbreviation when using the 'System date (formatted)' ("<fdt>") option. This calculates the <fdt> date from the specified timezone.
Type:	String
Optionality:	Optional (default used if not set).
Allowed:	
Default:	GMT, NCC default TZ for SLC; otherwise the default system TZ
Notes:	
Example:	`sendRequestDateTZ = "US/Eastern"`

sessionTimeout

Syntax:	`sessionTimeout = `*`seconds`*
Description:	The number of seconds to negotiate an open SSL session with a remote server before it is reset by the server
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	86400 (24 hours)
Notes:	
Example:	`sessionTimeout = 86400`

sqlBusyRetryCount

Syntax:	`sqlBusyRetryCount = `*`Num`*
Description:	The maximum number of 10 millisecond waits that the SQL busy handler attempts when a database query or update returns a BUSY status. To use this parameter, the `sqlUseBusyHandler` parameter must be set to **true**.
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	Integer from 1 through 50
Default:	20
Notes:	The database controls whether or not it calls the SQL busy handler.
Example:	`sqlBusyRetryCount = 20`

sqlBusyWaitInterval

Syntax:	`sqlBusyWaitInterval = Num`
Description:	When a database query or update returns a BUSY status, this parameter specifies the amount of time to wait (in milliseconds) before stopping the database query or update. There is a delay of 1 millisecond between each query or update, repeating up to the specified value in `sqlBusyWaitInterval`. This parameter is used when one of the following is true: • The SQL busy handler is in lock contention • The `sqlUseBusyHandler` parameter is set to **false**.
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	An integer from 0 through 100
Default:	20
Notes:	
Example:	`sqlBusyWaitInterval = 20`

sqlUseBusyHandler

Syntax:	`sqlUseBusyHandler = true|false`
Description:	Specifies whether to call the SQL busy handler or to wait a specified amount of time after a database query or update returns a BUSY status.
Type:	Boolean
Optionality:	Optional (default used if not set)
Allowed:	true – Calls the SQL busy handler when a database query or update returns a BUSY status. Use the `sqlBusyRetryCount` parameter to specify the maximum number of waits before stopping the query or update. false – Waits a specified amount of time when a database query or update returns a BUSY status. Use the `sqlBusyWaitInterval` parameter to specify the maximum amount of time to wait before stopping the query or update.
Default:	true
Notes:	
Example:	`sqlUseBusyHandler = true`

soapHeaderOverride

Syntax:

```
soapHeaderOverride = "<?xml version=\"version\"
encoding=\"encoding\" standalone=\"yes|no\"
?><soapenv:Envelope xmlns:soapenv=\"url\"
xmlns:xsd=\"url/XMLSchema\" xmlns:xsi=\"url/XMLSchema-
instance\" <soapenv:Header/><soapenv:Body>"
```

Description:	Override the standard XML and SOAP header tags.
Type:	String
Optionality:	Optional
Allowed:	
Default:	Not used
Notes:	Applies to SOAP only. Note this does not affect the close tags </soapenv:Body></soapenv:Envelope>, which will still be added by DAP.

Example:	`soapHeaderOverride = "<?xml version=\"1.0\" encoding=\"ISO-8859-1\" standalone=\"no\" ?><soapenv:Envelope xmlns:soapenv=\"http://schems.xmlsoap.org/soap/envelope/\" xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\" xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\" <soapenv:Header/><soapenv:Body>"`

suffixTagName

Syntax:	`suffixTagName = "`*`name`*`"`
Description:	Identifies the tag encapsulating the custom suffix text.
Type:	String
Optionality:	Optional
Allowed:	
Default:	Not set
Notes:	The tag itself is discarded before sending.
Example:	`suffixTagName = "CUSTOM_SUFFIX"`

templates

Syntax:	`templates = [`*`parameters`*`]`
Description:	Allows overriding of certain values on a per template basis. See *templates parameters* (on page 64).
Type:	Array
Optionality:	
Allowed:	
Default:	
Notes:	Not used by default.

Example:

```
templates = [
   {
      id = 83
      prefix = "xmlData="
      soapAction="PaymentService#OTPPayment"
      sendHeaderTag=false
   }
]
```

timedConnectTimeout

Syntax:	`timedConnectTimeout = `*`seconds`*
Description:	The number of seconds dapIF waits when connecting to an ASP. This connection timeout is for the Network layer connection (TCP/IP).
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	Integer
Default:	1
Notes:	
Example:	`timedConnectTimeout = 2`

`timestampTagName`

Syntax:	`timestampTagName = "`*`tag`*`"`
Description:	Expected time stamp tag in the XML message.
Type:	String
Optionality:	Optional (default used if not set)
Allowed:	
Default:	"TIMESTAMP"
Notes:	
Example:	`timestampTagName = "TIMESTAMP"`

`uncorrelatedRequestDir`

Syntax:	`uncorrelatedRequestDir = "`*`path`*`"`
Description:	The ASPManager process writes to an uncorrelated log file at this path if it receives a correlated response for which it cannot find a matching correlated entry.
Type:	String
Optionality:	Optional (default used if not set).
Allowed:	
Default:	
Notes:	
Example:	`uncorrelatedRequestDir = "/IN/service_packages/DAP/tmp/"`

`useTemplateSOAPTags`

Syntax:	`useTemplateSOAPTags = true|false`
Description:	Do not include any SOAP header tags in dapIF. Only use those from the template.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	true, false
Default:	false
Notes:	The close tags (</soapenv:Body></soapenv:Envelope>) will be added if not already present at the end of the template.
Example:	`useTemplateSOAPTags = false`

`useDefaultAddress`

Syntax:	`useDefaultAddress = `*`true|false`*
Description:	Specifies whether or not to populate **SOAPAction** field in message header with destination URL.
Type:	Boolean
Optionality:	Optional (default used if not set)
Allowed:	true - **SOAPAction** field will be populated with destination URL.
	false - **SOAPAction** field will be empty.
Default:	true
Notes:	
Example:	`useDefaultAddress = true`

Mapping parameters

The following parameters are valid for the `Mapping` section of the DAP configuration.

`InterfaceHandle`

Syntax:	`InterfaceHandle = "`*`Dap_Interface`*`"`
Description:	Specifies which SLEE Interface to connect to.
Type:	String
Optionality:	Optional (default used if not set).
Allowed:	dapIF — single SLEE handle for the DAP Interface ["dapIF_1","dapIF_2",...] — list of multiple SLEE handles
Default:	""
Notes:	This must be the SLEE Handle for the DAP Interface as defined in the SLEE config. Multiple SLEE handles can be defined in a list and requests will be load balanced between them.
Example:	`InterfaceHandle = "dapIF"`

`Protocol`

Syntax:	`Protocol = "H|S|A|P|L"`
Description:	The protocol to be used between the SLEE interface and the external ASP.
Type:	String
Optionality:	
Allowed:	H — XML S — SOAP A — HPSA P — PIXML L — LDAP
Default:	-
Notes:	For XML and SOAP, the ASP configuration defines whether it is over HTTP or HTTPS. For more information about this configuration, see *ASP tab fields* (on page 23). LDAP is only available if LDAP interface for DAP has been installed and configured. Refer to *LDAP Interface for DAP Technical Guide* for details.
Example:	`Protocol = "H"`

templates parameters

The following parameters are valid for the `templates` section of the DAP configuration.

`id`

Syntax:	`id = `*`id`*
Description:	ID of the template ID to act on.
Type:	Integer
Optionality:	Optional
Allowed:	
Default:	

Notes:	
Example:	`id = 83`

`prefix`

Syntax:	`prefix = "`*`pref`*`"`
Description:	The prefix to add in front of XML or SOAP
Type:	String
Optionality:	Optional
Allowed:	
Default:	not added
Notes:	
Example:	`prefix = "xmlData="`

`sendHeaderTag`

Syntax:	`sendHeaderTag=`*`true`*`|`*`false`*
Description:	Send the soapenv:Header tag
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	true, false
Default:	true
Notes:	Only applicable for SOAP
Example:	`sendHeaderTag=false`

`soapAction`

Syntax:	`soapAction="`*`action`*`"`
Description:	SOAP action to specify in the header
Type:	String
Optionality:	Optional
Allowed:	
Default:	If not specified uses destination URL
Notes:	Only applicable for SOAP
Example:	`soapAction="PaymentService#OTPPayment"`

Tracing Configuration for Checking DAP Requests

The tracing configuration in the DAP section of the **eserv.config** configuration file enables optional trace output to be recorded in the DapTracing section of the DAP log file. The trace checks DAP requests and responses sent by dapIF for a specified list of MSISDNs or prefixes.

Note: To enable writing trace output to the DAP log file, you must also enable debug in the DAP log file. See *Enabling DapTracing Debug* (on page 67) for details.

The tracing configuration has the following syntax:

```
tracing = {
    enabled = true|false
    templates = [
        {
```

```
        id = int
        msisdnParam = "$request_parameter"
        msisdns = [
            "msisdn_prefix"[,"msisdn_prefix"]
            ]
        }
    ]
}
```

The tracing configuration supports the following parameters:

`enabled`

Syntax:	`enabled = true|false`
Description:	Enables or disables trace output in the DAP log file for the specified request or response templates, and MSISDNs.
Type:	Boolean
Optionality:	Optional (default used if not set)
Allowed:	true or false
Default:	false
Notes:	The trace output is written to a special DapTracing debug section of the log file; therefore to enable trace output, DAP debug must also be enabled in the log file. See *Enabling DapTracing Debug* (on page 67) for details.
Example:	`enabled = true`

`templates`

Syntax:	`templates = [templates_parameters]`
Description:	List of DAP template IDs and MSISDNs to trace.
Type:	Array
Optionality:	Required
Notes:	A trace line is written to the DapTracing debug section in the DAP log file for each request or response that is processed for the specified templates.

Example:

```
templates = [
    {
    id = 83
    msisdnParam = "$msisdn"
    msisdns = ["123456"]
    }
]
```

`id`

Syntax:	`id = int`
Description:	The ID of the template to trace.
Type:	Integer
Optionality:	Required
Allowed:	A valid template ID
Example:	`id = 83`

`msisdnParam`

Syntax:	`msisdnParam = "$request_parameter"`
Description:	The name of the request parameter to trace. You must specify a request parameter for a MSISDN value or MSISDN prefix value.

Type:	String
Optionality:	Required
Allowed:	A valid request parameter
Default:	
Notes:	
Example:	`msisdnParam = "$msisdn"`

msisdns

Syntax:	`msisdns = ["`*`msisdn_prefix`*`"[,"`*`msisdn_prefix`*`"]]`
Description:	The list of MSISDNs or MSISDN prefixes to trace. To trace all MSISDNs, specify an empty list.
Type:	Array
Optionality:	Required
Example:	`msisdns = ["1234356"]`

Enabling DapTracing Debug

To enable writing DapTracing debug content to the DAP log file:

Step	Action

1	On the SLC, open the **/IN/service_packages/DAP/bin/dapIF.sh** file by using a text editor.
	The **dapIF.sh** file should contain the following lines:
	`# For a concise dapIF debug`
	`#DEBUG=dapInterface,dapInterface_extraDetails,ASPManager,ASPConnection,DAPSecureConnection,ConfigFileImpl,Config,DapFileWriter,`**`DapTracing`**
	
	`# For a verbose dapIF debug`
	`#DEBUG=dapInterface,dapInterface_extraDetails,ASPManager,ASPConnection,DAPSecureConnection,ConfigFileImpl,Config,DapFileWriter,DapReadWrite,DapPolling,DapTimeout,`**`DapTracing`**
	
	`export DEBUG`

2	If the **dapIF.sh** file: • Already includes DEBUG lines, then enable debug output by removing the # character from the beginning of one of the DEBUG lines, and make sure the DapTracing option is included in the DEBUG line. • Does not include a DEBUG line, then add a DEBUG line that includes the debug options you want to enable, and make sure the DapTracing option is included in the DEBUG line.
3	After the DEBUG line, add the following line if it does not exist already: `export DEBUG`
4	Save and close the **dapIF.sh** file.
5	Restart the SLEE on the SLC to load the updated **dapIF.sh** file.

About DAP Notifications

DAP Manager logs all or failed notifications for the predetermined time and predetermined size to the specified log directory for future processing.

The DAP notification configuration has the following syntax:

```
DAP = {
NotificationsLog = {
LogType = "String"
LogDirectory = "directory_path"
LogFileNamePrefix = "String"
MaxAgeSeconds = Seconds
MaxSizeEntries = Integer
}
}
```

The DAP notification configuration contains the following parameters:

`LogType`

Syntax: `LogType = "`*`String`*`"`

Description: Indicates the log type for the DAP notification dapIF to log failed or all notifications.

Type: String

Allowed: The following log types are allowed:

- IGNORE: No logging
- ERROR: Log failed notifications
- ALL: Log all notifications

Default: IGNORE

Example: `LogType = IGNORE`

`LogDirectory`

Syntax: `LogDirectory = "`*`directory_path`*`"`

Description: Indicates the name of the notifications log directory to which the notification log entries are written.

Type: String

Allowed: Any existing directory path

Default: /IN/service_packages/DAP/tmp/notification-logs

Example: `LogDirectory = "/IN/service_packages/DAP/tmp/notification-logs"`

`LogFileNamePrefix`

Syntax: `LogFileNamePRefix = "`*`String`*`"`

Description: Indicates the base name of the log files. The complete log file name will be appended with the start and stop times.

Example:dapNotifications_20150701081103-20150701081158

Type: String

Allowed: NA

Default: dapNotifications

Notes:

Example: `LogFileNamePrefix = "dapNotifications"`

`MaxAgeSeconds`

Syntax: `MaxAgeSeconds = `*`Seconds`*

Description: Indicates the seconds the log entries are cached before they are written to the log file.

Type: Integer

Allowed: > or = 1

Default: 60

Notes:	
Example:	`MaxAgeSeconds = 60`

`MaxSizeEntries`

Syntax:	`MaxSizeEntries = `*`Integer`*
Description:	Indicates maximum number of entries in the audit entry cache before the excess are written to the log file.
Type:	Integer
Allowed:	> or = 1
Default:	100
Notes:	
Example:	`MaxSizeEntries = 100`

SLEE.cfg Configuration

Introduction

The **SLEE.cfg** file must be configured to enable the DAP to work. Because all necessary SLEE configuration is done at installation time by the configuration script, this section is for information only.

The SLEE configuration file is located at **/IN/service_packages/SLEE/etc/SLEE.cfg**.

Refer to *SLEE Technical Guide* for details on SLEE configuration.

DAP SLEE configuration

During installation, the following line will be added to the **SLEE.cfg** file:

```
INTERFACE=dapIF dapIF.sh /IN/service_packages/DAP/bin EVENT
```

Usage:

```
INTERFACE=uniqueIdentifier interfaceName interfacePath interfaceType [eventCount
dialogCount]
```

Larger SLEE events

DAP requires that a pool of SLEE events of at least 3072 bytes is configured; for example:

```
MAXEVENTS=count 3072
```

Where *count* is the pool size.

For most efficient use of shared memory, a pool of SLEE events of 1024 bytes should also be configured, for example:

```
MAXEVENTS=count 1024
```

Configuration for Optimal Performance

Introduction

You can configure the DAP to optimize its performance. The optimal configuration settings will depend on the capabilities of the ASP.

HTTP version 1.1

If the ASP supports HTTP version 1.1 then the following configuration is required to optimize DAP performance. Set the `DAP` parameters:

- `nonBlockingConnections = false`
- `persistentConnections = true`

HTTP version 1.0

If the ASP supports HTTP version 1.0 then the following configuration is required to optimize DAP performance. Set the `DAP` parameter:

- `persistentConnections = false`

Multiple instances

You can increase the speed of traffic through the DAP by using multiple instances of the DAP interface. The speed will increase by the maximum speed of a single DAP interface multiplied by the number of instances.

Multiple instances of the DAP interface can be started by adding more interface definitions to the DAP SLEE configuration. See *SLEE.cfg Configuration* (on page 69) for details.

Note: If you add more interfaces to **SLEE.cfg**, then you must configure the list of interface handles in the `DAP` section of **eserv.config**. See *Mapping parameters* (on page 64) for details.

General

You can increase the volume of traffic through the DAP interface by increasing the maximum secondary connections allowed to the ASP. For details, see *ASP configuration* (on page 24).

DAP performance can also be improved by specifying the following configuration. Set the `DAP` parameters:

- `PollInterval = 10`
- `PollCount = 5000`
- `pendingQueueInMemory = true`

For more information on configuring DAP parameters refer to *DAP* **eserv.config** *configuration* (on page 47).

Warning: If `pendingQueueInMemory` is set to true then the pending queue will not be stored in persistent storage and therefore cannot be recovered following a failure and restart of the DAP process.

LDAP IF Configuration

This topic provides an overview of how the LDAP Interface for DAP is configured.

Configuration components

The LDAP interface for DAP is configured by the following mechanisms:

Component	Locations	Description	Further Information
eserv.config	All SLC machines	The LDAP IF is configured in the 'DAP' section of the eserv.config file.	eserv.config Configuration. See *Global Configuration* (on page 72).

SLEE.cfg	All SLC machines	The SLEE configuration is altered by the ldapScp package install to start the ldapIF. Default configuration should be checked but should generally not need to be modified.	*SLEE Technical Guide*
SMF GUI (database)	USMS	Statistics and EFM alarm configuration is performed by the ldapSms package install. Default configuration should not generally need to be modified.	Installing ldapSms on a SMS.
DAP GUI (database)	USMS	LDAP Service Providers (LDAP servers) must be defined using the DAP GUI.	*Data Access Pack User's & Technical Guide*, plus LDAP Interface specific details in *DAP Resource Configuration* (on page 78).
ACS control plans	USMS	Specific control plans must be developed to perform LDAP requests through the DAP framework.	*ACS Technical Guide*, *Data Access Pack User's & Technical Guide*, plus LDAP Interface specific details in *DAP Control Plan* (on page 86)

eserv.config Configuration

Introduction

This topic provides a high level overview of how the LDAP interface for DAP component is configured.

Configuration File Format

To organize the configuration data within the **eserv.config** file, some sections are nested within other sections. Configuration details are opened and closed using either { } or [ ].

- Groups of parameters are enclosed with curly brackets – { }
- An array of parameters is enclosed in square brackets – [ ]
- Comments are prefaced with a # at the beginning of the line

To list things within a group or an array, elements must be separated by at least one comma or at least one line break. Any of the following formats can be used, as in this example:

```
{ name="route6", id = 3, prefixes = [ "00000148", "0000473"] }
{ name="route7", id = 4, prefixes = [ "000001049" ] }
```

or

```
{ name="route6"
    id = 3
    prefixes = [
        "00000148"
        "0000473"
    ]
}
{ name="route7"
    id = 4
```

```
        prefixes = [
            "000001049"
        ]
    }
```

or

```
    { name="route6"
        id = 3
        prefixes = [ "00000148", "0000473" ]
    }
    { name="route7", id = 4
        prefixes = [ "000001049" ]
    }
```

Editing the File

Open the configuration file on your system using a standard text editor. Do not use text editors, such as Microsoft Word, that attach control characters. These can be, for example, Microsoft DOS or Windows line termination characters (for example, ^M), which are not visible to the user, at the end of each row. This causes file errors when the application tries to read the configuration file.

Always keep a backup of your file before making any changes to it. This ensures you have a working copy to which you can return.

Loading eserv.config configuration changes

To cause LDAP interface for DAP to reread from its **eserv.config** configuration file, send a SIGHUP signal to the ldapIF process. This will reload all configuration, including all connection configuration.

Existing connections will be dropped if the configuration has changed.

eserv.config files delivered

The ldapIF comes with an example **eserv.config** configuration fragment installed into the following file:

```
/IN/service_packages/DAP/etc/ldap.example.eserv.config
```

This fragment is not automatically installed into the live **eserv.config**. The system operator is expected to copy the sample configuration manually and modify it to suit their site-specific requirements.

By default, ldapControlAgent (which runs LDAP on the SLC) will read its live run-time configuration from the DAP section of:

```
/IN/service_packages/eserv.config
```

The ESERV_CONFIG_FILE variable can override the default location.

Global Configuration

Introduction

All additions to the eserv.config file to support the LDAP Interface for DAP can be found in the 'DAP' section of the eserv.config file. The example eserv.config addition for the LDAP interface can be found in the SLC (/IN/service_packages/DAP/etc/ldap.example.eserv.config). There are two areas in which additional configuration can be made.

- A new mapping must be added to the DAP 'Mapping' section to ensure that DAP LDAP requests are correctly forwarded to the LDAP Interface. If this Mapping is not present, then the ldapIF will still start up, but the DAP macro node will be unable to send messages to the ldapIF.

- An 'LDAP' configuration section inside the top level 'DAP' section provides site-specific global configuration parameters to overwrite the default. If this section is not present then the ldapIF will start and will run using built-in default values.

Example eserv.config file section

Here is the example **eserv.config** DAP section.

```
DAP = {

    Mapping = [

        # LDAP protocol
        {
            Protocol = "L"
            InterfaceHandle = "ldapIF"
        }

    ]

    LDAP = {
        requestSchema = "/IN/service_packages/DAP/etc/ldap_request_schema.xsd"
        responseSchema = "/IN/service_packages/DAP/etc/ldap_response_schema.xsd"

        validateRequestXML = true

        disconnectWhenIdleTime = 600

        connectionTimeout = 30

        connectionRetryTime = 30

        recordStatisticsEvery = 60

        houseKeepingInterval = 30

        maxRequestAge = 30

        cacheTimeoutInterval = 60

        noWorkSleepTime = 20000
    }
}
```

DAP Mapping

The available protocols are:

- “H”,
- “S”,
- “A” and
- "P”

With the installation of the LDAP interface for DAP, mapping “L” can be configured and used.

LDAP parameters

Here are the parameters that can be used in the LDAP section.

cacheTimeoutInterval

Syntax:	`cacheTimeoutInterval = <secs>`
Description:	The number of seconds after which a cached database record expires.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	Integers that are greater than zero.
Default:	60
Notes:	The interface caches database information related to ASPs, operations and operation response parameters. An entry in the cache will be re-read if (when accessing the entry) the entry has not been read directly from the database for more than 'cacheTimeoutInterval' seconds. **Example:** If cacheTimeoutInterval is 30, attempting to read a cache entry that was last reloaded (read from the database) more than 30 seconds ago will cause the entry to be reloaded. Attempting to read an entry that was last reloaded 15 seconds ago will not cause the entry to be reloaded. Shorter values for this parameter will mean that operator changes made in the USMS DAP GUI screens are more quickly recognised by the LDAP Interface, and hence a short value is appropriate for a testing environment. However, setting this value higher may improve system performance in a stable, production system. **Note:** The cacheTimeoutInterval applies only to the ASP and Operation configuration stored in the database. It does not apply the configuration values in eserv.config, as those values are not cached, and are reloaded only on receipt of a SIGHUP.
Example:	`cacheTimeoutInterval = 300`

connectionRetryTime

Syntax:	`connectionRetryTime = <secs>`
Description:	The number of seconds that the interface will wait after a failed attempt to connect to an ASP, before reattempting to connect.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	Integers that are greater than zero.
Default:	30
Notes:	
Example:	`connectionRetryTime = 30`

connectionTimeout

Syntax:	`connectionTimeout = <secs>`
Description:	The number of seconds that the interface will wait for a BindResponse message from an ASP (when attempting to connect to that ASP) before assuming that the connection attempt failed.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	Integers that are greater than zero.
Default:	30
Notes:	The interface will reattempt to connect after waiting for the length of time specified by the 'connectionRetryTime' parameter.
Example:	`connectionTimeout = 30`

disconnectWhenIdleTime

Syntax:	`disconnectWhenIdleTime = <secs>`
Description:	The length of time (in seconds) that a connection with an ASP can remain idle before the connection will be closed.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	Integers that are greater than or equal to zero.
Default:	0
Notes:	If the value of this parameter is '0' the interface will not close idle connections.
Example:	`disconnectWhenIdleTime = 300`

houseKeepingInterval

Syntax:	`houseKeepingInterval = <secs>`
Description:	The number of seconds that the interface will wait between doing internal structure clean ups.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	Integers that are greater than zero.
Default:	30
Notes:	The default value of this parameter should not generally need to be changed.
Example:	`houseKeepingInterval = 300`

maxRequestAge

Syntax:	maxRequestAge = <secs>
Description:	The number of seconds, after sending, before a timed out request can safely be removed from internal structures.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	Integers that are greater than zero.

Default:	30
Notes:	Occasionally the interface needs to clean out its structures to avoid gradually increasing memory usage over time. This parameter defines when it is safe to clean up a request that has timed out. **Example:** If maxRequestAge is 20, any messages that have timed out and that were sent more than 20 seconds ago will be cleaned up by the interface's housekeeping sweep. Messages that have timed out and that were not sent more than 20 seconds ago (and messages that have not yet timed out) will **not** be cleaned up. When a message is timed out, there is a chance that the response is delayed, and may be received subsequently. If the late response is received after the interface has cleaned its reference, then an “unmatched ID” warning message will be generated. By configuring an appropriate maxRequestAge, these spurious warnings can be avoided.
Example:	`MaxRequestAge = 300`

noWorkSleepTime

Syntax:	`noWorkSleepTime = <microseconds>`
Description:	The number of microseconds to sleep when there are no downstream requests to process.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	Integers that are greater than 0.
Default:	20000 (0.2 sec)
Notes:	
Example:	`noWorkSleepTime = 30000`

recordStatisticsEvery

Syntax:	`recordStatisticsEvery = <secs>`
Description:	The number of seconds over which the interface should aggregate its non-peg-count statistics.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	Integers that are greater than zero.
Default:	60
Notes:	This applies *only* to the MIN, MAX and MEAN latency statistics, which are computed over time. All other statistics are peg-count statistics which are recorded as and when the events occur. The value of recordStatisticsEvery should be set to a period equal to or shorter than the collection period for these statistics as defined in the SMF Statistics Management GUI screen.
Example:	`recordStatisticsEvery = 300`

`requestSchema`

Syntax:	`requestSchema = "<location>"`
Description:	The location of the file containing the XSD schema used to validate requests before they are sent.
Type:	String
Optionality:	Optional (default used if not set).
Allowed:	Any fully-qualified file containing a valid XSD schema
Default:	/IN/service_packages/DAP/etc/ldap_request_schema.xsd
Notes:	Although this parameter makes it possible to relocate the schema, it is strongly recommended that you do not modify the schema content because the schema is actually used by the interface. Setting this parameter incorrectly can affect the behaviour of the 'validateRequestXML' parameter. Request validation will be unconditionally turned off if any of the following is true: • the parameter points to a file that does not exist • the parameter points to a file that the interface does not have permission to read the file pointed at by the parameter contains XSD syntax errors
Example:	`requestSchema = “/config/ldap_request_schema.xsd”`

`responseSchema`

Syntax:	`responseSchema = “<location>”`
Description:	The location of the file containing the XSD schema that describes the structure of the XML translation of LDAP search response entries.
Type:	String
Optionality:	Optional (default used if not set).
Allowed:	Any fully-qualified file containing a valid XSD schema.
Default:	/IN/service_packages/DAP/etc/ldap_response_schema.xsd
Notes:	This parameter makes it possible to relocate the schema. The schema is not used for validation, but the information it contains may be useful to an ACS service designer when developing DAP response parameter XPath queries for a DAP Request node in an ACS control plan.
Example:	`responseSchema = “/config/ldap_response_schema.xsd”`

`validateRequestXML`

Syntax:	validateRequestXML = <*false|true*>
Description:	A switch that allows request validation to be turned on or off.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	**True :** The interface will attempt to perform request validation. **False:** The interface will not perform request validation.
Default:	true

Notes:	When validation is on, each request (while in its XML form) will be validated against the schema specified by the 'requestSchema' parameter. Validation will not be performed even when the parameter is 'true' when the 'requestSchema' parameter is incorrectly specified. Request validation is strongly recommended during testing of new DAP LDAP operations and new ASP operations. In a stable, production system, disabling validation may result in improvements in system performance if LDAP is frequently used at high call rates.
Example:	`validateRequestXML = false`

SLEE.cfg

Introduction

The SLEE must be configured to start the LDAP IF. The following line will be added to the SLEE configuration file (/IN/service_packages/SLEE/SLEE.cfg) automatically during installation:

```
 INTERFACE=ldapIF ldapIF.sh /IN/service_packages/DAP/bin EVENT
```

This default configuration should be appropriate for most environments. Refer to the *SLEE Technical Guide* for further details.

DAP Resource Configuration

Introduction

An LDAP request is sent only when a DAP Request node is used in an ACS control plan. Correctly configuring such a node requires DAP resources to be available first, specifically, the following must be defined:

- A DAP ASP server. See *Defining a DAP ASP for LDAP* (on page 79).
- A DAP operation definition. See *Defining a DAP Operation for LDAP* (on page 80).

Chapter 5

Using LDAP with DAP

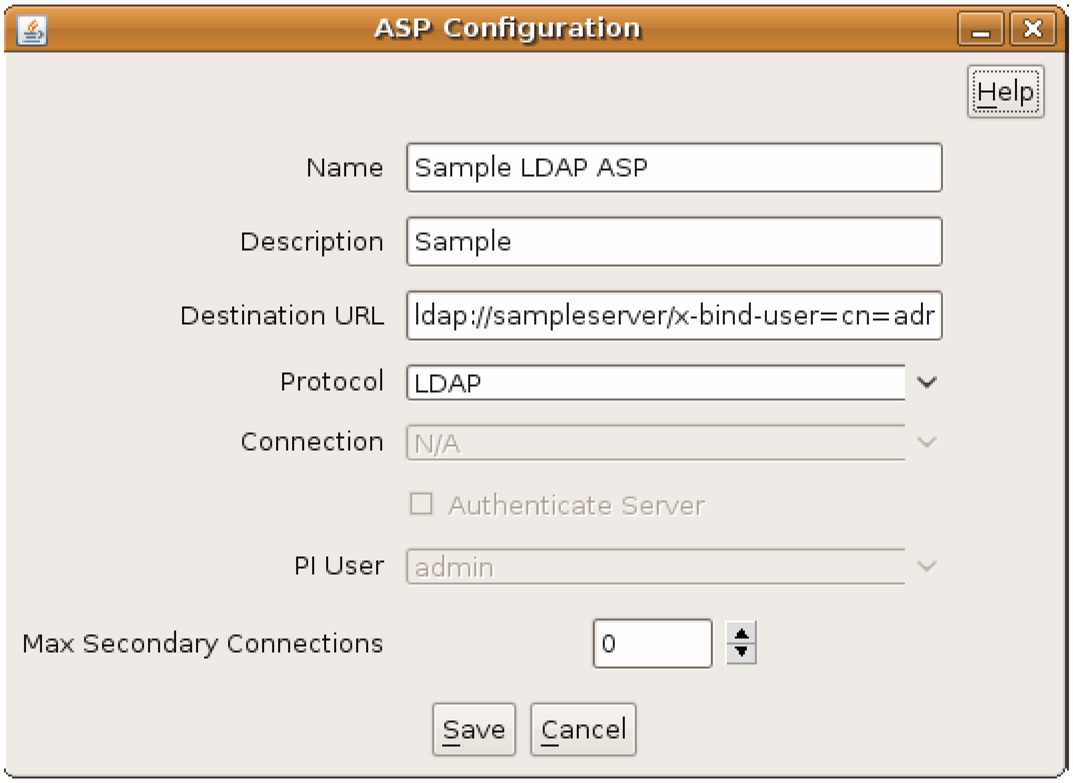
Defining a DAP ASP for LDAP

Introduction

This topic provides additional configuration information related to the definition of a DAP ASP for LDAP.

ASP Configuration screen

This screen shot shows a typical dialog window to Create or Edit a DAP ASP.



Name: Sample LDAP ASP
Description: Sample
Destination URL: ldap://sampleserver/x-bind-user=cn=adr
Protocol: LDAP
Connection: N/A
☐ Authenticate Server
PI User: admin
Max Secondary Connections: 0

ASP configuration fields

These comments apply specifically to defining an LDAP protocol ASP.

Field	Notes
Name	A unique name for this ASP.
Description	An optional description field for this ASP.
Destination URL	This field defines the “LDAP URL” which specifies the location of the server against which LDAP requests should be made. The URL full syntax is specified in RFC2255. The LDAP server for DAP supports a subset of the full field list. The supported syntax is: `ldap://<server-name>:<port>/x-auth-`

	`method=simple,x-encryption=none,x-version=<`*`LDAP protocol version`*`>,x-bind-user=<`*`user-name`*`>,x-bind-password=<`*`password`*`>`
	Where: • <*server-name*> = the host name or IP address of the LDAP server • <*port*> = the port number that the LDAP server is running on (optional, default: 389) • <*user-name*> = the name of the user account to connect with • <*password*> = the user account password (optional) • <*LDAP protocol version*> = the LDAP protocol version that the LDAP server supports. Supported versions are 2 and 3. • The x-auth-method, x-encryption and x-version parameters are all optional, and if specified, should be set only to the indicated values.
Protocol	Choose `LDAP` for the LDAP Interface for LDAP.
Connection	This field is not applicable to the LDAP protocol.
Authenticate Server	This field is not used for the LDAP protocol. Authentication is always used for LDAP.
PI User	This field is not applicable to the LDAP protocol.
Max Secondary Connections	This field is not used for the LDAP protocol. Only a single connection per ASP is used for LDAP.

Defining a DAP Operation for LDAP

Introduction

The DAP framework defines the concept of an “Operation”. Operations may be grouped into “Operation Sets”. The following discussion is primarily concerned with only those aspects of operation configuration specifically related to the LDAP Interface for DAP.

The LDAP operation definition is effectively a template which describes how ACS parameters correlate to the LDAP request. Every DAP Request node instance with an ACS control plan must specify the operation it will use.

Specifically the operation defines:

- Which LDAP ASP will receive this request.
- What parameters are included in the LDAP request.
- What are the sources within the ACS control plan call context for these parameters.
- What parameters are expected in the associated LDAP response.
- Where are these parameters copied into the ACS control plan call context.

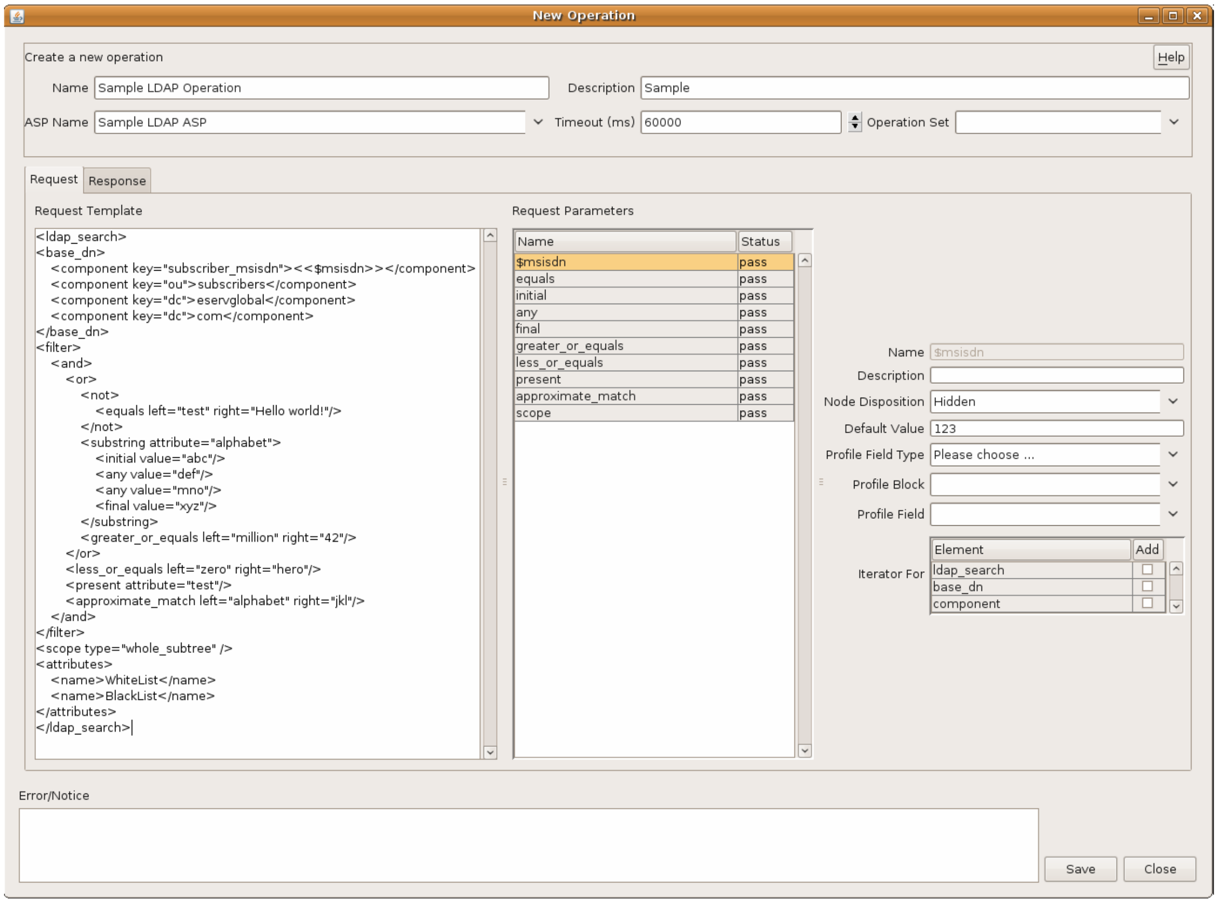
Basic configuration

There as some basic configuration settings on the operation screen. In addition there are two tabs - Request and **Response** which contain more detailed configuration.

Field	Notes
Name	A unique name for this operation.
Description	An optional description field for this ASP.
ASP Name	Each operation is associated with a specific ASP. Choose your predefined LDAP ASP from the combo box.
Timeout	Specify the time-out in milliseconds to use for requests based on this operation. This must be strictly greater than zero.
Operation Set	Choose an operation set for this operation to belong to, or an empty operation set.

Request configuration

The **Request** parameters tab defines the encoding of the LDAP request.



Name: Sample LDAP Operation
Description: Sample
ASP Name: Sample LDAP ASP
Timeout (ms): 60000
Operation Set:

Request Template

```
<ldap_search>
<base_dn>
   <component key="subscriber_msisdn"><<$msisdn>></component>
   <component key="ou">subscribers</component>
   <component key="dc">eservglobal</component>
   <component key="dc">com</component>
</base_dn>
<filter>
   <and>
      <or>
         <not>
            <equals left="test" right="Hello world!"/>
         </not>
         <substring attribute="alphabet">
            <initial value="abc"/>
            <any value="def"/>
            <any value="mno"/>
            <final value="xyz"/>
         </substring>
         <greater_or_equals left="million" right="42"/>
      </or>
      <less_or_equals left="zero" right="hero"/>
      <present attribute="test"/>
      <approximate_match left="alphabet" right="jkl"/>
   </and>
</filter>
<scope type="whole_subtree" />
<attributes>
   <name>WhiteList</name>
   <name>BlackList</name>
</attributes>
</ldap_search>
```

Request Parameters

Name	Status
$msisdn	pass
equals	pass
initial	pass
any	pass
final	pass
greater_or_equals	pass
less_or_equals	pass
present	pass
approximate_match	pass
scope	pass

Name: $msisdn
Description:
Node Disposition: Hidden
Default Value: 123
Profile Field Type: Please choose ...
Profile Block:
Profile Field:

Iterator For

Element	Add
ldap_search	☐
base_dn	☐
component	☐

Error/Notice

Request parameter tab fields

Here are the key fields on the **Request** parameter tab.

Field	**Notes**
Request Template	This is the XML definition of the LDAP request. The DAP framework will determine the appropriate ACS control plan parameters that the LDAP interface needs to insert into the XML template, and will send these parameter values to the LDAP interface for DAP. The LDAP interface will substitute the parameters into the XML request template, and then will translate the complete XML request into LDAP, and finally will send it to the configured ASP. In addition, the LDAP-specific aspects of the Request Template field are discussed in further detail below.
Request Parameters	See descriptions about: • how to define DAP request parameters. • the use of the parameter detail fields: **Name, Description**, **Node Disposition**, **Default Value**, **Profile Field Type**, **Profile Block**, **Profile Field** and **Iterarator**.

Request template

The Request Template fields is (with one exception) a complete, well-formed XML document which must match the LDAP interface for DAP Request Schema file:

```
/IN/service_packages/DAP/etc/ldap_request_schema.xsd
```

The exception to the well-formed XML is the use of "<< $var >>" within the XML document. These are special place-holders which denote DAP substitution variables.

A DAP substitution variable is indicated by either:

- An element with empty content, and/or
- An element containing an explicit "<< $var >>" substitution variable.

For example:

```
<var1 this="that"></var1>
<param me="you"><< $var2 >></param>
<param he="him" foo="bar"><< $var3 >></param>
```

Examining the above, the DAP screen would determine that before sending this request, there are three variable fields in this request which need to be filled from values in this control plan call context. The variables are "var1" (empty content), "var2" and "var3" (explicit named).

LDAP Request template

A request template for an LDAP operation must have a specific syntax, as defined in the *ldap_request_schema.xsd* schema file. In general, the fields of the LDAP operation are standardised. The only significant area for flexibility is in the definition of the <filter> element, which represents the "filter" attribute of the LDAP SearchRequest as per RFC1777. This is an arbitrary nesting of comparisons, combined with a logical and/or test.

Please refer to RFC1777 and the request XSD file for the definitive definition. However, the following "complete example" shows all possible LDAP schema elements in use.

```
<ldap_search>
```

```
<base_dn>
    <component key=”subscriber_msisdn”><<$msisdn>></component>
    <component key=”ou”>subscribers</component>
    <component key=”dc”>example</component>
    <component key=”dc”>com</component>
</base_dn>
<filter>
    <or>
        <and>
            <not>
                <equals left=”vpnlevel” right=”NoAccess”>
            </not>
            <substring attribute=”yahooUsername”>
                <initial value=”ax”>
                <any value=”hot”>
                <any value=”l33t”>
                <final value=”zz”>
            </substring>
            <greater_or_equals left=“cashBalance” right=”1000”>
        </or>
        <less_or_equals left=”yearsToLive” right=”10”>
        <present attribute=”royaltyFlag”>
        <approximate_match left=”formOfAddress” right=”sir”>
    </or>
</filter>
<scope type=”whole_subtree”>
<attributes>
    <name>WhiteList</name>
    <name>BlackList</name>
    <name>MotherMaidenName</name>
</attributes>
</ldap_search>
```

Elements

Here are the elements in the ldap_search Operation Request template.

Element	Notes
ldap_search	This is the outer level element which denotes the template for an LDAP SearchRequest operation as specified in RFC1777.
base_dn	This defines the content of the baseObject parameter in the LDAP SearchRequest. It consists of one or more component sub-elements.
component	This defines the components of the base in sequence from the outermost level to the root of the DN. In the example shown here, the subscriber's MSISDN is included in the DN. Alternatively, you might specify a base DN as a point in the namespace tree further towards the root and request the server to search the whole subtree. The base DN and filter strategy is entirely dependent on the nature of the application and the nature of the LDAP server's data structure.

filter	This element specifies the search filter. Before designing a search filter, it is necessary first that you understand the role and potential of the filter field as defined in RFC1777. Secondly, it is necessary for you to understand the structure of the user data stored in the LDAP server. Once you have designed the LDAP search filter, this Operation Request Template allows you to specify an XML template in a natural fashion with a direct mapping into sub-elements. Recall that the filter matching specified here is actually performed by the LDAP server as part of its process for locating the appropriate user object for which to return the requested attributes. Refer to the documentation for your LDAP server for the authoritative specification of how filter elements will be interpreted.
and	This element evaluates to “true” if all contained filter elements evaluate to “true”.
or	This element evaluates to “true” if one or more contained filter elements evaluate to “true”.
not	This element evaluates to “true” if the single filter element it contains evaluates to “false”.
substring	This elements evaluates to “true” if the attribute with name specified by parameter “attribute” is determined to substring match according to the sub-elements. A substring element must contain at least one “initial”, “any” or “final” attributes. If multiple “initial”, “any” or “final” attributes are present, then *all* must match. E.g. in the example given, the attribute of name “yahooUserName” must match the pattern “ax*hot*l33t*zz” where * indicates a sequence of zero or more characters.
initial	Within a substring element, determines that the element must begin with the specified sequence in order for substring to match.
any	Within a substring element, determines that the element must contain within it the specified sequence in order for substring to match. If multiple “any” elements are specified, the order may be significant.
final	Within a substring element, determines that the element must end with the specified sequence in order for substring to match.
equals	This elements evaluates to “true” if the attributed with name specified by parameter “left” is determined to have value equal to that specified by parameter “right”.
greater_or_equals	This elements evaluates to “true” if the attributed with name specified by parameter “left” is determined to have value greater than or equal to that specified by parameter “right”.
less_or_equals	This elements evaluates to “true” if the attributed with name specified by parameter “left” is determined to have value less than or equal to that specified by parameter “right”.

present	This elements evaluates to “true” if the attributed with name specified by parameter “attribute” is present for the object.
approximate_match	This elements evaluates to “true” if the attributed with name specified by parameter “left” is determined by the ASP server to have value “approximately equal” to that specified by parameter “right”. Refer to RFC1777 and the documentation for the LDAP Server.
scope	This should specify type=”whole_subtree”. Other values may be supported in the future.
attributes	The attributes element contains the names of all the returned attributes which are to be requested from the LDAP server and which are to copied from the LDAP response into the ACS call context. The configuration on the Response Tab defines where these elements are to be placed within the call context. Some elements may be multi-valued, in which case they may be placed within sub-profiles. Refer to the *Data Access Pack User's & Technical Guide* for further information on configuring DAP nodes to receive multi-valued fields.
name	Each name element specifies one attribute name to be requested from the LDAP server and written into the ACS call context. Again, note that some attributes are multi-valued. Refer to the *Data Access Pack User's & Technical Guide*.

Note: The current behaviour in the DAP request template screen is that it treats all elements with empty content as if they were variables. This can result in some element tags (e.g. “present” and “scope”, and others) from always being incorrectly interpreted as variables.

When this occurs, you will need to simply configure this “ghost parameter” as having an empty string default value.

Request parameters

The second and third columns in the **Request** tab for the Operation screen define the source and default values for the per-invocation variables which are filled from the buffers in the ACS control plan call context at run-time.

The use and interpretation of these fields is the same for LDAP as for other DAP protocols.

Response configuration

The **Response** tab contains the configuration allowing the operator to define the destination within the ACS control plan call context where user data received in the LDAP response will be placed. When specifying the XPATH for returned LDAP fields, use an XPATH such as:

```
//name[@key="userFieldName"]/text()
```

Note that the “text()” refers to the content of the returned value, not the final stored type in the profile. Hence “text()” is used for when the returned value to be written to the call context is a string or an integer.

Note that the double quotes around the returned user field name are required.

Otherwise, the use and interpretation of all configuration fields on this tab is the same for LDAP as for other DAP protocols.

DAP Control Plan

Using the DAP LDAP operation

To use the DAP LDAP operation defined according to the previous sections, simply use the DAP Request within an ACS control plan. The specific DAP Request node instance must be configured to specify the Operation previous configured.

Chapter 6

Background Processes

Overview

Introduction

This chapter explains the processes that are started automatically by Service Logic Execution Environment (SLEE).

Note: This chapter also includes some plug-ins to background processes which do not run independently.

In this chapter

This chapter contains the following topics.

c_rehash

Purpose

c_rehash is an *openssl* (on page 92) utility that takes a certificates directory as an argument. For each certificate file in the directory c_rehash creates a symbolic link to the certificate file, where the symbolic link name is the hash value of the certificates file. This enables fast certificate lookup for programs that search using the certificate hash value.

Location

c_rehash is located in the following directory on SLC and VWS nodes:

/IN/services_packages/DAP/bin

Startup

c_rehash is run by the **dapReadyCertificates.sh** (on page 95) script.

Configuration

This binary has no specific configuration. **dapReadyCertificates.sh** uses the variable DEFAULT_C_REHASH_PATH to define the location of c_rehash. The default value for the DEFAULT_C_REHASH_PATH variable is **/IN/service_packages/DAP/bin**.

dapIF

Purpose

dapIF is a SLEE interface. It is the main Oracle Communications Network Charging and Control Data Access Pack DAP client that sends and receives XML requests to external ASPs. It listens for SLEE requests and messages from ASPs.

It can trigger a PI command from a control plan using the DAP Send Request feature node when communicating with an ASP using the PIXML protocol.

Location

This binary is located on SLC and VWS nodes.

Startup

The interface is started by the SLEE, through the **/IN/service_packages/DAP/bin/dapIF.sh** shell script.

Configuration

dapIF is configured in the `DAP` section of **eserv.config** and the DAP Resources screen.

For more information about the:

- **eserv.config** parameters, see *DAP* **eserv.config** *configuration* (on page 47)
- DAP Resources screen, see *Resources* (on page 21)

Command line parameters

dapIF accepts the following command-line parameters at start up.

```
dapIF [-u usr/pwd|--user usr/pwd]
```

Note: Either the `-u` or the `--user` option can be used.

-u usr/pwd

Syntax:	`-u` *`usr/pwd`*
Description:	The userid and password combination to use to log into the local Oracle instance.
Type:	String
Optionality:	Optional (default used if not set).
Allowed:	
Default:	/
Notes:	Cannot be used with the `--user /usr/pwd` (on page 88) option.
Example:	`-u smf/smf`

--user /usr/pwd

Syntax:	`--user` *`usr/pwd`*
Description:	The userid and password combination to use to log into the local Oracle instance.
Type:	String
Optionality:	Optional (default used if not set).
Allowed:	
Default:	/
Notes:	Cannot be used with the `-u usr/pwd (on page 88)` option.

Example: `--user smf/smf`

Failure

In case of failure alarms will be raised to the syslog.

Output

There is no output from this process.

dapMacroNodes

Purpose

This slee_acs plug-in provides the DAP macro nodes.

The nodes provided are:

- Send Request
- DAP Request

For more information about:

- Macro node libraries, see *ACS Technical Guide*
- The CPE, see *CPE User's Guide*

Location

This library is located on SLCs.

Startup

If dapMacroNodes is configured in **acs.conf**, it is made available to slee_acs when slee_acs is initialized. It is included in the `acsChassis` section of **acs.conf** in a `MacroNodePluginFile` entry as follows:

```
acsChassis
 MacroNodePluginFile dapMacroNodes.so
```

Configuration

dapMacroNodes is configured in the `DAP` section of **eserv.config**. For more information, see *DAP* **eserv.config** *configuration* (on page 47).

dapTypeConversion

Purpose

This SLEE-ACS plug-in provides conversions from ACS profile fields to types usable by DAP.

Location

Located on SLCs.

Startup

This library will be loaded based on configuration made by the packages on install.

Configuration

This library has no specific configuration.

ldapIF

Purpose

The ldapIF process is the main process in the LDAP IF component. It is a SLEE interface process. It accepts SLEE request messages initiated from DAP Request nodes within slee_acs control plans and translates them into LDAP requests to be sent over TCAP to waiting LDAP servers. The response is returned via the reverse path back to the DAP Request node.

Location

This binary is located on SLC nodes.

Startup

This task is started by the SLEE, by the following lines (or similar) in SLEE.cfg:

```
 INTERFACE=ldapIF ldapIF.sh /IN/service_packages/DAP/bin EVENT
```

Notes:

- ldapIF.sh is a shell script which starts the ldapIF process.
- The above are defaults and may vary.
- Only a single instance of the ldapIF process will be started.

Configuration

Global configuration for ldapIF is in eserv.config (or wherever the ESERV_CONFIG_FILE environment variable indicates). The ldapIF can run without explicit configuration by using internal default configuration settings. Sending a SIGHUP signal to the ldapIF will force it to reload the eserv.config configuration.

For more information regarding ldapIF global configuration via eserv.config, see *Global Configuration* (on page 72). For more information about causing ldapIF to reload its configuration, see *Loading* **eserv.config** *configuration changes* (on page 72).

The ldapIF also uses per-ASP and per-operation settings. These are read from the SLC database (which contains copies of data replicated from the master configuration on the USMS nodes). This database configuration is reloaded periodically as part of the normal ldapIF processing.

For more information regarding ldapIF service configuration via the DAP GUI, see *DAP Resource Configuration* (on page 78).

Example configuration

For an example of an LDAP IF configuration section of an eserv.config file, see *Example eserv.config file section* (on page 73).

Failure

The ldapIF will be monitored by the SLEE watchdog. The watchdog will restart ldapIF if it fails to respond to regular heartbeat events. For more details about how the watchdog monitors SLEE processes, see the *SLEE Technical Guide*.

ldapIF generates standard SMS alarm log messages to alert operators to any misconfiguration or abnormal processing. For more information about the alarms generated by ldapIF, see *LDAP Interface for DAP Alarms Guide*.

Output

The ldapIF process writes error messages to the system messages file. Under normal processing it will echo all alarm messages to STDERR. As a SLEE process, this will be merged with all SLEE output.

See Debug output for more information.

Status reports

At any time, you can send ldapIF a SIGUSR1 which will cause it to generate a status output listing to STDOUT. This will usually be redirected to `/IN/service_packages/DAP/tmp/ldapIF`.log file.

Sample output is:

```
** Connection Status **
  ASP: LDAP1
  Host: parker, port: 389
  Connection state: IDLE
** Connection Status **
  ASP: LDAP2
  Host: grimm, port: 389
  Connection state: READY
    Time connected: 11s
    Number of pending (unsent) requests: 0
    Number of requests waiting for response: 0
    Total requests processed: 30
```

Note: Generating the status report will not affect the connections, and can be done safely while the system is under production load.

libdapChassisActions

Purpose

This slee_acs plug-in implements the chassis actions which are used by the DAP macro nodes when they need to interact with components outside slee_acs.

Location

This library is located on SLCs.

Startup

If libdapChassisActions is configured in **acs.conf**, it is made available to slee_acs when slee_acs is initialized. It is included in the `acsChassis` section of **acs.conf** in a `ChassisPlugin` entry.

```
acsChassis
 ChassisPlugin libdapChassisActions.so
```

Configuration

This binary has no specific configuration.

libDAPManager.so

Purpose

The **libDAPManager.so** is a combined connection manager and XML interface. A DAP client will use **libDAPManager.so** to communicate with an ASP.

Start-up

The library is linked at run time by a DAP client.

Location

libDAPManager.so is located wherever a DAP client is installed (for example, on SMS, SLC or VWS.

eserv.config configuration

libDAPManager has configuration available in the `DAP` section of **eserv.config**. For more information, see *DAP* **eserv.config** *configuration* (on page 47).

Command line parameters

Command line parameters for **libDAPManager.so** are the same as those for dapIF. Refer to Parameters.

Failure

In the event of a failure, alarms will be written to the system log.

Output

There is no output from this process.

openssl

Purpose

openssl is used by **dapReadyCertificates.sh** (on page 95) and *c_rehash* (on page 87) to concatenate and rehash certificate files, creating links to the certificate files by hash value so that DAP can quickly find the certificates that it requires.

Location

openssl is part of the operating system installation. It is used on the SLC and VWS nodes. Depending on the operating systems version openssl is present as one of:

- **/usr/sfw/bin/openssl**
- **/usr/bin/openssl** with a symbolic link from **/usr/sfw/bin/openssl** to **/usr/bin/openssl**

Startup

openssl is started by **dapReadyCertificates.sh** (on page 95).

Note: dapReadyCertificates.sh attempts to use openssl at the location configured in the *openSSLPath* (on page 57) parameter in the **eserv.config** file.

Configuration

openssl has some configuration which is set when used by **dapReadyCertificates.sh** (on page 95). To configured this process, use the configuration available to **dapReadyCertificates.sh**.

sqlite3

Purpose

sqlite3 is an embedded SQL database engine which reads and writes directly to the database file on disk. Programs that link with the sqlite3 library can have SQL database access without running a separate RDBMS process.

It handles the DAP pending queue.

Location

This binary is located on the SMS node.

Configuration

sqlite3 accepts the following command line parameters.

```
/IN/service_packages/DAP/bin/sqlite3 path/pendingRequests.db '<VACUUM|PRAGMA
integrity_check>;'
```

To resize the database after a lot of data has been removed, run VACUUM as acs_oper.

To raise alarms about malformed db files, use PRAGMA integrity_check.

Chapter 7

Tools and Utilities

Overview

Introduction

This chapter explains the Oracle Communications Network Charging and Control Data Access Pack DAP tools and utilities that are available.

In this chapter

This chapter contains the following topics.

dapReadyCertificates.sh

Purpose

dapReadyCertificates.sh prepares the certificates located in the directory specified in the `certificatePath` (on page 51) parameter into the form required by *openssl* (on page 92) (by concatenating them into human-readable form and running *c_rehash* (on page 87) on them). **dapReadyCertificates.sh** concatenates the files into the file specified by the `certificatesName` (on page 52) parameter.

See *Certificate Management* (on page 13) for an overview of how certificates are handled.

Warning: The concatenated file is overwritten each time **dapReadyCertificates.sh** is run. To keep the existing file, move it to a directory other than the one specified by `certificatePath`, or rename it with a suffix other than **.pem** (otherwise it will be concatenated into the new file along with the other ***.pem** files in `certificatePath`).

Location

This binary is located on SLC and VWS nodes.

Configuration

dapReadyCertificates.sh accepts the following parameters from the `DAP` section of **eserv.config**:

- `certificatePath` (on page 51)
- `certificatesName` (on page 52)
- `openSSLPath` (on page 57)

dapSchemaTool

Purpose

This tool is used to export existing ASPs and operations from one server (that is, a testing server) and import them to another server (that is, a production server).

Usage

Format

```
dapSchemaTool -E data|-I data -n name [-s operation_set_name] [-f filename] -u
db_user -p db_password [-c TNS_connect_string]
```

Export example

```
/dapSchemaTool -E asponly -n abc -u SMF -p SMF -c server_SMF
```

Import example

```
/dapSchemaTool -I operation -n xyz -f testOutput.txt -u SMF -p SMF -c server_SMF
```

Arguments

This table describes the function of each command argument.

Argument	Description
-E	Export from database
-I	Import to database
-n	ASP, operation, or operation set name to export or import
-s	Operation set name (optional, only required if the given operation name is not unique)
-f	Filename for exported data (optional, required for importing, uses the standard output for exporting if not specified)
-u	Oracle username of database to use
-p	Oracle password of database to use
-c	Oracle TNS connect string (optional, uses $ORACLE_SID to connect to the local database if not specified)
Data arguments for-E and -I options - one of these must be used	
asponly	Only the given ASP
asp	The given ASP and all attached operations, as well as the operation set each operation belongs to, and all parameters each operation contains
operation	The given operation, its parameters, and the associated ASP and the operation set (the associated operation set will be exported but will not be imported using this argument)
operationset	The given operation set and its all operations, as well as the associated ASP and parameters of each operation

Chapter 8

About Installation and Removal

Overview

Introduction

This chapter provides information about the installed components for the Oracle Communications Network Charging and Control (NCC) application described in this guide. It also lists the files installed by the application that you can check for, to ensure that the application installed successfully.

In this Chapter

This chapter contains the following topics.

Installation and Removal Overview

Introduction

For information about the following requirements and tasks, see *Installation Guide*:

- NCC system requirements
- Pre-installation tasks
- Installing and removing NCC packages

Checking the Installation

Checking the DAP SMS Installation

On successful installation, the package will have created the following directories:

- **/IN/service_packages/DAP/db**
- **/IN/service_packages/DAP/lib**
- **/IN/service_packages/DAP/tmp**

The following feature nodes will have been installed and added to the ACS database:

- Send Request
- DAP Send Request
- DAP VXML

Checking the DAP SLC Installation

On successful installation, the package will have created the following directories:

- **/IN/service_packages/DAP/bin**
- **/IN/service_packages/DAP/db**

- **/IN/service_packages/DAP/etc**
- **/IN/service_packages/DAP/lib**
- **/IN/service_packages/DAP/tmp**

The following binaries and interfaces will have been installed:

- **/IN/service_packages/DAP/bin/dapIF**

The following configuration files will have been installed:

- **/IN/service_packages/DAP/etc/example.eserv.config**

The following shared libraries will have been installed:

- **/IN/service_packages/DAP/lib/dapMacroNodes.so**
- **/IN/service_packages/DAP/lib/libdapChassisActions.so**

Glossary of Terms

ACS

Advanced Control Services configuration platform.

ANI

Automatic Number Identification - Term used in the USA by long-distance carriers for CLI.

ASN.1

Abstract Syntax Notation One - a formal notation used for describing data transmitted by telecommunications protocols. ASN.1 is a joint ISO/IEC and ITU-T standard.

ASP

- Application Service Provider, or
- Application Server Process. An IP based instance of an AS. An ASP implements a SCTP connection between 2 platforms.

CC

Country Code. Prefix identifying the country for a numeric international address.

CLI

Calling Line Identification - the telephone number of the caller. Also referred to as ANI.

Connection

Transport level link between two peers, providing for multiple sessions.

CPE

Control Plan Editor (previously Call Plan Editor) - software used to define the logic and data associated with a call -for example, "if the subscriber calls 0800 *nnnnnn* from a phone at location *xxx* then put the call through to *bb bbb bbbb*".

cron

Unix utility for scheduling tasks.

DAP

Data Access Pack. An extension module for ACS which allows control plans to make asynchronous requests to external systems over various protocols including XML and LDAP.

DTMF

Dual Tone Multi-Frequency - system used by touch tone telephones where one high and one low frequency, or tone, is assigned to each touch tone button on the phone.

GUI

Graphical User Interface

HTML

HyperText Markup Language, a small application of SGML used on the World Wide Web.

It defines a very simple class of report-style documents, with section headings, paragraphs, lists, tables, and illustrations, with a few informational and presentational items, and some hypertext and multimedia.

HTTP

Hypertext Transport Protocol is the standard protocol for the carriage of data around the Internet.

IN

Intelligent Network

IP

1) Internet Protocol

2) Intelligent Peripheral - This is a node in an Intelligent Network containing a Specialized Resource Function (SRF).

IP address

Internet Protocol Address - network address of a card on a computer.

ISDN

Integrated Services Digital Network - set of protocols for connecting ISDN stations.

ITU

International Telecommunication Union

Messaging Manager

The Messaging Manager service and the Short Message Service components of Oracle Communications Network Charging and Control product. Component acronym is MM (formerly MMX).

MIN

Mobile Identification Number, also known as an MSID.

MM

Messaging Manager. Formerly MMX, see also *XMS* (on page 103) and *Messaging Manager* (on page 100).

MSID

Mobile Subscriber Identification, also known as an MIN.

MSISDN

Mobile Station ISDN number. Uniquely defines the mobile station as an ISDN terminal. It consists of three parts; the country code (CC), the national destination code (NDC) and the subscriber number (SN).

PI

Provisioning Interface - used for bulk database updates/configuration instead of GUI based configuration.

PICS

Protocol Implementation Conformance Statement applicable to the relevant protocol.

Service Provider

See Telco.

SGML

Standard Generalized Markup Language. The international standard for defining descriptions of the structure of different types of electronic document.

SLC

Service Logic Controller (formerly UAS).

SLEE

Service Logic Execution Environment

SMS

Depending on context, can be:

- Service Management System hardware platform
- Short Message Service
- Service Management System platform
- NCC Service Management System application

SN

Service Number

SOAP

Simple Object Access Protocol. An XML-based messaging protocol.

SQL

Structured Query Language is a database query language.

SRF

Specialized Resource Function – This is a node on an IN which can connect to both the SSP and the SLC and delivers additional special resources into the call, mostly related to voice data, for example play voice announcements or collect DTMF tones from the user. Can be present on an SSP or an Intelligent Peripheral (IP).

SSL

Secure Sockets Layer protocol

SSP

Service Switching Point

TCAP

Transaction Capabilities Application Part – layer in protocol stack, message protocol.

TCP

Transmission Control Protocol. This is a reliable octet streaming protocol used by the majority of applications on the Internet. It provides a connection-oriented, full-duplex, point to point service between hosts.

Telco

Telecommunications Provider. This is the company that provides the telephone service to customers.

Telecommunications Provider

See Telco.

TLS

Transport Layer Security. Cryptographic protocol used to provide secure communications. Evolved from SSL.

URL

Uniform Resource Locator. A standard way of specifying the location of an object, typically a web page, on the Internet.

VWS

Oracle Voucher and Wallet Server (formerly UBE).

WSDL

Web Services Description Language.

XML

eXtensible Markup Language. It is designed to improve the functionality of the Web by providing more flexible and adaptable information identification.

It is called extensible because it is not a fixed format like HTML. XML is a `metalanguage' — a language for describing other languages—which lets you design your own customized markup languages for limitless different types of documents. XML can do this because it's written in SGML.

XMS

Three letter code used to designate some components and path locations used by the Oracle Communications Network Charging and Control *Messaging Manager* (on page 100) service and the Short Message Service. The published code is *MM* (on page 100) (formerly MMX).

Index