Oracle® Communications ASAP

Server Configuration Guide

Release 7.2

E24629-01

April 2012

ORACLE®

Oracle Communications ASAP Server Configuration Guide, Release 7.2

E24629-01

Contents

6 Managing the Service Activation Request Manager

7 Configuring Network Element Processors, Resource Pools, and Devices

A asap_utils

B Stored Procedures (Deprecated)

Preface

This guide explains how to configure Oracle Communications ASAP using Service the Activation Configuration Tool (SACT), ASAP services using XML, ASAP security, and dynamic network element (NE) routing. It also describes how to use stored procedures for configuring ASAP services, system events and alarms, service request processors (SRPs), network elements, and network element processors (NEPs).

Audience

This document is intended for system administrators, system integrators, and other individuals who need to maintain and work with ASAP.

Related Documents

For more information, see the following documents in the ASAP Release documentation set:

- *Oracle Communications ASAP Release Notes*
- *Oracle Communications ASAP Concepts*
- *Oracle Communications ASAP Installation Guide*
- *Oracle Communications ASAP Service Request Translator User's Guide*
- *Oracle Communications ASAP Order Control Application User's Guide*
- *Oracle Communications ASAP Cartridge Development Guide*
- *Oracle Communications ASAP Security Guide*
- *Oracle Communications ASAP System Administrator's Guide*
- *Oracle Communications ASAP Developer's Guide*

Note: To download the *ASAP Developer's Guide* from the Oracle software delivery Web site, you must select **Oracle Communications Service Activation Developer Documentation Pack**. You can visit the Oracle software delivery Web site at:

`http://edelivery.oracle.com`

1

ASAP Server Configuration Overview

This chapter provides an overview of Oracle Communications ASAP server configuration tasks.

Overview of ASAP Server Configuration Tasks

After you have installed ASAP and defined the initial user security, perform the following procedures to configure ASAP.

- Set up ASAP security. For more information, see *ASAP Security Guide*.
- Build custom service request processor (SRP) interfaces or use the Java SRP or C SRP emulator that ASAP provides. ASAP provides two mechanisms to configure SRP interfaces: the Service Activation Configuration Tool (SACT) and stored procedures. See "Configuring ASAP Servers" for more information.
- Configure network element processors (NEPs) and network elements (NEs). To configure NEPs, use the SACT. See "Configuring Network Element Processors, Resource Pools, and Devices" for more information. To configure NEs, see Oracle Communications Design Studio for Activation and *ASAP Cartridge Development Guide*.
- Deploy services. ASAP provides two mechanisms to configure ASAP services: the installCartridge script and Design Studio for Activation. For more information, see "Configuring ASAP Servers" and see *ASAP Installation Guide*.
- Configure Atomic Service Description Layer (ASDL) command routing options (see *ASAP Cartridge Development Guide*).
- Configure system events and alarms. For more information, see *ASAP System Administrator's Guide*.
- Tune ASAP by modifying configuration parameters. For more information, see *ASAP System Administrator's Guide*.
- Optionally configure remote servers. It may be necessary to configure ASAP to execute one or more of its processes on a separate, or remote, server. This can be done to improve performance (for example, to distribute processing loads) or because certain downstream interfaces or networks are not accessible from the machine where ASAP is running. For more information, see *ASAP Installation Guide*.
- Monitor ASAP using the asap_utils utility. For more information, see *ASAP System Administrator's Guide*.

2

Configuring ASAP Servers

This chapter provides information about configuring Oracle Communications ASAP servers using the Service Activation Configuration Tool (SACT).

About the Service Activation Configuration Tool

The SACT provides a way of adding, deleting, and modifying ASAP servers and ASAP server parameters, including:

- ASAP servers such as:
  - Network element processors (NEPs) and Java NEPs (JNEPs)
  - Service request processors (SRPs)
  - Slave control servers
- Global configuration parameters
- Server specific configuration parameters
- NEP resource pools (connectionPools)

> **Note:** The SACT does not configure system events and alarms. For information on configuring system events and alarms, see *ASAP System Administrator's Guide*.
>
> Reference information about the Java NEP (JNEP), the Java SRP (JSRP), and Java Management Extensions (JMX) can be found in *ASAP Online Reference*.

The SACT uses XML server configuration files to configure ASAP, and has several advantages over the use of stored procedures or scripts. By using the SACT to load XML server configuration files, you are insulated from the internal structure of ASAP, including database languages, such as Oracle syntax. Moreover, you can reuse an XML configuration file for similar configurations and use version control for the file.

The SACT supports server side configuration activities, and the Service Activation Deployment Tool (SADT) supports ASAP cartridge or individual XML service configuration file deployment. For more information about SADT, see *ASAP Cartridge Development Guide*.

To work with the SACT, Oracle recommends that you have experience with:

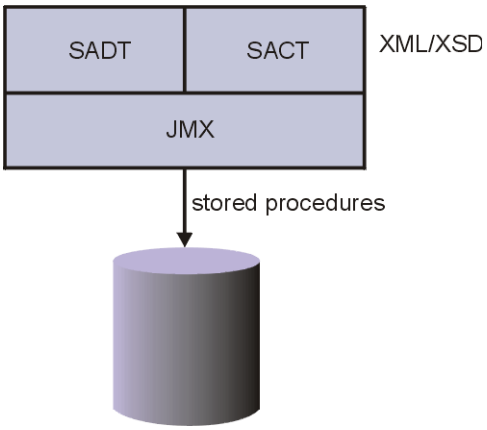
- XML
- XML schemas

- ASAP system configuration principles
- ASAP service modeling principles

About Service Activation Configuration Tool Resource Definitions

Figure 2–1 shows how the SACT and the SADT deploy an XML file or service activation model archive (SAR) file through a JMX management interface located in the ASAP WebLogic server instance when you first install ASAP. You can use the SADT or Oracle Communications Design Studio to deploy a cartridge into ASAP.

Figure 2–1 XML Deployment



Before the SACT can process an XML file and configure ASAP server resources through the JMX interface, the XML file must conform to the resource definitions defined in the *ASAP_Home*/**xml/xsd/ActivationConfig.xsd** schema file. You can view a sample XML configuration file that conforms to this schema in the *ASAP_Home*/**sample/sadt/SampleCommonConfig.xml** file.

> **Note:** Schema validation for XML data processed by the SACT and the SADT is turned off by default.

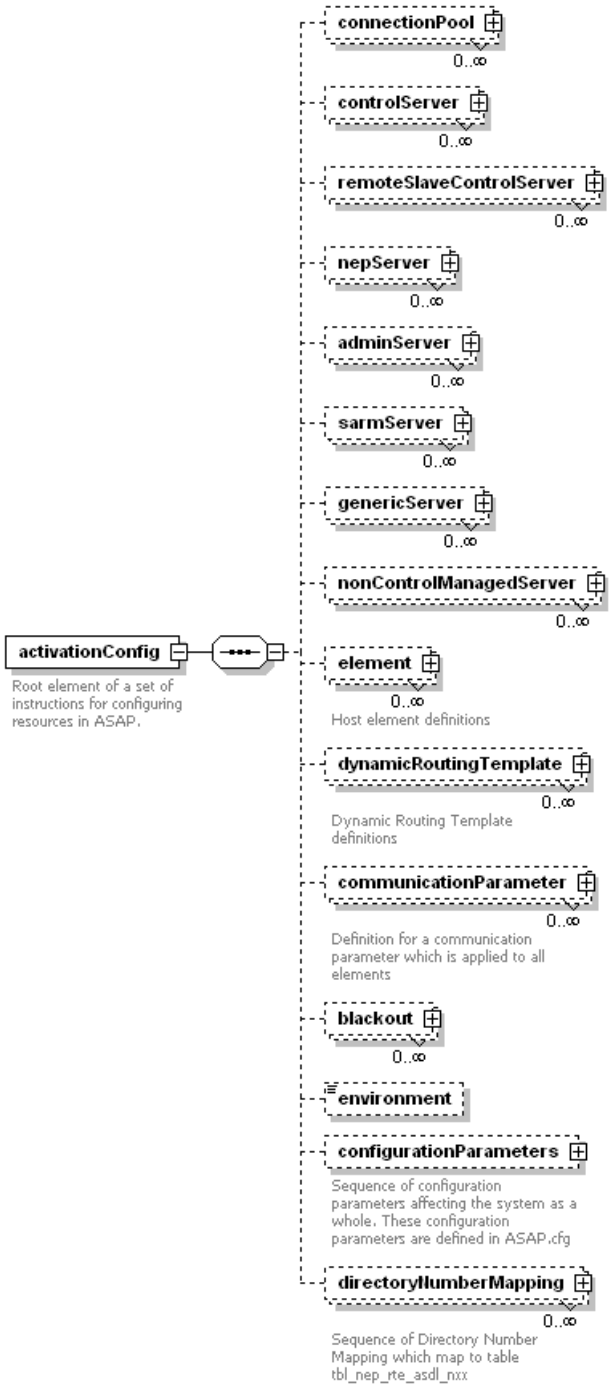
You can use the SACT to:

- Add new resources such as ASAP servers, connection pools and their devices, and communication parameters
- Delete existing resources such as ASAP servers, connection pools and their devices, and communication parameters
- Modify existing resources such as ASAP servers, connection pools and their devices, and communication parameters

About Using the Service Activation Schema to Write an XML File

Figure 2–2 shows the **ActivationConfig.xsd** schema elements. Each element corresponds to configurable ASAP server or server parameter schema definitions which your XML server configuration file must conform to.

Figure 2–2 ActivationConfig.xsd Schema Elements



The ASAP installer creates some of the elements listed in Figure 2–2 during the installation process. For example, ASAP supports one Service Activation Request Manager (SARM), Admin, JSRP, and Order Control Application (OCA) server. You do not need to add or delete these resources using SACT.

You can use the SACT to add configuration parameters to the servers that the ASAP installer creates. Some elements you may need to add include:

- NEP and JNEPs (see "Configuring NEPs")

- SRPs (see "Configuring a C SRP Emulator")
- Slave control servers (see "About the Control and Daemon Servers")
- Resource pools (connectionPools) and devices (see "Configuring Resource Pools and Resource Pool Devices")
- **ASAP.cfg** configuration parameters (see "Configuration Parameters")

When authoring an XML configuration file consisting of several elements, observe the order in which elements are listed in the schema. For example, if defining an NEP server and a resource pool (connectionPool) in an XML configuration file, ensure that the NEP server definition is located before the resource pool definition since the **ActivationConfig.xsd** schema requires this order.

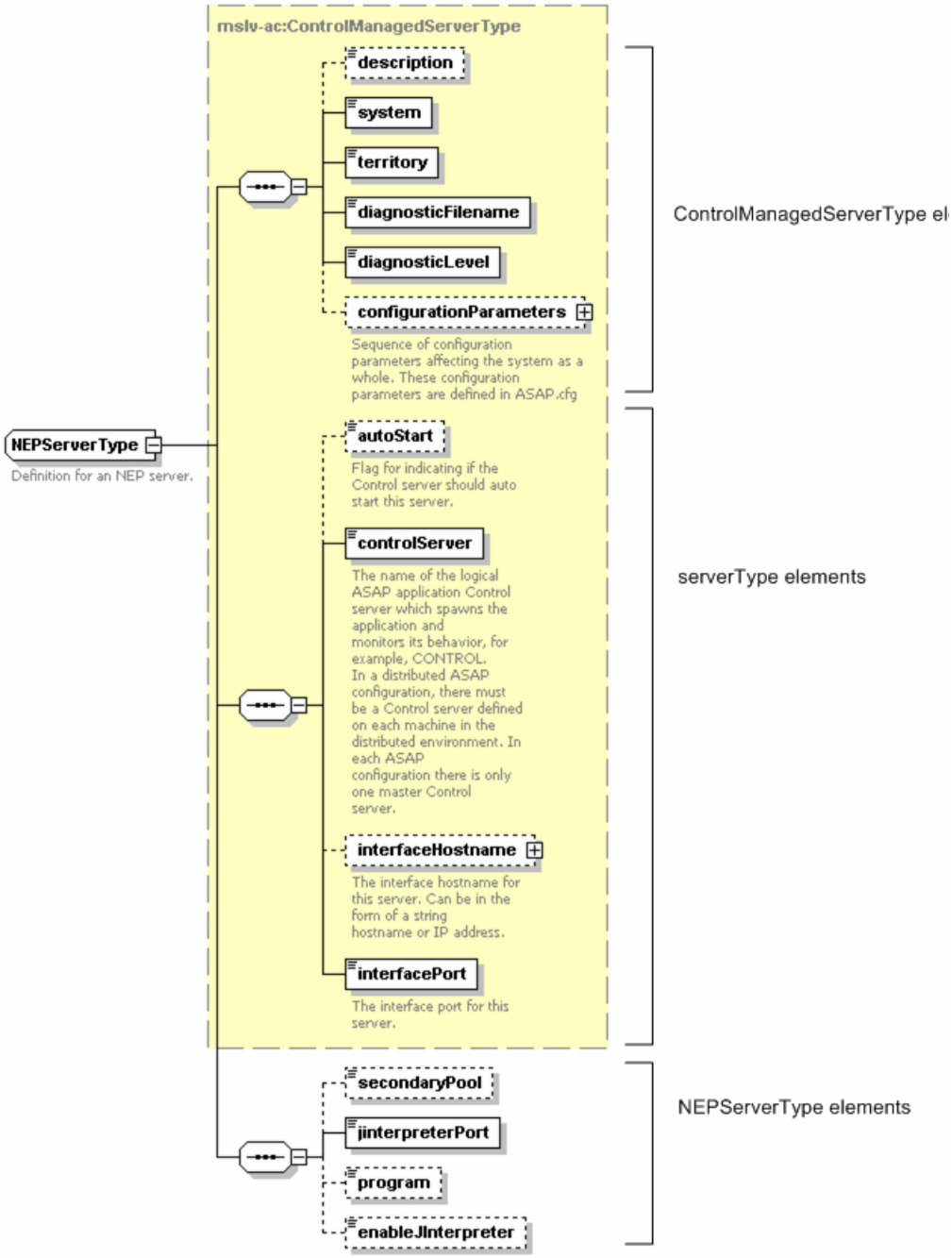
Oracle recommends that you use an XML editor to create an XML document, as XML editing tools usually have validators that ensure that the XML document is valid.

The schema contains base types and extensions for a given entity. For an NEP, for example, the XML schema identifies:

- The **base** type and **serverType**, which contains the most common entries: **description**, **system**, **territory**, **diagnosticFilename**, **diagnosticLevel**.
- **ControlManagedServerType**, an extension of the base **serverType**, which adds the **autoStart** flag, control server name, and interface information on top of the serverType.
- **NEPServerType**, an extension of **ControlManagedServerType**, which contains the elements **secondaryPool** and **jinterpreterPort**.

Figure 2–3 shows the structure that has to be observed when authoring an NEP configuration file. The XML configuration file must observe the order in which the elements appear in the schema.

Figure 2–3 NEPServerType Schema



The following is an excerpt from the **ControlManagedServerType** in the schema.

```
<xs:complexType name="ControlManagedServerType">
        <xs:complexContent>
                <xs:extension base="mslv-ac:ServerType">
                        <xs:sequence>
                                <xs:element name="autoStart" type="xs:boolean"
default="true" minOccurs="0">
                                        <xs:annotation>
                                                <xs:documentation>Flag for
indicating if the Control server should auto start this server.</xs:documentation>
```

```
                                        </xs:annotation>
                                    </xs:element>
                                    <xs:element name="controlServer"
type="mslv-ac:ServerNameType">
                                        <xs:annotation>
<xs:documentation>The name of the logical ASAP application Control server which
spawns the application and monitors its behavior, for example, CONTROL. In a
distributed ASAP configuration, there must be a Control server defined on each
machine in the distributed environment. In each ASAP configuration there is only
one master Control server.</xs:documentation>
                                        </xs:annotation>
                                    </xs:element>
                                    <xs:element name="interfaceHostname"
type="mslv-ac:HostType" minOccurs="0">
                                        <xs:annotation>
                                                <xs:documentation>The interface
hostname for this server. Can be in the form of a string hostname or IP
address.</xs:documentation>
                                        </xs:annotation>
                                    </xs:element>
                                    <xs:element name="interfacePort"
type="mslv-ac:PortNumberType">
                                        <xs:annotation>
                                                <xs:documentation>The interface
port for this server.</xs:documentation>
                                        </xs:annotation>
                                    </xs:element>
                            </xs:sequence>
                            <xs:attribute name="defaultControlServer"
type="mslv-ac:ServerNameType"/>
                            <xs:attribute name="defaultInterfacePort"
type="mslv-ac:PortNumberType"/>
                            <xs:attribute name="defaultInterfaceHostname"
type="xs:string"/>
                    </xs:extension>
            </xs:complexContent>
</xs:complexType>
```

About Configuring an XML Configuration File to Prompt for Values

You can write the XML configuration file in such a way that it prompts the user for values.

This feature is useful if you are a cartridge developer who creates productized cartridges for deployment in customer sites. ASAP cartridges can contain XML server configuraiton files that require ASAP server specific information. Using the prompting feature, you can write an installation program that enables the customer to populate ASAP server related values (for example, ASAP server IP addresses and resource pools). For more information about adding XML server configuration files to an ASAP cartridge, see *ASAP Cartridge Development Guide*.

Prompts are delimited in the XML file with the character %. For example:

```
<communicationParameter>
       <label>IP_ADDRESS</label>
              <value defaultValue="localhost">
                   <value>%NAME</value>
              </value>
              <description>String</description>
```

```
            <deviceName>DEV_1</deviceName>
</communicationParameter>
```

For every parameter delimited with % (**NAME** in the above example) the command-line client displays a prompt to request a value from the user. This prompt displays the default value **<value defaultValue="localhost">** and a description of the parameter in **<description>** element. The value the user enters at the prompt is used in the configuration file.

About Configuring an XML File to Replace Values with Environment Variables

You can author the XML file so that, on deployment, all values that are delimited with the character "$" are replaced with the environment variable of the same name. If the environment variable is not defined, the default value is used. This feature can save time when the ASAP server environment variables have already been defined in the *ASAP_Home*/**Environment_Profile**.

```
<communicationParameter>
       <label>NMTOKEN</label>
       <value defaultValue="SARM Name">
              <value>$SARM</value>
       </value>
       <description>String</description>
       <deviceName>NMTOKEN</deviceName>
</communicationParameter>
<communicationParameter>
       <label>NMTOKEN</label>
       <value>
              <value>$SYSTEM</value>
       </value>
       <description>String</description>
       <deviceName>NMTOKEN</deviceName>
</communicationParameter>
```

In the previous example, if the $SARM and $SYSTEM environment variables are not defined, the default Value in the XML server configuration file is used.

About the Service Activation Configuration Tool

There are three SACT scripts, and they perform the following functions:

- **sactConfig** is a configuration script with a replace option. You can use this script to add or replace elements.
- **sactConfigNR** is a configuration script without a replace option. You can use this script to add elements, but you cannot use it to replace existing elements.
- **sactUnconfig** is a script to unconfigure elements. You can use this script to delete existing elements.

These scripts use the *ASAP_Home*/**scripts/asapConfig** command to perform their functions. This command deploys one or more XML files. The XML files can be individual, or they can be bundled in a SAR file. **asapConfig** searches the SAR file for an XML deployment descriptor that describes how each XML component needs to be deployed, and for any **SarPatch_configure** and **SarPatch_unconfigure** files.

> **Note:** **SarPatch_configure** and **SarPatch_unconfigure** are used by the SACT and contain ASAP server configuration information. These files should be distinguished from **SarPatch** and **SarPatch_undeployed**, used by SADT. For more information about **SarPatch** and **SarPatch_undeploy**, see *ASAP Cartridge Development Guide*.

The **asapConfig** command uses the following syntax.

```
asapConfig -Dwlurl=WL_URL -Dwlusr=WL_user_name -Dwlpwd=WL_user_passwd  [-DENV_
ID=ENV_ID] [replace][silent] command cfg_file
```

Where:

- *WL_user_name*: Set this value to the ASAP WebLogic server instance user name.
- *WL_user_passwd*: Set this value to the ASAP WebLogic server password.
- *env_ID*: This value must be set to environment ID for the ASAP server.
- **silent**: This option enables silent mode. The configuration utility will use the defaults for all the prompting parameters or the options file instead of prompting the user for a value.
- **replace**: This option enables overwrite mode. This means that all resources specified in the file that conflict with existing resources will overwrite existing resources. If this option is not specified the default value for overwrite is false.
- *command*: This value can be either **configure** or **unconfigure**. This command string sets the utility in configure or unconfigure mode. The **asapConfig** command generates a response file if the command is **configure**. The file name has the same extension as the input file. For example, if the input file is JustSample.sar, the output response file may appear as JustSample.20030423.160253.sar. The output file is placed in the same directory where the cfg_file is located. It is recommended to keep the response file and use it for the unconfigure operation in future.
- *cfg_file*: This value can be either an XML configuration file or the name of the SAR that includes XML configuration files.
- (optional) **help**: This option displays a help screen.

> **Note:** Oraclerecommends that you restart ASAP after adding or changing configuration information with the SACT.

The SACT scripts simplify the use of the **asapConfig** command.

Configuring the SACT Scripts and UNIX Environment Variables

Before you can use the SACT scripts for the first time, you must configure the $CLASSPATH and $PATH UNIX environment variables and edit the SACT scripts so that they include the ASAP WebLogic server host name and port number.

To configure the SACT scripts:

1. Log on to a UNIX terminal.
2. Verify that the following JAR files are located in the $CLASSPATH of your UNIX environment:
   - xmlparserv2.jar

- ant.jar
- weblogic.jar
- asaplibcommon.jar

For example, use the echo command to verify that the JAR files are contains in the $CLASSPATH.

```
echo $CLASSPATH
```

3. Verify that the *ASAP_Home*/**scripts/asapConfig** script and **ant** are in the $PATH of you UNIX environment:

```
echo $PATH
```

4. Edit the SACT script you want to use and replace the variables indicated within the text of the script (**Dwlurl** requires the WebLogic host name and port number).

For example:

```
errorStrLen=30
wluser=system
wlpass="`$ASAP_BASE/scripts/GetCSFPassword $wluser`"

if [[ "$wlpass" = *notFound* ]] && [ ${#wlpass} -gt $errorStrLen ]; then
  echo "Invalid username!"
else
  asapConfig -Dwlurl=myhost:3456 -Dwlusr=$wluser -Dwlpwd=$wlpass -DENV_ID=$ENV_
ID replace configure $1
fi
```

5. Remove the comments and save the script.
6. Repeat this procedure for the other SACT scripts.

Running the SACT Scripts

Use the following procedure to run **sactConfig**, **sactUnconfig**, and **sactConfigNR**.

1. Ensure that your ASAP WebLogic server instance is running.
2. Source the *ASAP_home*/**Environment_Profile**:

```
. ./Environment_Profile
```

3. Start ASAP if it is not already running.

```
start_asap_sys
```

4. Go to the location of the XML file you created that conforms to the **ActivationConfig.xsd** schema (see "About Using the Service Activation Schema to Write an XML File").
5. Run one of the SACT scripts (**sactConfig**, **sactUnconfig**, or **sactConfigNR**).

For example:

```
sactConfig adding_one_nep.xml
```

6. Stop and restart ASAP.

For example:

```
stop_asap_sys
start_asap_sys
```

7. Verify that the changes you made took effect.

   For example:

   - If you added or deleted a server, use the **status** command to verify that the server was added or deleted.
   - If you added a configuration parameter, check the **ASAP.cfg** file to make sure it was added.

Transforming ASAP Database Configurations or Service Models into XML

The DB to SACT XML Transformation Tool enables you to convert database entries to an XML file that complies to the ASAP **ActivationConfig** schema. This tool has been designed for ASAP users who created their server configuration using stored procedures and want to start using XML configuration files for this purpose.

The transformation tool translates data from the following tables in the CONTROL and SARM databases:

- CONTROL
  - tbl_appl_proc
  - tbl_component
  - tbl_listeners
- SARM
  - tbl_nep
  - tbl_resource_pool
  - tbl_host_clli
  - tbl_ne_config
  - tbl_clli_route
  - tbl_comm_param
  - tbl_asap_srp

> **Note:** The DB to SACT XML Transformation Tool does not display **ASAP.cfg** attributes.

To start the transformation tool, enterthe following command:

```
export_tool.sh [-m [-p ctrl_pswd]] -t config [out_xml_1 [out_xml_2]] [-h]
```

where:

- **-m:** Modifies *ASAP_Home*/**xml/xslt/dbinfo_*.xsl** to replace connection strings with appropriate values. If you do not specify -m, -p *ctrl_pswd* has no effect. You only have to use -m the first time you use the **export_tool.sh** script. This automatically updates the **dbinfo_sarm.xsl** and **dbinfo_ctrl.xsl** in the *ASAP_home*/**xml/xslt** directory
- **-p**: The control password. It obtains ASAP security information for the -m modification.
- *ctrl_pswd*: The control sever password.

- **-t service** or **config**: Identifies whether to transform service (service model) data or ASAP configuration data (activation configuration). Service is default type
- *out_xml_1*: The default is **transOut1_config.xml** in the current directory.
- *out_xml_2*: The default is **transOut2_config.xml** in the current directory.
- **-h**: Shows help information and then terminates.

The database information should be specified in two style sheet files, **dbinfo_ctrl.xsl** and **dbinfo_sarm.xsl**, which are imported by the **DumpDB_config.xsl** style sheet. These style sheet files are in the *ASAP_home*/**xml/xslt** directory. These two style sheets represent the two databases: CONTROL (which contains server data) and SARM (which contains network data).

DumpDB_config.xsl selects all entries in the listed tables and generates an internally formatted XML document. Then, the tool applies the style sheet **Formalize_config.xsl** to the raw XML file so that it conforms to the resource configuration schema.

3

About the Control and Daemon Servers

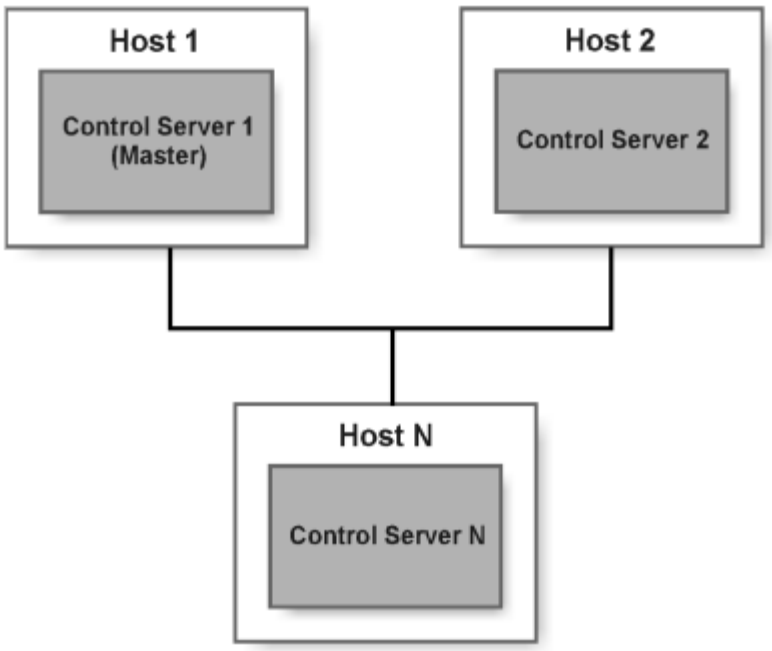
This chapter provides information about the Control and Daemon servers and configuration parameters associated with these servers.

About Control Servers and Fork Agents

The Control server is an application server that controls all other Oracle Communications ASAP applications and manages system startup and shutdown. For more information about starting up ASAP, see *ASAP System Administrator's Guide*. In a distributed environment, a Control server is required on every ASAP machine to start and shut down ASAP processes. One of the Control servers must be configured as the master.

Figure 3–1 illustrates the control and master server configuration in a distributed ASAP environment.

Figure 3–1 Control and Master Server Configuration



ASAP consists of application processes running in the background as servers and clients. Application processes can be configured using the Service Activation Configuration Tool (SACT) (see "Configuring ASAP Servers"). ASAP uses each service request processor (SRP), service activation request manager (SARM), network element processors (NEPs), and Admin processes as an application server, and they are configured to run as part of the system.

The Control server starts these applications by first verifying that the UNIX program executable file for the application server is located in the *ASAP_home*/**programs** directory and is executable. If it is executable, the Control server then instructs the fork agent process to spawn a child process, which in turn overlays itself with the application executable. At this point, the application process starts and the fork agent returns details of the application back to the Control server. For more information about configuring the fork agent, see "Fork Agent Process Generation Configuration".

In addition to providing details about the application back to the Control server, the fork agent also helps the Control server manage alarms generated by the individual server processes. For more information see "Control Server Alarm Generation".

The Control server monitors every client and server application in the ASAP system. If an application terminates unexpectedly, the Control server generates the alarm for the relevant operational community and can also attempt to restart a terminated process. For information about alarm configurations, see the chapter about monitoring and managing ASAP in *ASAP System Administrator's Guide*. For more information about configuring Control server monitoring and setting process restart attempts, see "Control Server Database and File System Monitoring".

After the application processes have been identified based on your service requirements, determine the Host machine for each process. You can use a single machine or multiple machine configuration to run the ASAP system. Each host machine is then assigned a Control server to control and monitor the ASAP processes for the machine. For more information about installing ASAP on multiple machines, see *ASAP Installation Guide*.

Next, assign the application processes (servers and clients) to a particular machine, a Control server.

Table 3–1, shows an example of an ASAP system:

Table 3–1 Sample ASAP Configuration

ASAP Process	Host	Description
SARM	HOST1	SARM server for the system
NEPDMS, NEPAXE, NEP5ESS	HOST2	NEPs for the system on HOST2 where the ports and communications devices are installed
SRPTCPIP	HOST1	SRP Server that manages the TCP/IP interface with a host system
SRPLU62	HOST1	SRP Server that manages the LU62 interface with a host system
LU62SEND & LU62RECV	HOST1	LU62 communication clients that pass data between the Host system and SRP2. These LU62 clients must be started before SRP2 is started.

You may want to configure ASAP to execute one or more of its processes on a separate or remote server. This can be done to improve performance (for example, to distribute processing loads), or because certain downstream interfaces or networks are not accessible from the machine where ASAP is running.

The server that runs the master control server is called the local machine, host or server. The server where the remote server is running is called the remote machine, host or server.

About the ASAP Daemon Server

The ASAP daemon makes it possible for the WebLogic server to reside on a different machine than the one on which an ASAP instance resides. Consequently, the ASAP daemon manages I/O operations required by Oracle WebLogic Server against a remote ASAP instance.

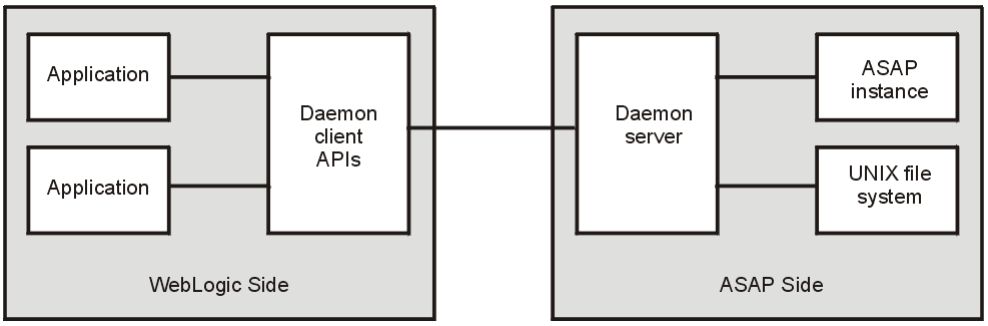
The ASAP daemon handles the file I/O operation requests issued from another UNIX user group or another machine. The ASAP daemon accommodates I/O operations between ASAP applications that use a WebLogic server, such as the SACT, Service Activation Deployment Tool (SADT), and Java SRP.

Specifically, the ASAP daemon supports WebLogic applications to do the following:

- File operations against an ASAP instance, as an alternative to Java's File class:
  - ASAP file read or write
  - ASAP file/directory state checking, such as isFile, canRead, canWrite, exists, length, or lastModified time
  - Other ASAP file manipulations, such as delete, renameTo
- Special commands performed on the ASAP side:
  - File copy from source to target
  - File move from source to target
  - File extract from a jar to a certain file
  - Execute a script
  - Other special commands

Figure 3–2 displays I/O operations for an ASAP instance on a UNIX file system, such as reading and writing a file, or executing a UNIX command or ASAP script. Some methods of accessing ASAP or the UNIX file system, such as RPCs and JDBCs, are independent of the daemon.

Figure 3–2 ASAP Daemon Configuration



4

Configuring Service Request Processors

This chapter provides information about the C and C++ service request processors (SRPs).

About Service Request Processor Servers

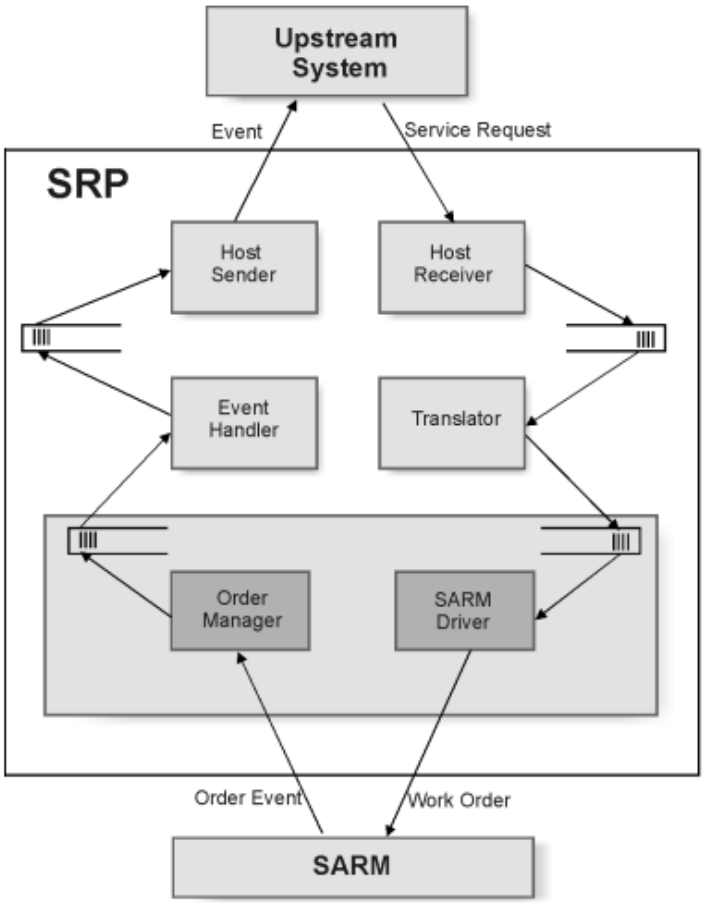
An SRP is an Oracle Communications ASAP server that controls service request reception and translation from an upstream system and service activation notifications from ASAP back to the upstream system.

With an SRP, you can:

- Generate and submit work orders
- Subscribe and manage work order events
- Query work order information

The SRP accepts and translates native service requests (work orders) into Common Service Description Layer (CSDL) commands. Using information from the work order, the SRP determines the parameters that are associated with the CSDL command and sends them to the Service Activation Request Manager (SARM) for provisioning.

Figure 4–1 SRP Processing Sequence



ASAP provides the following SRPs:

- C SRP Emulator: A generic C based SRP server/client located in the *ASAP_Home*/**programs/srp_emul** file that can submit work orders to and receive orders from the SARM. It is used for performance benchmarking, system testing, and the prototyping of new SRPs. See "Configuring a C SRP Emulator" for more information.
- C ++ Csol SRP Emulator: A sample C++ based SRP client located in the *ASAP_Home*/**programs/csolsrp_emul** file that can submit work orders to the SARM. Used to support C and C++ object-oriented technologies. See "Configuring the C++ Csol SRP Emulator" for more information.

ASAP provides the following two SRP APIs:

- C SRP API: Used to support C-based SRPs.
- C++ SRP API: Used to create C++ object-oriented-based SRPs. For more information about the C++ API, see "Using the C++ SRP API".

SRP Translation of Native SRP Work Orders to ASAP Work Orders

Generally, a work order or service request in its native format has one or more service elements. Each element identifies a service to be provisioned and an associated action such as add, remove, change, query, and so on.

Possible service elements include the following:

- Universal Service Order Code (USOC), which is used by many telephone companies.
- Service Offering (SOFF)

The SRP receives a work order or service request (SRP work order) from the originating system in its native format. It locates the service identifiers and associated data actions specified in the order, and then constructs the following SRP work order components:

- Native SRP work order header information, which can include details particular to the originating system that may or may not be required in the provisioning process.
- SRP work order parameters that are global to all services on the work order. For example, the main directory number (DN) or line equipment number (EN/LEN) may be specified here.
- One or more SRP work order service elements and associated parameters. Such service element specific parameters may override the Global SRP work order parameters on that particular service element.

The SRP determines these service elements and parameters in an SRP-specific manner that is highly coupled to the order originating system.

This SRP work order format may not model the service request format of particular customers. For example, some service requests model the services under the individual line, which adds an additional level to this SRP work order.

ASAP can contain several SRPs that are receiving work orders and service requests in different native formats. Each SRP performs custom translations for each work order or service request source to produce the same sets of CSDL commands and parameters representing the provisioning activity to be performed.

Each SRP must translate the same provisioning request received from different sources, such as work orders or service requests, into the same set of CSDL commands and parameters. This allows the SRP to insulate the rest of ASAP from the details of external systems.

Note: This section refers to an SRP that translates work orders into formats that the SRP can process using custom C or C++ scripts. The ASAP Service Request Translator (SRT) provides this functionality for XML messages and passes the information to the JSRP. For more information about SRT, see *ASAP Service Request Translator User's Guide*. SRT cannot be used with a C SRP or a C++ SRP.

Configuring a C SRP Emulator

You can configure C-based SRP emulator using the Service Activation Configuration Tool (SACT) or using stored procedures (see "Stored Procedures (Deprecated)"). This section describes the SRP configuration steps using the SACT.

The C SRP emulator makes use of the SRP, Interpreter, Server Application, and Common API libraries. SRP configuration information is located in static tables in the ASAP Control server database and the SARM database.

You can start the C SRP emulator using the **start_asap_sys** command. For more information see *ASAP System Administrator's Guide*.

Adding a C SRP Emulator

To add an C SRP emulator to the system:

1. Export the current database using **export_tool.sh** (see "Transforming ASAP Database Configurations or Service Models into XML").

   For example:

   ```
   export_tool.sh -m -p abc -t config
   ```

2. Open the **transOut2_config.xml** file using a text editor.

   For example:

   ```
   gedit transOut2_config.xml
   ```

3. Remove all elements except for the existing SRP.

   ```
   <activationConfig
   xmlns="http://www.metasolv.com/ServiceActivation/2003/ActivationConfig"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.metasolv.com/ServiceActivation/2003/ActivationCo
   nfig
   M:\hlam_view\ASAP_base\ASAP\jmx\xsd\ActivationConfig.xsd">

   <genericServer name="SRP_envid" xsi:type="SRPServerType">
            <description>SRP Emulator Server</description>
            <system>envid</system>
            <territory>envid</territory>
            <diagnosticFilename>SRP_envid.diag</diagnosticFilename>
            <diagnosticLevel>LOW_LEVEL</diagnosticLevel>
            <autoStart>true</autoStart>
            <controlServer>CTRLenvid</controlServer>
            <interfaceHostname>
                     <hostname>test.system.com</hostname>
            </interfaceHostname>
            <interfacePort>40012</interfacePort>
            <protocol>OPEN_SERVER</protocol>
            <program>srp_emul</program>
            <serverType>SRP</serverType>
            <SRPListenPort>40012</SRPListenPort>
   </genericServer>

   </activationConfig>
   ```

   where *envid* is the environment ID for your ASAP instance.

4. Modify the SRP element with new values where required.

   For example:

   ```
   <activationConfig
   xmlns="http://www.metasolv.com/ServiceActivation/2003/ActivationConfig"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.metasolv.com/ServiceActivation/2003/ActivationCo
   nfig
   M:\hlam_view\ASAP_base\ASAP\jmx\xsd\ActivationConfig.xsd">

   <genericServer name="SRP_2envid" xsi:type="SRPServerType">
   ```

```
        <description>SRP Emulator Server</description>
        <system>envid</system>
        <territory>envid</territory>
        <diagnosticFilename>SRP_2envid.diag</diagnosticFilename>
        <diagnosticLevel>LOW_LEVEL</diagnosticLevel>
        <autoStart>true</autoStart>
        <controlServer>CTRLenvid</controlServer>
        <interfaceHostname>
                <hostname>test.system.com</hostname>
        </interfaceHostname>
        <interfacePort>40052</interfacePort>
        <protocol>OPEN_SERVER</protocol>
        <program>srp_emul</program>
        <serverType>SRP</serverType>
        <SRPListenPort>40052</SRPListenPort>
</genericServer>

</activationConfig>
```

5. Add configuration parameters for the SRP element where required (see "SRP API Parameters"). For example:

```
<activationConfig
xmlns="http://www.metasolv.com/ServiceActivation/2003/ActivationConfig"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.metasolv.com/ServiceActivation/2003/ActivationCo
nfig
M:\hlam_view\ASAP_base\ASAP\jmx\xsd\ActivationConfig.xsd">

<genericServer name="SRP_2envid" xsi:type="SRPServerType">
        <description>SRP Emulator Server</description>
        <system>envid</system>
        <territory>envid</territory>
        <diagnosticFilename>SRP_2envid.diag</diagnosticFilename>
        <diagnosticLevel>LOW_LEVEL</diagnosticLevel>
        <autoStart>true</autoStart>
        <controlServer>CTRLenvid</controlServer>
        <interfaceHostname>
                <hostname>test.system.com</hostname>
        </interfaceHostname>
        <interfacePort>40052</interfacePort>
        <protocol>OPEN_SERVER</protocol>
        <program>srp_emul</program>
        <serverType>SRP</serverType>
        <SRPListenPort>40052</SRPListenPort>
</genericServer>

</activationConfig>
```

6. Save the file.
7. Update the file using **sactConfig** as described in "Running the SACT Scripts".

Deleting a C SRP Emulator

To delete a C SRP emulator:

1. Export the current database using **export_tool.sh** (see "Transforming ASAP Database Configurations or Service Models into XML").

   For example:

```
export_tool.sh -m -p abc -t config
```

2. Open the **transOut2_config.xml** file using a text editor.

   For example:

   ```
   gedit transOut2_config.xml
   ```

3. Remove all elements except for the SRP you want to delete.

   For example:

   ```
   <activationConfig
   xmlns="http://www.metasolv.com/ServiceActivation/2003/ActivationConfig"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.metasolv.com/ServiceActivation/2003/ActivationCo
   nfig
   M:\hlam_view\ASAP_base\ASAP\jmx\xsd\ActivationConfig.xsd">

   <genericServer name="SRP_envid" xsi:type="SRPServerType">
            <description>SRP Emulator Server</description>
            <system>envid</system>
            <territory>envid</territory>
            <diagnosticFilename>SRP_envid.diag</diagnosticFilename>
            <diagnosticLevel>LOW_LEVEL</diagnosticLevel>
            <autoStart>true</autoStart>
            <controlServer>CTRLenvid</controlServer>
            <interfaceHostname>
                     <hostname>test.system.com</hostname>
            </interfaceHostname>
            <interfacePort>40012</interfacePort>
            <protocol>OPEN_SERVER</protocol>
            <program>srp_emul</program>
            <serverType>SRP</serverType>
            <SRPListenPort>40012</SRPListenPort>
   </genericServer>

   </activationConfig>
   ```

   where *envid* is the environment ID for your ASAP instance.

4. Save the file.
5. Update the file using **sactUnconfig** as described in "Running the SACT Scripts".

Adding Configuration Parameters to a C SRP Emulator

To add configuration parameters to a C SRP emulator:

1. Export the current database using the **export_tool.sh** (see "Transforming ASAP Database Configurations or Service Models into XML").

   For example:

   ```
   export_tool.sh -m -p abc -t config
   ```

2. Open the **transOut2_config.xml** file using a text editor.

   For example:

   ```
   gedit transOut2_config.xml
   ```

3. Remove all elements except for the SRP you want to add configuration parameters to. For example:

```
<activationConfig
xmlns="http://www.metasolv.com/ServiceActivation/2003/ActivationConfig"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.metasolv.com/ServiceActivation/2003/ActivationCo
nfig
M:\hlam_view\ASAP_base\ASAP\jmx\xsd\ActivationConfig.xsd">

<genericServer name="SRP_envid" xsi:type="SRPServerType">
         <description>SRP Emulator Server</description>
         <system>envid</system>
         <territory>envid</territory>
         <diagnosticFilename>SRP_envid.diag</diagnosticFilename>
         <diagnosticLevel>LOW_LEVEL</diagnosticLevel>
         <autoStart>true</autoStart>
         <controlServer>CTRLenvid</controlServer>
         <interfaceHostname>
                  <hostname>test.system.com</hostname>
         </interfaceHostname>
         <interfacePort>40012</interfacePort>
         <protocol>OPEN_SERVER</protocol>
         <program>srp_emul</program>
         <serverType>SRP</serverType>
         <SRPListenPort>40012</SRPListenPort>
</genericServer>

</activationConfig>
```

where *envid* is the environment ID for your ASAP instance.

4. Add the configuration parameters. For example:

```
<activationConfig
xmlns="http://www.metasolv.com/ServiceActivation/2003/ActivationConfig"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.metasolv.com/ServiceActivation/2003/ActivationCo
nfig
M:\hlam_view\ASAP_base\ASAP\jmx\xsd\ActivationConfig.xsd">

<genericServer name="SRP_envid" xsi:type="SRPServerType">
         <description>SRP Emulator Server</description>
         <system>envid</system>
         <territory>envid</territory>
         <diagnosticFilename>SRP_envid.diag</diagnosticFilename>
         <diagnosticLevel>LOW_LEVEL</diagnosticLevel>
         <configurationParameters>
            <configurationParameter xsi:type="WO_MGR_DELAY">
                  <value>5</value>
            </configurationParameter>
         </configurationParameters>
         <autoStart>true</autoStart>
         <controlServer>CTRLenvid</controlServer>
         <interfaceHostname>
                  <hostname>test.system.com</hostname>
         </interfaceHostname>
         <interfacePort>40012</interfacePort>
         <protocol>OPEN_SERVER</protocol>
         <program>srp_emul</program>
         <serverType>SRP</serverType>
         <SRPListenPort>40012</SRPListenPort>
</genericServer>
```

```
</activationConfig>
```

5. Save the file.
6. Update the file using **sactConfigNR** as described in "Running the SACT Scripts".

Configuring the C++ Csol SRP Emulator

The C++ Csol SRP emulator is a standalone program designed to use the C++ SRP libraries to create and submit work orders to the SARM. The emulators retrieve work order information and receive events from the SARM. The events are collected from the SARM and are logged to the SRP database and the diagnostic file.

The C++ SRP emulators use the C++ SRP libraries.

For the SARM to communicate with the C++ SRP emulator, you must configure the emulator in **tbl_asap_srp** in the SARM database. Enter the following information:

- Set the **srp_conn_type** field to **O** so that the SARM can make Open Client connections to the emulator.
- Set the **srp_host_name** field to the name of the UNIX host that the emulator is running on.
- Set the **srp_host_port** field to the socket port number that the emulator and SARM can communicate with.

For more information, see "tbl_asap_srp" in *ASAP Developer's Guide*.

Starting the C++ Csol SRP Emulator

To start the C++ Csol SRP emulator:

1. Source the *ASAP_home*/**Environment_Profile**:

```
. ./Environment_Profile
```

2. Start ASAP if it is not already running.

```
start_asap_sys
```

3. Enter the following command:

```
csolsrp_emul svr_name ctrl_password -c ctrl_svr_name -l diag_level [-f diag_file]
```

where:

- *svr_name*: The name of the SRP server.
- *ctrl_password*: The control server password.
- *ctrl_svr_name*: The name of the control server.
- *diag_level*: The diagnostics level. For more information on diagnostic levels, see *ASAP System Administrator's Guide*.
- *diag_file*: The name of the diagnostics file.

For example:

```
csolsrp_emul $SRP abc -c $CTRL -l KERN -f CSOL
```

Using the C++ SRP API

The C++ SRP API provides you with an interface with the SARM. It allows C++ SRPs to generate, manipulate, and submit ASAP work orders, handle events, and access SARM databases in a multi threaded environment.

The C++ SRP receives requests from external systems and translates them into ASAP work orders using the C++ SRP API. The C++ SRP submits the work orders to the SARM. The SARM sends a confirmation to the C++ SRP for each work order it receives and then generates events and forwards them to the C++ SRP to identify the progress of the work order.

The C++ SRP uses native threads to support symmetric multiprocessing and to provide maximum scalability of ASAP across multiple CPUs.

- ASAP libraries:
  - Common Object-Oriented Library (liboo_asc) – Provides common utilities and database access
  - Thread Framework Library (libthreadfw) – Provides the underlying thread framework
- Third party libraries:
  - Open Client (Sybase)

For information on ASAP libraries, see the *ASAP Developer's Guide*.

After the work order is created, you can submit it to ASAP through the C++ SRP. The C++ SRP provides interfaces for you to process events and access ASAP orders. It also insulates the user from most exceptional condition handling.

5

Configuring Java Service Request Processors and Web Services

This chapter describes the Java service request processor (JSRP) and a Web Service interface to the JSRP.

About Java Service Request Processor Servers

A JSRP is an Oracle Communications ASAP server that controls service request reception and translation from an upstream system and service activation notifications from ASAP back to the upstream system.

With an JSRP you can:

- Generate and submit work orders
- Subscribe and manage work order events
- Query work order information

The JSRP accepts and translates native service requests (work orders) into Common Service Description Layer (CSDL) commands. Using information from the work order, the SRP determines the parameters that are associated with the CSDL command and sends them to the Service Activation Request Manager (SARM) for provisioning.

You can implement the following types of JSRPs, depending on the upstream technology being employed.

- JSRP and Web Service: Provides a standardized OSS through Java interface and Web Service interface into ASAP's provisioning functionality within a WebLogic server instance.

  For more information about the JSRP and Web Service, see "About JSRP, Web Service, and OCA SRP Components".

- Custom Java SRP: You can create a custom JSRP not based on a WebLogic server instance using the information provided in *ASAP Java Online Reference*. For more information about configuring a custom JSRP, see "Configuring a Custom Java SRP client".

- OCA SRP: Used internally by the ASAP Order Control Application (OCA) client. OCA SRP co-exists with Java SRP in WebLogic and shares some of the Java-based APIs used in JSRP.

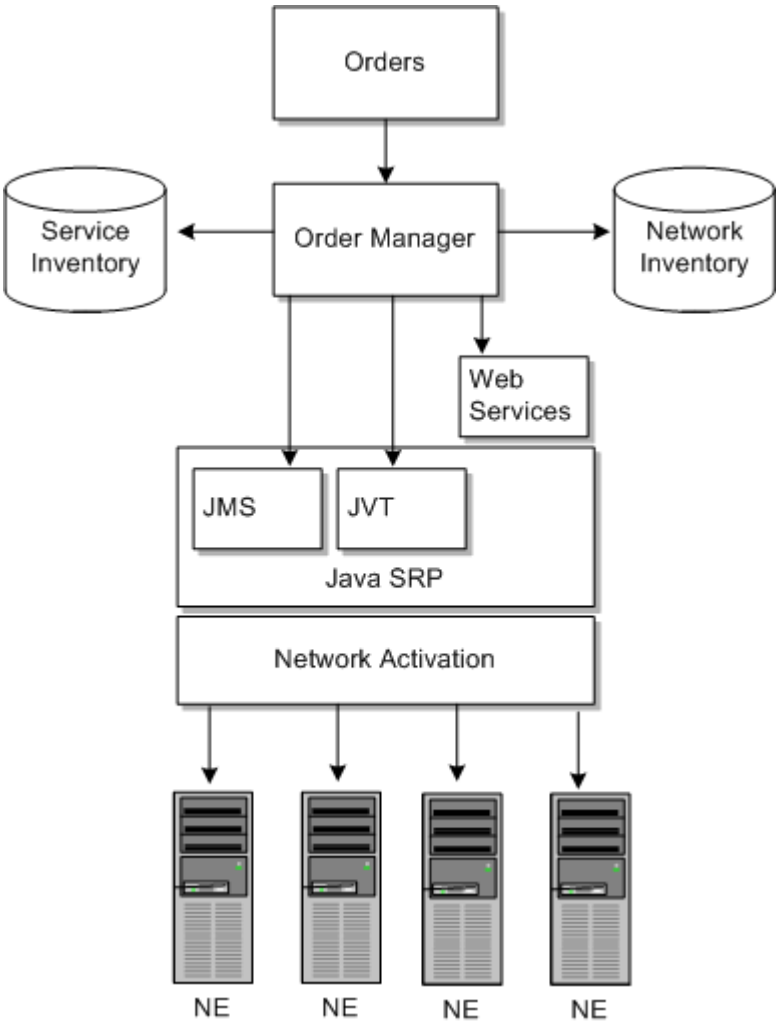
  For more information on the OCA SRP, see "Configuring a OCA SRP".

About JSRP, Web Service, and OCA SRP Components

The JSRP is an upstream component into ASAP that provides a standardized OSS through Java (OSSJ) interface into the ASAP provisioning functionality and also a Web Services API implementation.

Figure 5–1 displays a system-level view of the JSRP. The JSRP receives requests from an upstream system, potentially an order manager: for example, Oracle Communications Order and Service Management (OSM) or a customer care system. Requests are translated into ASAP commands, which are then submitted to the SARM for processing.

Figure 5–1 JSRP, System Level View



The JSRP provides two types of interfaces: XML over JMS and Java value types (JVT) over RMI/IIOP. The JSRP is fully compliant with the OSSJ Service Activation API version 1.0.

In addition to the JSRP, Web Service provides an interface for SOAP/JMS messages. The Web Service implementation translates these SOAP/JMS messages into JVT work orders and sends them to the JSRP's JVT interface.

For more information about these interfaces, see "About the JSRP Server and Web Service Interfaces".

The JSRP and Web Service implementation are situated between the SARM and clients designed to send messages to either Web Service, JVT, or JMS interfaces. These clients are located on a remote application, such as OSM.

Oracle recommends that you create a Store and Forward (SAF) agent for requests and responses to and from the JMS or Web Service interfaces and the JMS or Web Service clients and a JMS bridge for work order state changes from ASAP to the JMS or Web Service clients. For more information about SAF and the JMS bridge, see "About Connecting JSRP JMS and Web Service Interfaces to a Remote Application".

The JVT interface does not require SAF or the JMS bridge.

The following list summarizes the components required for the two JSRP interfaces and the Web Service interface to a remote application (for example, OSM):

- JSRP JMS Interface to and from SAF and JMS Bridge to and from JSRP JMS Client (OSM)
- ASAP Web Service interface to and from SAF and JMS Bridge to and from JSRP Web Service Client (OSM)
- JSRP JVT Interface (RMI) to and from JSRP JVT Client (OSM)

For more information about creating a JVT or JMS Client, see the discussion about developing Java SRP client applications in *ASAP Java Online Reference* available from the Oracle software delivery Web site along with the ASAP installation files. *ASAP Java Online Reference* also provides a Java library for the JVT interface and lists the XML schemas used for constructing XML-based work order messages.

For more information about the Web Service API and the Web Services Description Language (WSDL) file, see *ASAP Developer's Guide*.

For information about sample JVT and JMS clients and source files that you can use to test the Java SRP, go to *ASAP_Home*/**samples/jsrp/README.txt**.

The JMS, JVT, and Web Services JMS modules, connection pools, and JDBC stores are automatically configured when you install ASAP. For information about these components, see "About the JSRP Server and Web Service Interfaces".

The configuration of the Java SRP deployment descriptors are automatically completed when you install ASAP. If you must fine-tune the deployment descriptor attributes of the Java SRP, see "Modifying JSRP Parameters (Deployment Descriptors) in WebLogic".

The implementation of the Java SRP and Web Services API does not affect any provisioning logic in the existing ASAP product. Support for OSSJ and Web Services is implemented as translations to and from existing types and structures in ASAP, to OSSJ and Web Services conferment types and structures.

As with the other types of SRPs, the Java SRP and Web Services API allows the client to generate and submit work orders, manage work order events, and retrieve work order information.

Note: The implementation of the Java SRP and Web Services API does not affect any provisioning logic in the existing product and is consistent with other upstream interfaces (C++ SRP, OCA SRP).

About the JSRP Server and Web Service Interfaces

The configuration of the WebLogic Server involves defining server resources such as connection pools, JDBC stores, and JMS modules such as connection factories, queues,

and topics that are required for the Java SRP and the Web Services implementation. When you install ASAP, the Java SRP and Web Service configuration are automatically created. You can verify the Java SRP and Web Service configuration using the WebLogic management console.

Table 5–1 lists and describes the JMS modules used by the JSRP interfaces and Web Service.

Table 5–1 Java SRP, OCA SRP, and Web Services JMS Modules

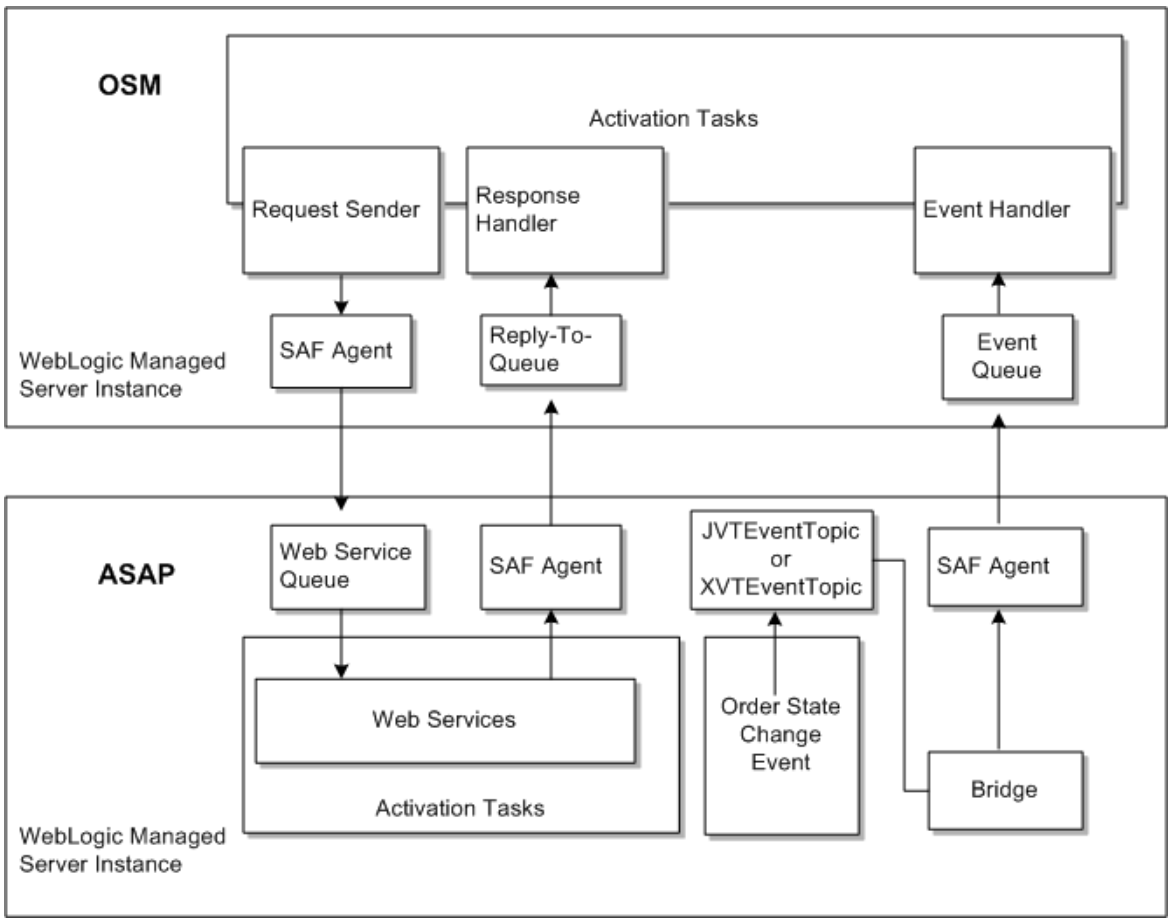
JMS Module Name	Description
CMWSAdapterQueue	The publisher to the CMSW queue is Design Studio. It publishes ASAP Cartridges. The subscriber to this queue is the CMSW Adapter located in WebLogic.
ErrorQueue	The publisher for the ErrorQueue is WebLogic. If any error happens within any WebLogic queue, this message is published to the ErrorQueue. This is an internal WebLogic queue and does not effect ASAP components. For more information about this queue, see the WebLogic Server documentation.
JVTEventTopic	The publisher for the JVTEventTopic is the JSRP. Work order events get published to this topic. Subscribers are JSRP and Web Service clients that wish to receive notification of JSPR Events published to the JVTEventTopic in the form of Java object responses.
MessageQueue	The JSRP client (for example OSM) publishes JMS requests to this queue in the form or JSRP work orders. The subscriber to this queue is the JSRP server.
OCAEventTopic	The publisher for the OCAEventTopic is the OCA SRP. Work order events get published to this topic. Subscribers are OCA clients that wish to receive notification of OCA events published to the OCAEventTopic in the form of Java object responses.
OCAMessageQueue	The OCA client publishes JMS requests to this queue in the form or OCA work orders. The subscriber to this queue is the OCAJ server.
TopicConnectionFactory	The JSRP server, and OCA server uses this TopicConnectionFactory to create a topic publisher that sends ASAP events to XVTEventTopic, JVTEventTopic, and OCAEventTopic.
WebServiceQueue	The Web Services client (for example OSM) publishes SOAP requests to this queue in the form or Web Service work orders. The subscriber to this queue is the Web Services server. It decomposes the SOAP Work order, creates a JVT work order, and submits the work order via the JSRP RMI interface. This interface is shared with JVT.
XVTEventTopic	The publisher for the XVTEventTopic is the JSRP. Work order events get published to this topic. Subscribers are JSRP clients that wish to receive notification of JSPR Events published to the XVTEventTopic in the form of xml responses.

About Connecting JSRP JMS and Web Service Interfaces to a Remote Application

Oracle recommends that you create a SAF agent and JMS bridge between the ASAP WebLogic server and the remote application WebLogic server. Oracle recommends this SAF agent and JMS bridge for the JMS or Web Service interfaces to ensure reliable communication.

Figure 5–2 illustrates an example SAF and JMS bridge configuration between the Web Service interface on ASAP and a Web Service client on a remote application, in this case, OSM.

Figure 5–2 SAF and JMS Bridge Between OSM and ASAP



In this example, an OSM SAF agent sends requests to the ASAP request queue, and ASAP returns responses through the ASAP SAF agent to the OSM reply-to queue. In addition, ASAP sends work order state changes from the JSRP XVTEventTopic through a JMS bridge with a SAF agent to the OSM event queue.

For detailed instructions for creating SAF and JMS bridges between ASAP and OSM, see *Configuring WebLogic Resources for OSM Integration With ASAP And UIM On Different Domains (Doc ID 1431235.1)* knowledge article in Oracle Support, `https://support.oracle.com`. This article is applicable to any remote application that uses a WebLogic JMS server to send and receive Web Service or JMS messages.

Modifying JSRP Parameters (Deployment Descriptors) in WebLogic

This section provides information about how to modify JSRP parameters or deployment descriptors in WebLogic Server. You can update the JSRP parameters described in Table 5–2 by using WebLogic Workshop (Eclipse component) or by editing the **ejb-jar.xml** file manually.

Table 5–2 JSRP Configuration Parameters (Deployment Descriptors)

Parameter Name	Type	Description
SERVICE_TYPE	java.lang. String	Do not change this parameter. This parameter differentiates between JSRP and OCA instances, and can be either JSRP or OCA.
UnidFormat	java.lang. String	This parameter defines the work order (WO) format template. If the JSRP does not provide the WO_ID, the WO_ID is generated based on the format template.
APPLICATION_DN	java.lang. String	Do not change this parameter.
OCA_ATTEMPTS	java.lang. Integer	This parameter configures the number of times that the OCA Server tries to submit a WO to SARM.
VALIDATION_ENABLED	java.lang. Boolean	This parameter, when enabled, validates the JSRP WO requests against the XML Schema. Enabling this attribute can provide additional security. For more information, see "Configuring Validation of Received Data".
BACKWARD_COMPATIBLE_ SCHEMA	java.lang. Boolean	It's flag to indicate if it's backward compatible with previous format.
INCLUDE_SERVICE_ACTION_ DETAIL	java.lang. Boolean	Includes service action details in events.

USE_ORIGINAL_INSTANCE_ NUMBER	java.lang. Boolean	This indicate in even if set to true, then the JSRP uses the original instance number from the primaryKey and type contained in a message received from the Service Request Translator (SRT).
		`if (useOriginalServiceInstanceNumbers()) {`
		`  typeElement.appendChild( pKeyText );`
		`  idElement.appendChild( typeText );`
		`} else {`
		`  typeElement.appendChild( typeText );`
		`  idElement.appendChild( pKeyText );`
		`}`

NO_LISTENING_THREAD	java.lang. Boolean	Disables the SARM listening port for the JSRP. Set this parameter to false.
NO_WAIT_START_ORDER	java.lang. Boolean	Disables the JSRP functionality that waits for a start order messages from the SARM. Set this parameter to false.

To edit the JSRP parameter file manually:

1. Go to *WebLogic_domain*/**servers/***WebLogic_server*/**upload/asap***ENV_ID*/**app** (where the *WebLogic_domain* is the installation directory for your WebLogic Server domain, *WebLogic_server* is the name of your WebLogic Server domain, and *ENV_ID* is the ASAP environment ID.
2. Do the following:

```
jar xvf asapENV_ID.ear srp.jar
jar xvf srp.jar META-INF/ejb-jar.xml
```

3. Edit **ejb-jar.xml** to modify JSRP parameters. For example:

```
    <env-entry>
        <env-entry-name>APPLICATION_DN</env-entry-name>
        <env-entry-type>java.lang.String</env-entry-type>

<env-entry-value>System/dc2/ApplicationType/ServiceActivation/Application/1-0;7
-2;ASAP/Comp/</env-entry-value>
```

```
      </env-entry>
      <env-entry>
        <env-entry-name>SRP</env-entry-name>
        <env-entry-type>java.lang.String</env-entry-type>
        <env-entry-value>JSRPdc2</env-entry-value>
      </env-entry>
      <env-entry>
        <env-entry-name>SRP_BACKWARD_COMPATIBLE</env-entry-name>
        <env-entry-type>java.lang.Boolean</env-entry-type>
        <env-entry-value>true</env-entry-value>
      </env-entry>
      <env-entry>
        <env-entry-name>DEFAULT_JMS_REPLY_TO</env-entry-name>
        <env-entry-type>java.lang.String</env-entry-type>

<env-entry-value>System/dc2/ApplicationType/ServiceActivation/Application/1-0;7
-2;ASAP/Comp/XVTEventTopic</env-entry-value>
      </env-entry>
      <env-entry>
        <env-entry-name>VALIDATION_ENABLED</env-entry-name>
        <env-entry-type>java.lang.Boolean</env-entry-type>
        <env-entry-value>false</env-entry-value>
      </env-entry>
      <env-entry>
        <env-entry-name>BACKWARD_COMPATIBLE_SCHEMA</env-entry-name>
        <env-entry-type>java.lang.Boolean</env-entry-type>
        <env-entry-value>false</env-entry-value>
      </env-entry>
      <env-entry>
        <env-entry-name>INCLUDE_SERVICE_ACTION_DETAIL</env-entry-name>
        <env-entry-type>java.lang.Boolean</env-entry-type>
        <env-entry-value>false</env-entry-value>
      </env-entry>
      <env-entry>
        <env-entry-name>USE_ORIGINAL_INSTANCE_NUMBER</env-entry-name>
        <env-entry-type>java.lang.Boolean</env-entry-type>
        <env-entry-value>false</env-entry-value>
      </env-entry>
      <env-entry>
        <env-entry-name>NO_LISTENING_THREAD</env-entry-name>
        <env-entry-type>java.lang.Boolean</env-entry-type>
        <!-- changed to false to activate ASAP behavour -->
        <env-entry-value>false</env-entry-value>
      </env-entry>
     ...
```

4. Do the following:

```
jar uvf srp.jar META-INF/ejb-jar.xml
jar uvf asap<ENV_ID>.ear srp.jar
```

5. Redeploy the **asap***ENV_ID***.ear** file.

Configuring Validation of Received Data

You can configure the JSRP to validate any ASAP RPC parameters of "char" type if VALIDATION_ENABLED in the JSRP deployment descriptor is set to "true". If the maximum lengths are exceeded, the JSRP throws an exception. For example, in the XML/JMS interface, if specifying <mslv-sa:organizationUnit>POTS12345678</mslv-sa:organizationUnit> in

createOrderByValueRequest (which exceeds the maximum length), the following exception is thrown:

```
<sa:createOrderByValueException
xmlns:co="http://java.sun.com/products/oss/xml/Common"
xmlns:mslv-sa="http://www.metasolv.com/oss/ServiceActivation/2003"
xmlns:sa="http://java.sun.com/products/oss/xml/ServiceActivation"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<sa:illegalAttributeValueException>
<co:message>String attribute value[POTS12345678] exceeding its maximum
length[8]</co:message>
</sa:illegalAttributeValueException>
</sa:createOrderByValueException>
```

Exceptions also will be thrown in the JVT interface.

Setting Log Levels

You can configure the Java SRP to filter log messages that are being written into the WebLogic log file by changing the level of message severity.

There are three levels of severity for the Java SRP:

- info
- warning
- error

By default, the asapSeverityLevel is error.

See *ASAP System Administrator's Guide* for details on configuring the logging levels.

Uninstalling the Java SRP

For more information on uninstalling the Java SRP, see the procedure for uninstalling WebLogic in *ASAP Installation Guide*.

Configuring a Custom Java SRP client

You can use the Control server to manage (start, stop, auto-restart) a custom JSRP client that is not based in a WebLogic server like any other ASAP component. Perform the following steps to configure a Java SRP Client.

1. Write a custom Java SRP client by extending the **ASCAppl** class (see the sample code below).
2. Implement **applMain**() method, which is the main function of the Java SRP client (see "Sample Code for a Custom JSRP Client").
3. Implement the **initialize**() method instead of providing a specialized constructor, if necessary (see "Sample Code for a Custom JSRP Client").
4. Using SACT, populate the Control database and ASAP Environment (**tbl_appl_proc** and **tbl_component**) for the new Java SRP client.

```
<activationConfig
xmlns="http://www.metasolv.com/ServiceActivation/2003/ActivationConfig"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.metasolv.com/ServiceActivation/2003/ActivationCo
nfig
M:\hlam_view\ASAP_base\ASAP\jmx\xsd\ActivationConfig.xsd">
```

```
<nonControlManagedServer name="JSRPenvid" xsi:type="JSRPServerType">
        <description>Java SRP</description>
        <system>envid</system>
        <territory>envid</territory>
        <diagnosticFilename>JSRPenvid.diag</diagnosticFilename>
        <diagnosticLevel>LOW_LEVEL</diagnosticLevel>
        <SARMListenPort>40070</SARMListenPort>
        <SRPListenHostname>
                <IPaddress>10.156.53.74</IPaddress>
        </SRPListenHostname>
        <SRPListenPort>40077</SRPListenPort>
</nonControlManagedServer>

</activationConfig>
```

5. Write the script to run the Java SRP client (see "Sample Script to Run the Custom JSRP Client").

Sample Code for a Custom JSRP Client

```
package com.metasolv.activation;

import java.util.*;
import java.io.*;
import java.sql.SQLException;
import java.text.*;
import com.mslv.activation.server.*;

public class SampleJSrpClient extends ASCAppl
{
 public void initialize()
 {
        super.initialize();
        // Do additional initialization
……..
 }

..
.

 public void applMain()
 {
    for(;;) {
        try {
            // Do actual process for JSRP client
……..……
        } catch(Exception ex ) {
        }
    }
  }

}
```

Sample Script to Run the Custom JSRP Client

```
#!/bin/ksh

#
```

```
# =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
#
# (c) 2008 Oracle.
# The CLASSPATH variable is used by the primordial class loader in the java
virtual
```

Configuring a OCA SRP

The OCA SRP is similar to Java SRP and resides in WebLogic. The OCA SRP shares some of the Java-based APIs used in JSRP. The OCA system includes a GUI based client for configuring the OCA SRP and integrates the work order related main functionality in this single program.

Setting OCA SRP Configuration Parameters

In addition to the parameters contained in the previous table, the following parameters in *ASAP_Home*/**config/OCA.cfg** were used by the deprecated CORBA SRP emulator. Now, they are required by the new OCA SRP.

Table 5–3 describes the configuration parameters that must be set in the *ASAP_Home*/**config/OCA.cfg** configuration file. For more information about configuring the OCA client, see *ASAP Order Control Application User's Guide*.

Table 5–3 Session Configuration Parameters

Configuration Variable	Required	Default Value	Description
HOST	Yes	localhost	Name of host machine where WebLogic is installed. The session associated OCA SRP server is running as an application of this WebLogic.
PORT	Yes	2345	Port number on which WebLogic is listening. The session associated OCA SRP server is running as an application of this WebLogic.
ENV_ID	Yes	S123	Environment ID of ASAP instance. The session associated OCA SRP server is a component of this ASAP system.
SYSTEM_ NAME	Yes	OCA_S123	Name of the OCA Server to which the OCA client connects. Used to support multiple ASAP systems from a single client.

Table 5–4 Global Configuration Parameters

Configuration Variable	Required	Default Value	Description
SYSTEM_ NAME	Yes	OCA_S255	Name of the ASAP system to which the OCA client connects. Used to support multiple ASAP systems from a single client.
LANGCODE	No	USA	Language code for the language in which translatable information is returned to and displayed by the client. If applied, all translatable fields are shown in the OCA client native language, as specified by the LANGUAGE variable. A language other than the default language (USA) must be configured in ASAP.

Table 5–4 (Cont.) Global Configuration Parameters

Configuration Variable	Required	Default Value	Description
IDLETIMEOUT	No	0	Maximum idle time, in minutes, after which the OCA SRP server terminates the OCA client connection. Set to zero (0) to disable this feature.
DATE_ FORMAT	-	-	dd/MM/yyyy or MM/dd/yyyy
TIME_ FORMAT	-	-	AM_PM (12-hour clock with an AM or PM designation) or NON_AM_PM (a 24-hour clock).
OCA_EVENTS_ SUPPORTED	Yes	True	The flag should be false when connecting to servers on the HP10.20 platform. Events are not supported for this platform.
MAXW_OPEN_ DETAILS	No	3	The number of detail windows a user can have open at one time.

6

Managing the Service Activation Request Manager

This chapter describes the Service Request Activation Manager (SARM).

About Managing Service Activation Request Manager Servers

The SARM performs the following tasks:

- Determines and coordinates requests between the various Service Request Processor (SRPs) and Network Element Processors (NEPs) in the ASAP system. (SRP)
- Manages connections to network elements (NEs)
- Manages work order priority and load balancing to NEs
- Translates Common Service Description Layer (CSDLs) received from the SRP, into Atomic Service Description Layer (ASDL) commands that it sends to the NEP
- Determines routing for each ASDL and transmits it to the correct NEP
- Determines any ASDL spawning or conditional logic
- Publishes work order events back to the originating SRPs
- Stores ASDL to Action processor to Java or State Table command implementation configuration
- Initiates work order rollback
- Processes work order data in the order in which they are defined in the service definition (cartridge)
- CSDL Level within a word order, used to rearrange the order in which CSDLs are processed

The SARM manages all of these tasks based on the configuration data contained in an ASAP cartridge. These cartridges define required work order parameters, that the SRP turns into a CSDL command and sends to the SARM. The SARM processes the CSDL based on the information contained in the cartridge.

SARM to SRP Event Notification

The SARM to SRP functional interface consists of work order event notifications transmitted by the SARM to the SRP as the work order is being provisioned. When it receives these notification events, the SRP processes each event in the manner

appropriate to the SRP for that particular event. The SRP can choose to ignore some of these event notifications.

SRP Work Order Event Management

As the SRP transmits the ASAP work order to the SARM for provisioning, the SARM returns several notification events to the SRP related to the provisioning activity. The SRP passes these events to the originating system in the native format of that system. This provides the originating system with extensive feedback about the order provisioning progress.

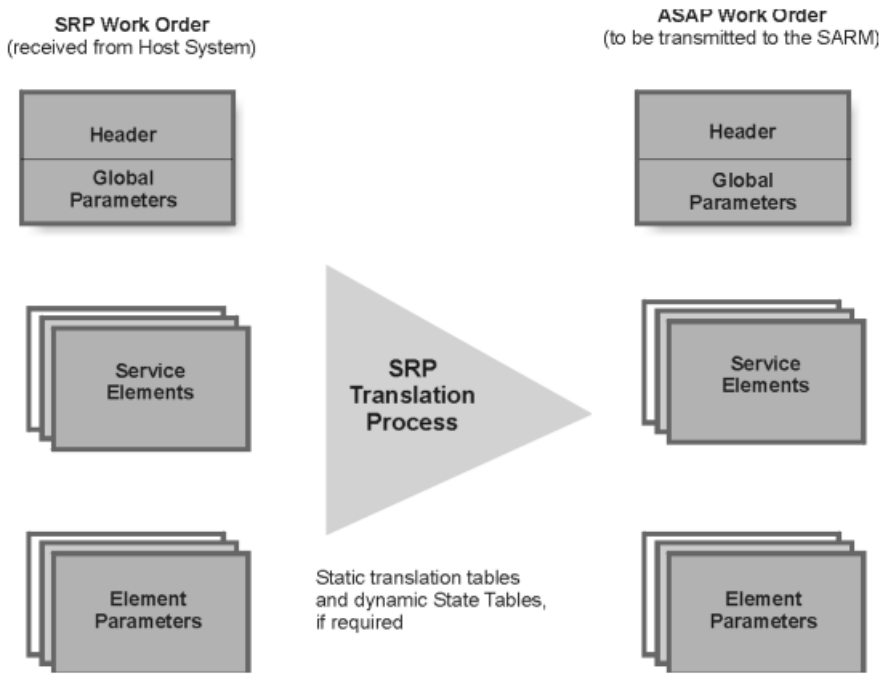
In addition to the notification events that the SRP passes back to the originating system, the SRP also transmits the following SRP originated events:

- **SRP WO Acknowledgement Event** – Acknowledges the successful receipt of the work order.
- **SRP WO Translation Error Event** – Signifies that the SRP was unable to translate the SRP work order successfully. In most cases, the work order is transmitted to the SARM with a Translation Error status, and then the SARM holds it and waits for manual intervention.
- **SRP WO Rejection Event** – Notifies the originating system that either the SRP or the SARM rejected this copy of the work order. The causes of this event vary from customer to customer, but it generally results from an attempt to update an existing order in ASAP when it is either In-Progress or Completed.

The SRP, which is generally under your control, can define additional events back to the originating system. However, you can set up the system to exclude certain SARM-originated events from being transmitted back to the originating system.

Using this data structure, the SRP performs a translation from the native SRP work order format to the ASAP work order format. Figure 6–1 describes this translation.

Figure 6–1 SRP Work Order Translation



The various ASAP work order components created by this process are outlined in the following section.

NEP to SARM Event Notifications

The SARM's main function regarding NE queue management is the maintenance of NE status information. The SARM maintains status information for each NE in ASAP. This information includes:

- The technology and software load.
- The current NE State (Down, Connecting, Available, Maintenance, Port Failure).
- The average processing time for an ASDL to an NE.
- The number of connections available to an NE.
- The number of pending ASDL requests to an NE (in the Pending queue).
- The number of ASDL requests currently in progress (in the In Progress queue).
- The number of ASDL requests waiting to be retried to an NE (in the Retry queue).

This information is available in real-time from the SARM and SARM database through an internal NE monitor table that reflects the present state of all NEs in the system.

For information on viewing the monitor table, see Appendix A, "asap_utils.".

Table 6–1 shows the notifications that the NEP sends to the SARM.

Table 6–1 NE Notifications

Notification	Description
NE Available	The NEP transmits this notification to inform the SARM that a particular NE is available. This prompts the SARM to begin transmitting ASDLs to the NEP.
Auxiliary Connection Failure	The NEP notifies the SARM that the auxiliary connection request could not be completed.
NE Port Failure	The NEP transmits this notification to the SARM to inform it that the connections to a particular NE are down. This can happen at any point during the provisioning process. The SARM marks its internal NE status as Port Failure, and then moves the ASDL commands from the In Progress queue to the Pending queue. The NEP periodically attempts to re-establish the primary connection to the NE.
Device Error	The NEP sends this notification to the SARM when a particular connection to the NE is down. The SARM moves the ASDL command that was being provisioned on that port from the In Progress queue to the Pending queue. If no other connection is available to the NE, the SARM marks its internal NE status as 'NE Unavailable Due to Port Failure', and then moves the ASDL commands from the In Progress queue to the Pending queue. The NEP periodically tries to reestablish the primary connection to the NE.

Table 6–1 (Cont.) NE Notifications

Notification	Description
NE Maintenance Mode	Upon receipt of this notification, the SARM sets its internal NE status to Maintenance and queues all ASDL requests to the NE in the Pending queue. The NEP periodically tries to establish a connection to the NE. Once the NEP reestablishes this connection, it transmits an NE Available notification to the SARM which triggers the SARM to resume provisioning ASDLs to the NE. If no other connection is available to the NE when the SARM receives Maintenance Mode notification, it sets the internal status of the NE to Maintenance and queues all ASDL requests to that NE in the Pending queue. Once the NEP re-establishes the connection to the NE, it transmits an NE Available notification to the SARM, which prompts the SARM to resume provisioning ASDLs to that NE.

When the SARM receives one of the above notifications from the NEP, it sets the NE status before moving all ASDLs from the In Progress queue to the Pending queue for that NE.

Returned Parameter Types and Formats

The NEP generates various parameters in conjunction with the State Table Interpreter or the JInterpreter, and transmits them back to the SARM.

The SARM can receive the following types of parameters from an NEP:

- **Information Parameters** – Generated by the NEP State Tables or the Java methods and returned to the SARM when the ASDLs or the methods complete. These are the only parameters that the SRP can retrieve from the SARM. Information parameters usually contain data requested by the originating system.
- **Global Parameters** – Generated by the NEP State Tables or the Java methods and returned to the SARM when the ASDLs or the methods complete. These parameters are used to maintain context between different Common Service Description Layer (CSDL) commands on the same work order.
- **CSDL (Local) Parameters** – Created by the NEP State Tables or the JInterpreter provisioning methods while processing ASDLs and returned to the SARM upon ASDL or method completion. CSDL parameters maintain context between different ASDLs on the same CSDL.
- **ASDL Rollback Parameters** – Generated by the NEP State Tables or the JInterpreter as the ASDL command is processed. These parameters maintain any information required in the rollback operation of a particular ASDL.

  When the ASDL completes, the SARM automatically adds all ASDL parameters from the forward provisioning leg to the list of Rollback parameters. If rollback is required on this ASDL, the SARM transmits the rollback ASDL command to the NEP with these rollback parameters.

For more information about these parameters, see the *ASAP Cartridge Development Guide*.

7

Configuring Network Element Processors, Resource Pools, and Devices

This chapter provides information about configuring network element processors (NEPs), resource pools, and devices.

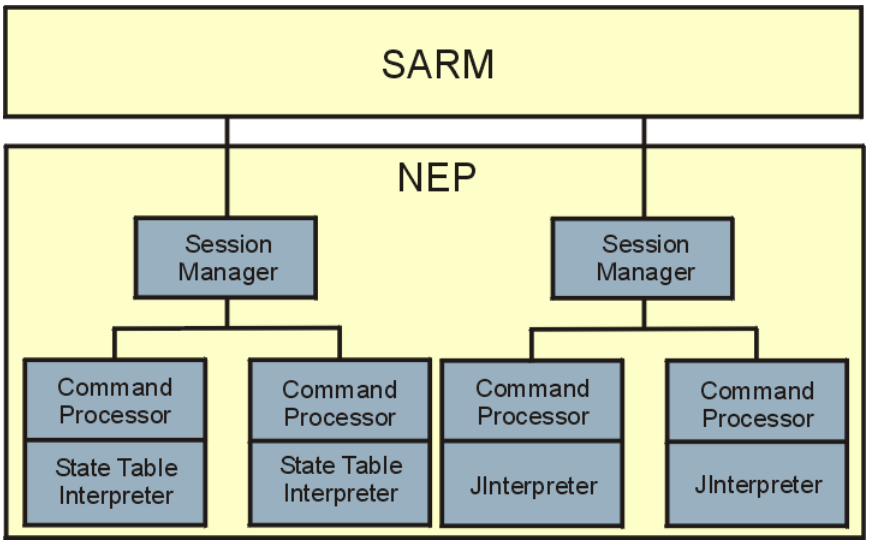
About Configuring Network Element Processors

The NEP is the ASAP component that manages all interaction with network elements (NEs) and Element Management Systems (EMSs). The NEP receives Atomic Service Description Layer (ASDL) requests from the Service Activation Request Manager (SARM) and determines the State Table scripts or Java scripts that are required to execute a dialog with the NE. It returns responses and the provisioning status from the NE to the SARM.

To manage a large number of NEs, multiple NEPs can be deployed.

A single NEP communicates with NEs through two types of interpreters: C-based and Java-based. C-based interpreter uses a set of State Tables that act as an interface between specific NEs and ASDL command execution. The Java interpreter executes Java provisioning classes. These Java provisioning classes support next-generation protocols such as CORBA, HTTP, and XML.

Figure 7–1 NEP Components



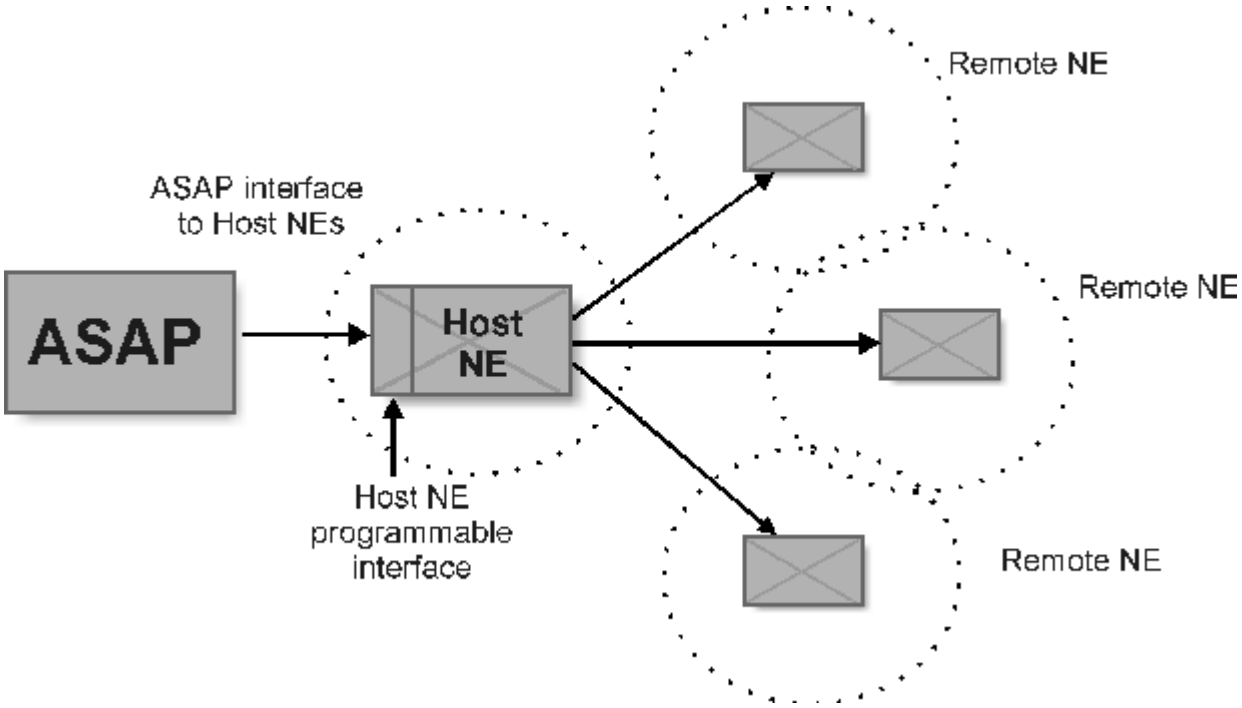
When you start ASAP or the NEP, both NEP interpreters are created (the Java Interpreter must be configured to start). The status command shows two processes for an NEP: a C process and a Java process. The name of the Java process appears as J$NEP ($NEP refers to the UNIX environment variable representing the NEP server name). The J prefix indicates a Java process.

Host and Remote NEs

A host NE is a network element that has an interface through which remote NEs can be provisioned. Several remote NEs covering a given area can be associated with a host NE, which increases the effective coverage of the NE group. Host NEs are not required to have remote NEs assigned to them. An EMS can be considered a host NE if it is configured to manage multiple physical NEs.

Figure 7–2 shows service requests in the form of ASDL commands that are routed through the host NE to the appropriate remote NE.

Figure 7–2 ASAP to NE Routing



Every ASDL references a target NE and a software load and version. This information can be configure statically in ASAP or can be provided dynamically as parameters on a work order. The software load and version is used by ASAP to identify a specific interpreter type (C or Java) and program (State Table or Java class.method) to execute.

NEP Components

This section describes the session manager, command processor, State Table Interpreter, and JInterpreter in more detail.

Session Manager

A session manager is a thread that manages all high-level interaction with an NE regardless of how many connections are established to that NE. There is a one-to-one

relationship between NEs and session managers within the NEP. Each NE has one session manager that manages at least one resource pool.

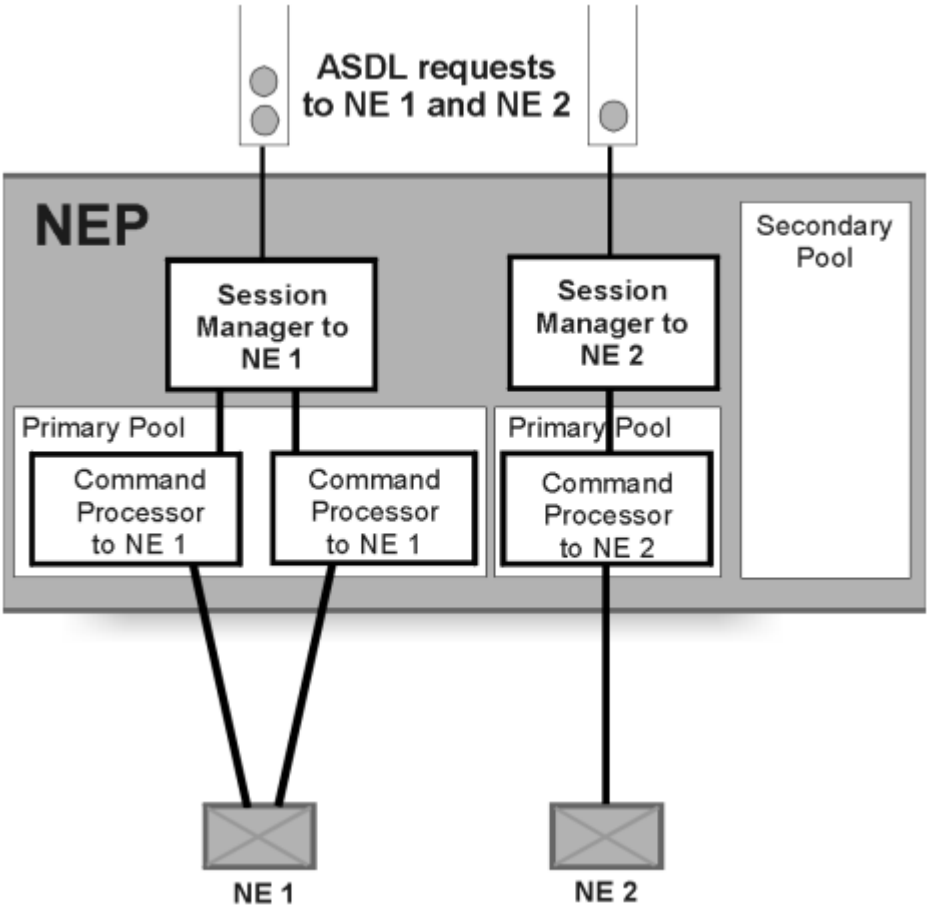
A resource pool is a set of dedicated or shared communication devices that use specific communication protocols and parameters. For example, if NEs are serial asynchronous lines, these devices correspond to physical serial ports.

There are different types of resource pools in the NEP:

- **Primary Pool** – A pool dedicated to a particular NE. It is only used to communicate with that NE. It is not necessary that all devices in the primary pool are used.
- **Auxiliary Pool** – A pool used to connect to any of the NEs that the NEP manages. A session manager can acquire these devices to communicate with its NE. When the session manager no longer requires the devices, it returns them to the auxiliary pool for other session managers to use. For example, a serial modem device could be part of an auxiliary pool.

For example, consider an NEP that is managing two NEs. The first NE has a primary pool of two devices, the second has a primary pool of one device, and the NEP has an auxiliary pool. Figure 7–3 represents the NEP configuration.

Figure 7–3 NEP Device Pools



Command Processor

A command processor is a thread that is spawned for each device, in either the primary or auxiliary pool, that manages all interaction with the NEs on a particular connection by using State Table programs or Java methods.

When a session manager receives the first connect request from the SARM, the session manager binds to a command processor in its primary pool. It requests the command processor to connect to and log in to the NE. This connection is termed the primary connection. There is only one designated primary connection to each NE. All other connections to this NE are auxiliary connections, even if they use devices from the primary pool.

The SARM uses a configurable threshold to determine when auxiliary connections to a particular NE are created and destroyed. If auxiliary connections are required, the SARM sends an NE Connect request to the session manager. The session manager then uses an available command processor in its primary pool to open another connection to the NE. If there are no other available devices in the primary pool, the session manager attempts to bind to a command processor from the secondary pool. By doing this, the session manager can manage many concurrent connections to a given NE. For instance, an NEP with five concurrent connections to an NE maintains one session manager and five command processors to manage that NE.

Each command processor can invoke either a State Table Interpreter or JInterpreter. There are still two physical processes within the NEP: a C/C++ process and Java process. The C/C++ process houses the core NEP functionality including the State Table Interpreter. The Java process hosts the JInterpreter.

The layers of functionality of the command processor are:

- **Application Layer** – Handles the communication to the session manager and uses the Interpreter functionality to provision ASDLs through either State Table or Java execution.
- **Communication Layer** – Employs the Multiple Protocol Manager (MPM) functionality to connect to host NEs through various communication protocol drivers. The MPM connects devices to host NEs with various communication protocols. MPM functionality provides a transparent interface between the user-created scripts and the protocol-specific communication details. Protocol-specific communication parameters such as asynchronous serial (wired or dialup), TCP/IP (Telnet, FTP, socket), LDAP, and SNMP interfaces are maintained in the SARM database. The NEP loads these parameters from the SARM after determining the communication protocol to use and prior to connecting to the NE. Every device interface driver provides the functionality to connect to the NE, disconnect from the NE, and transfer data using the specified protocol.

The communication layer functionality is transparent to ASAP users. Once a connection is established to the NE, the application layer handles the execution of the ASDL. The communication layer takes over when problems occur with the connection to the NE. If an ASDL was being executed when the problem occurred, the SARM is notified.

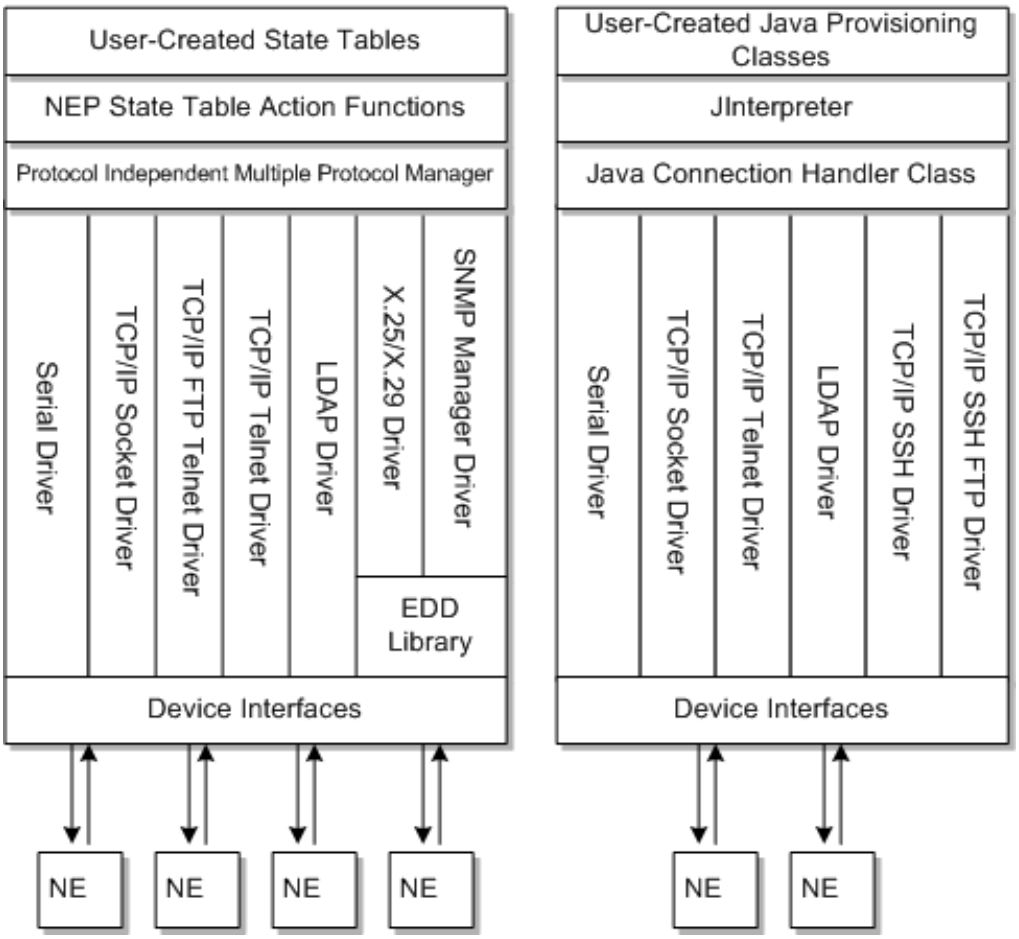
Interpreters

ASAP's interpreters execute custom code (State Table or Java) to handle device-specific communication with NEs. The custom code is designed to allow for easy, flexible and modular changes to the communication interfaces between ASAP and network elements. State Tables or Java provisioning classes are developed and maintained by a

solutions developer (systems integrator, cartridge developer or professional service) independent of the core ASAP product.

Figure 7–4 shows the logical NEP command processor structure.

Figure 7–4 Logical NEP Command Processor Structure



Interpreter code is commonly used for:

- **NE Integration** – The interpreters provide the basic facilities to communicate to, integrate with, and provision an NE. Different versions of State Tables or Java provisioning classes can be applied to the same NE technology but use a different software load. ASDL commands are mapped to the appropriate program using the technology and software version of the NE.
- **Virtual Screen Emulation** – Interpreters provide the ability to communicate with an NE using VT100 terminal emulation.
- **NE Response Parsing** – The interpreter framework provides the ability to parse NE responses. For example, NE reports can be analyzed to determine the status of the provisioning request.

Once ASAP is in production, modifications to interpreter programs are normally required only when new services are added.

Interpreter Cache Flush

NEP interpreters can load provisioning programs from either the database or the CLASSPATH. Once loaded, they are maintained in a cache for faster access. This cache can be flushed during runtime so that the NEP can dynamically reload State Tables for execution. This allows developers to dynamically load and execute new or modified provisioning programs without having to restart the server.

Cache flushing is accessed via **asap_utils** options "103. Flush State Table Cache".

Note: Only State tables can be flushed.

JInterpreter

With the JInterpreter, you can use Java to provision NEs. The JInterpreter is a Java server process that receives requests from the NEP and executes particular methods in Java classes. Downstream protocol support is provided by the core product (telnet) or by third-party libraries (CORBA, LDAP, and socket).

A Java process is paired with each C/C++ NEP process. Each Java process maintains the set of JInterpreters that execute Java provisioning classes assigned to that NEP. When the NEP starts, both State Table and JInterpreter processes are created.

Note: You can optionally disable the Java-based component of the NEP. See "Device Enabling/Disabling" for more information.

Programming with the JInterpreter requires custom connect methods to handle the logic of both connecting and logging in to the target element. This is the equivalent to LOGIN State Tables. Protocol libraries can be supplied either by third-party vendors (e.g. HTTP, JMS, LDAP, CORBA, SOAP, SFTP) or Oracle Communications (for example, telnet). The device type flag in JInterpreter only associates communication parameters with a specific protocol type (configured in **tbl_comm_param**). This means that communication parameters are made available to devices based on their association to the same device type. For information the internal mapping and management of protocols, see *ASAP Cartridge Development Guide*.

As with State Tables, new device types can be defined by modifying the constraints on **tbl_comm_param** and **tbl_resource_pool** to permit new device type definitions.

Note: Do not assign device types that are already used by the NEP.

Customizing the JInterpreter

To provide application-specific functionality for the NE connection/disconnection, and provisioning, you must extend the following base classes:

- **com.mslv.activation.jinterpreter. NEConnection** – A Java class that manages the connection and disconnection procedures to and from NEs. Developers need to implement the methods **NEConnection.connect()** and **NEConnection.disconnect to** perform the connection and disconnection from the NE. Oracle Communications provides default connection class implementations for telnet, socket, and LDAP. The socket and LDAP classes are wrappers around third-party libraries. Implementations for other protocols can be developed by a customer or integrator.

- **com.mslv.activation.jinterpreter.JProcessor** – A Java class that contains one or more provisioning methods. The provisioning methods are the JInterpreter equivalent of State Tables. Provisioning classes must derive from the **com.mslv.activation.jinterpreter.JProcessor** Java class.

For information on JInterpreter Java classes, refer to the *ASAP Java Online Reference.*

Managing Provisioning Classes

Provisioning classes contain one or more provisioning methods. Provisioning methods are the JInterpreter equivalent of traditional State Tables. Provisioning classes must derive from the abstract class, **mslv.activation.jinterpreter.JProcessor**. Provisioning methods must be public methods of the provisioning class with the following signature:

```
public void aProvisioningMethod()
```

The **mslv.activation.jinterpreter.JProcessor** class provides details on obtaining ASDL parameter values, setting the ASDL exit type, and returning parameters to SARM. For example, the SocketProcessor custom class implements a method to access to socket for messages, and set ASDL exit state to SUCCESS.

A sample demonstration class showing the implementation of a JProcessor using a connection to a socket is located in

*ASAP_Home***/samples/JeNEP/jenep_demo/Socket/SocketProcessor.java**

You can organize the methods into classes as required. The Java-enabled NEP, which maps ASDLs to provisioning methods, assumes a class name of *technology_software load* when no class is specified in the mapping; that is, when no dot **('.')** is found in the program column of table **tbl_nep_asdl_prog** for the requested ASDL.

JInterpreter provisioning classes differ from State Table action functions. With State Tables, the logic used for State Table action functions is procedural while the JInterpreter is object oriented.

Each JProcessor instance (whether it is a manifestation of the JProcessor class or a subclass) is a separate object with its own attributes. If there are two JProcessor instances, each time a method such as returnInfoParm is called from an instance, the parameters are stored in an internal structure of each instance. If this method is invoked from two different class instances that implement the same parent class, the behavior of the two methods may not be the same.

If you call returnXXXParam in instance 1, the parameters are stored in instance 1's attributes. Calling returnXXXParam in instance 2 stores the parameters in instance 2's attributes. The attributes are not shared across instances. With the JInterpreter, the JProcessor instance that is mapped to the ASDL and loaded and executed by the JInterpreter is the instance whose parameters are retrieved and returned to SARM. Each JProcessor instance operates and maintains its own state. The advantage is that the JInterpreter allows concurrent ASDLs to be processed without affecting the specific parameters of each ASDL.

Oracle recommends that you separate the protocol logic (that is, the connection logic) from the provisioning logic. This facilitates the relationship between the communication protocols and provisioning logic resulting in easier maintenance.

For more information on the tbl_nep_asdl_prog table, refer to the *ASAP Developer's Guide*.

Dynamic Reloading of Provisioning Classes

You can only reload provisioning classes that inherit from **mslv.activation.jinterpreter.JProcessor**. The lookup of classes is based on a delegation model. The following conditions apply when dynamically reloading provisioning classes:

- If the class exists in the **CLASSPATH**, it is loaded by the primordial class loader. These classes cannot be unloaded.
- If the configuration variable **LOAD_JCLASS_FROM_DB** in ASAP.cfg is set to **1**, the class loader tries to find the class in the database. If it is not found in the database, then lookup proceeds to step 3.
- If the class exists in the **JINTERP_CLASSPATH** environment variable, it is loaded by a URL class loader.

Using the JInterpreter Utility Script

Use the JInterpreter utility script, **jedd_utils** to load the provisioning class to the database and to unload the provisioning class from the database. The **jedd_utils** options are:

```
[1] Add provisioning class
[2] Delete provisioning class
[3] List provisioning classes
[Q] Quit
```

To add a provisioning class:

1. In the command line, type **jedd_utils -c Control Server Name** (the name you designated). The jedd_utils menu appears.
2. In the **Enter an option** field, type **1**.
3. In the **Program name** field, type the name of the JInterpreter program using **Package.class** name format.
4. In the **Class file name** field, type the JInterpreter class path.

To delete a provisioning class:

1. In the command line, type **jedd_utils -c $CONTROL** (the name you designated). The **jedd_utils** menu appears.
2. In the **Enter an option** field, type **2**.
3. In the **Program name** field, type the name of the JInterpreter program that you want to delete. The provisioning class is deleted using the **Package.class** name format.

To list provisioning classes:

1. In the command line, type **jedd_utils -c <$CONTROL>** (the name you designated). The **jedd_utils** menu appears.
2. In the **Enter an option** field, type **3**. A list of the JInterpreter provisioning classes appears.

State Table Interpreter

State Tables are interpreted database table-resident programs invoked by the State Table Interpreter.

State Tables are used primarily in ASAP NEPs but can also be invoked in other areas within ASAP. For instance, State Tables provide a mechanism for changing the base translation of a Service Request Processor (SRP). This can be useful as the needs of your company change and you require more sophisticated translation.

Programming State Tables is similar to writing macros. Oracle recommends that those who implement State Tables have some programming skills. In addition, they should thoroughly understand the entire provisioning process. Writers of State Table action functions should be skilled UNIX (or Linux) and C programmers.

A State Table can be created and maintained in a flat ASCII file. This allows effective source control and release procedures to be exercised in the development and release of State Tables. State Tables are compiled into SQL insert scripts to be loaded into the SARM database. To translate State Tables to SQL format use the **oracomp_npg** script.

When using **oracomp_npg**, keep in mind that the script inserts **set_escape_on** feature; as a result, backslashes that appear in the State Table are removed when the file is read into the database. Consequently, if you want to maintain control characters in the converted file, you must insert a backslash "\" before control characters.

For example, if the State Table contains the following line…

```
BCONCAT       '%VAR_BIN_MML=%VAR_MML:\n'
```

…and you want to maintain "\n" as a control character, ensure that this line in the State Table reads as follows:

```
BCONCAT       '%VAR_BIN_MML=%VAR_MML:\\n'
```

Note: It is not necessary to add a backslash to the control character "&".

Table 7–1 shows the fields used in A line of State Table code.

Table 7–1 State Table Line Contents

Field Name	Description
Line number	The four-digit line number for the programming statement. The starting line number must be greater than 0000. Oracle recommends that these line numbers be incremented in units of 10. This makes it easier to add lines later on.
Action	The name for the action code. This is the action to be performed. The associated action function is invoked.
Action string	A specific character string that is provided as an input parameter to the action function.
Action integer	A number that is provided as input to the action function that modifies its behavior.

Customizing Interpreter State Table Actions

ASAP provides a set of State Table actions action functions used to write State Table programs. These actions suffice for most processing required to interact with an NE. However, you can provide additional logic in the State Tables.

Note: Writing State Table action functions or modifying existing ones requires a skilled UNIX (or Linux) and C programmer. These customized action functions reside in code that is controlled and maintained by the systems personnel.

To provide additional logic in the State Tables, you can choose one of the following options:

- Creating libraries of State Table functions that provide particular functionality using the existing State Table action functions. You can define libraries of State Table functions that are called from other State Table programs. This lets you create and group together State Table routines that perform related functions.
- Writing new State Table actions that can be called from within the State Tables. Newly-created actions generally perform processing actions that either cannot be performed with existing state table actions, or would be too inefficient if implemented using state table logic alone.
- Overwriting existing core State Table actions provided as part of the core product.

When the Interpreter executes a State Table, the Interpreter reads the State Table from the database. Subsequent calls for the State Table are returned from this cache.

The Interpreter provides a configuration variable, SWITCH_OPTIMIZATION, that causes the Interpreter to optimize State Tables as it loads them into memory. When executed, these State Tables may appear slightly different because the Interpreter executes the instructions in a different order.

For more information on customizing Interpreter State Table actions, refer to the *ASAP Developer's Guide*.

It is possible for one State Table to execute another before returning control to the initial State Table. This provides flexibility and modularity in State Table implementation.

Connection Management

The SARM is responsible for tracking the status (up/down) of NEs as well as the number of open and available connections to each NE. If an ASDL request needs to be routed to an NE with no available connections, the SARM coordinates with the NEP to create and initialize a connection. Once the connection is available, the SARM sends the ASDL to the NEP for interpreter execution.

Before ASDLs can be provisioned, ASAP must establish a connection to the NE. If this connection cannot be established, the device associated with the connection is automatically temporarily disabled.

You can use the Automatic Re-enable feature to re-enable the device after a configurable time interval. If the failed connection is the primary connection, the NEP retries connecting to the NE after a defined interval. If the failed connection is an auxiliary connection, the SARM is notified of the auxiliary connection failure.

Many NEs require a session to login before proceeding with provisioning. This logic is implemented in LOGIN state tables or NEConnection.connect() Java methods. A LOGIN/connect implementation can be provided for every software load and version combination. ASAP handles the execution of the login procedure before provisioning an ASDL. If the login fails, ASAP handles the event as a connection failure. Upon login

failure, the connection is not disabled, instead, the command processor "sleeps" for a configurable interval and then retries the connection rather than disabling the device.

Similarly, when a particular NE requires logging off, ASAP handles this as part of the disconnection procedure. A LOGOFF State Table or NEConnection.disconnect() method can be defined for an NE to be invoked before disconnection from the NE occurs.

Connection Requests

This section describes the connection requests in ASAP.

Primary Connection

The first connection that is established successfully to an NE is designated as the primary connection. All subsequent connections are termed auxiliary connections. An end-user does not need to care or know whether their State Table or Java program has been assigned a primary or auxiliary connection.

A primary connection is treated internally by SARM. A primary connection is always the first connection to be established and the last connection to be dropped. Auxiliary connections on the other hand are spawned and killed according to thresholds for efficient connection management to an NE.

In dynamic routing, once the primary connection is dropped, the session manager for that NE is removed from memory.

For more information on dynamic routing, see page 196.

Auxiliary Connections

If more than one connection is required to an NE, auxiliary connections are opened. The SARM sends a connect request to the session manager which uses the next available command processor in its primary pool to open an auxiliary connection to the NE.

If there are no available entries in the primary pool, the session manager attempts to bind to a command processor from the secondary pool.

Dynamic routing never uses connections from the auxiliary pool, only from the dynamically-generated primary pool.

If the initial connection attempt fails or the port is configured to be disabled upon login failure and the login attempt fails, an auxiliary failure message is sent to SARM instead of retrying the connection.

Dial-up Connections

ASAP supports dial-up connections to an NE via State Tables. A dialup connection is designated by specifying a serial port dialup device type. The NEP automatically performs login, logoff and retries as necessary to manage the dialup connection.

For more information, refer to *ASAP Cartridge Development Guide*.

Disconnection Requests

The SARM controls connection disconnects by requesting that a session manager disconnect a device from an NE. The SARM issues disconnect requests based on kill thresholds or drop timer triggers.

The disconnect request types are:

- **Primary Disconnect** – A primary connection is disconnected when there is only a primary connection to the NE and the drop timer has expired.
- **Auxiliary Disconnect** – The SARM requests that one of the auxiliary connections be terminated. The session manager finds the appropriate command processor and requests it to disconnect from the NE. If the command processor is from the auxiliary pool, it returns to the auxiliary pool for all session managers within the NEP to use.

Drop Timeout Parameter

The SARM sends a request to the NEP to disconnect the primary connection to a particular NE after a period of provisioning inactivity, called Drop Timeout. If you set the drop timeout for a busy NE to a high value, such as 24 hours, the connection should never time out and is considered a dedicated connection. If you set the drop timeout to a lower value for an NE that is not very busy, the connection may time out depending on the load to the NE. In general, you should configure NEs that are busy to not time out, and NEs that are not busy to time out.

The DROP_TIMEOUT parameter is configured in **tbl_ne_config** in *ASAP Developer's Guide*.

Idle ASDL Generation

As discussed above, primary connections are disconnected based on the drop timeout interval, while auxiliary connections are disconnected as the number of pending ASDLs drops below the kill threshold.

You can configure the NEP, using communication parameters in **tbl_comm_param**, to check the integrity of an idle connection by generating an idle ASDL after a period of inactivity on the connection. The period of inactivity is defined in the parameter IDLE_TIMER_INTERVAL in seconds. The idle ASDL is defined in the parameter IDLE_TIMER_ASDL. IDLE_TIMER_ASDL is mapped to a State Table which can then perform any user-defined functions.

You can configure any device that connects to an NE to trigger IDLE_TIMER_ASDL when the connection has been idle for a specified period of time. The NEP does not send a message to the SARM that is related to processing this special ASDL since the activity is entirely within the communication layer. You can associate any State Table program or Java method with this idle ASDL, and any return status generated by the program has no effect on previous or future ASDLs from the SARM.

The execution of the IDLE_TIMER_ASDL State Table or Java program generates a message warning that an idle connection is still alive.

The communication parameter IDLE_TIMER_INTERVAL defines the idle timer. If this parameter is not defined or is set to zero, the IDLE_TIMER_ASDL is not triggered regardless of how long a connection remains idle.

The idle ASDL Interval is set as follows:

```
IDLE_TIMER_ASDL = <ASDL NAME>
IDLE_TIMER_INTERVAL = 60
```

Automatic Maintenance Mode

When an NE becomes unavailable because of a Maintenance Mode condition returned from a State Table, the session manager notifies the SARM about the status of the NE and automatically performs periodic retries to re-establish the NE connection. The session manager also issues a MAINTNCE system event that can be used to notify ASAP users that the NE is in Maintenance Mode.

Connection Thresholds

This section describes the connection thresholds in ASAP.

Spawn Threshold

The SARM triggers the creation and login of new NE connections when the number of ASDLs in the pending queue exceeds the spawn threshold. Connections created in this manner are always auxiliary connections. Once an auxiliary connection has been established, it is used concurrently with other connections to provision ASDLs to a target NE.

By default, the NEP only opens a primary connection to an NE. However, auxiliary connections can be configured to act like dedicated connections by setting a very low spawn threshold.

If the spawn threshold is exceeded by more than one ASDL request, the SARM requests that the NEP establish another auxiliary connection. This process continues as long as:

- The number of ASDL requests exceeds the spawn threshold.
- Connections to the NE are available.
- The maximum number of connections configured by the user is not exceeded.

> **Note:** The spawn threshold must always be greater than the kill threshold if multiple connections are required to a particular NE.

Kill Threshold

SARM triggers the logoff and destruction of NE connections when the number of ASDLs in the pending queue drops below the kill threshold. Connections dropped in this manner are always auxiliary connections.

When the kill threshold is exceeded, the SARM requests that one of the auxiliary connections be terminated. The session manager determines the appropriate command processor and requests that it disconnect from the NE. If the command processor was assigned from the auxiliary pool, it is returned to the auxiliary pool and is made available for all session managers within the NEP to use.

After each ASDL completion, the SARM determines if the current number of ASDLs in the Pending queue is less than the kill threshold. If it is and the ASDL was completed by an Auxiliary NE connection, the SARM sends an NE Disconnect request to the NEP. The NEP disconnects the appropriate Auxiliary port and the SARM decrements its count of available connections to that NE. This process continues as long as the ASDL Pending queue contains fewer ASDL requests than specified by the kill threshold, or until all Auxiliary connections are disconnected.

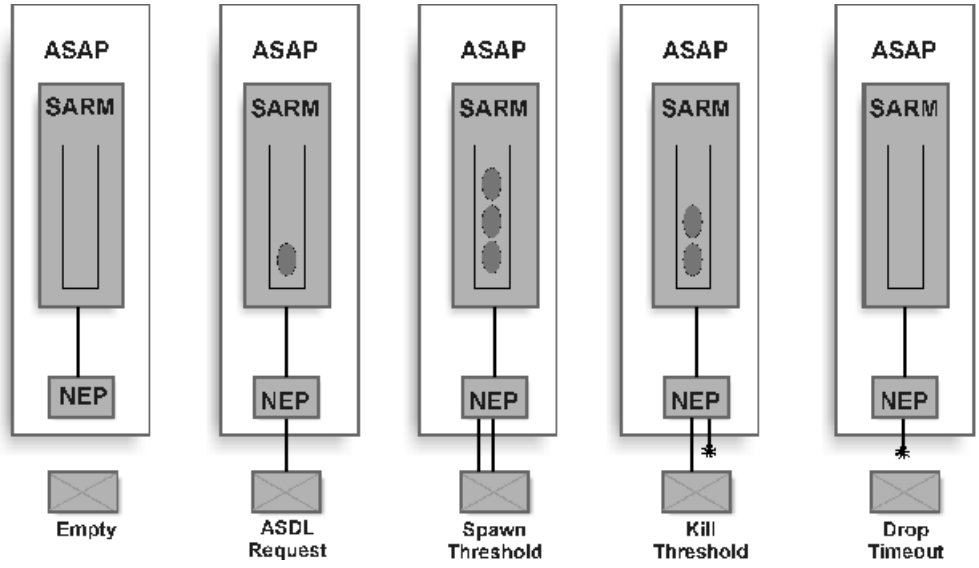
If the ASDL was completed on the Primary connection, the SARM does not send a disconnect request for the primary connection. Primary connections are dropped according to the drop timeout interval.

Maximum Available Connections

You must configure the maximum number of connections (MAX_CONNECTIONS) that can be established to each NE. The spawn threshold allows additional connections to be opened as required until the number of connections to the NE equals the maximum number of connections.

If the NE is configured within ASAP to allow more connections than is actually available in the primary and auxiliary resource pools, and the SARM sends a request to open the unavailable connection, the NEP sends an Auxiliary connection Failure Notification. When configuring the NE, you must ensure that the maximum number of allowed connections configured is not greater than the total number of connections available.

Figure 7–5 ASAP Thresholds



Note: All stored procedures that are accessed through ASAP APIs must return 0 to indicate if there is an open database cursor inside the stored procedure that needs to be closed by the ASAP C code later. If a stored procedure opens a database cursor and sends it back to the ASAP API, then the ASAP API needs to handle the closing of the connection, otherwise the connection stays open indefinitely. This may cause the ASAP system to crash when the maximum available connections are exceeded.

Device Throughput

Using the NE instance throughput control you can manage the number of ASDLs/transactions sent to the network element over time by specifying the minimum number of milliseconds each transaction should take. (A value of 0 disables NE instance throughput control.) Setting an appropriate value for throughput allows you to maximize the number of transactions processed while avoiding performance degradation due to flooding the NE with more requests than it can handle.

The throughput value is stored in **tbl_ne_config** in the column throughput. For more details, refer to *ASAP Developer's Guide*.

The NE throughput value can be configured when the NE is designed. In addition, the throughput value can be added to existing NEs or modified on existing NEs using **asap_utils**.

You can query the NE to confirm the throughput value and whether throughput on the NE is throttled by this value through **asap_utils**. See "7. ASDL/NE Queue Summary".

Device Enabling/Disabling

The System Administrator may want to deactivate an NE if, for example, the NE is down for maintenance or is in the process of upgrading. In this case, the System Administrator would:

- Deactivate and subsequently reactivate the NE through an Administrative RPC Interface, available through **asap_utils**. When the NE is disabled, any ASDLs currently being provisioning are allowed to finish. All other ASDLs for that NE are sent to the Pending queue until the NE is reactivated.

  For more information on **asap_utils**, see Appendix A, "asap_utils.".

- Set up error thresholds for the NE to deactivate itself whenever it receives too many hard failures on commands. The Enhanced Error Management functionality provides a detailed tracking scheme for hard failures related to provisioning by an NEP.

  Error thresholds control the release of specific ASDL commands to an NE to prevent an excessive number of errors from occurring. The SARM disables the NE if the actual number of consecutive errors exceeds the configured threshold in the specified time period. ASDLs for the NE are sent to the Pending queue until the NE is reactivated.

> **Note:** The dynamic activation or deactivation of an entire NE is not persistent across SARM restarts. To permanently activate or deactivate an NE, you must update the NE definitions in the database.

Automatic Device Re-enabling

A device can be disabled as the result of a connection failure. If this device is the only connection to the NE, all provisioning to the NE stops until the device is re-enabled. A device can be re-enabled either manually or automatically. A device can be re-enabled manually by using the **asap_utils** utility. Or the device will be automatically re-enabled by the command processor after a period of time has elapsed.

Device Screen and Line Diagnostics

An audit trail of all raw data transmitted to and from an NE can be used for diagnostic purposes. The ASAP System Administrator can enable or disable the dumping of such data to a file by choosing the Enable or Disable Line Diagnostics entry through **asap_utils**.

Communication to an NE through the device interface can be terminal-emulation-based or message-based. For terminal-based communication, the System Administrator can dump the contents of the virtual screen into a file by choosing the Enable or Disable Screen Diagnostics menu through **asap_utils**.

For more information on asap_utils, see Appendix A, "asap_utils.".

Connection-related ASDLs

Typically, the purpose of an ASDL command is to provision NEs. The State Table or Java class.method associated with an ASDL sends provisioning commands and processes NE responses. Most NEs require login procedures to control access to the NE. ASAP has a set of ASDLs related to NE connection management procedures including login, logoff, dialup, and hang-up (in the case of modem connections), and

connection integrity checking. These ASDLs are not sent from an upstream system or the SARM, but rather are executed by the NEP as part of connection management.

Since the connection management procedure is NE-specific, each connection management ASDL may require the execution of different State Tables or Java programs for different NE hosts.

The following connection management ASDLs are reserved as keywords to be used by ASAP. You must not use them as ASDL command labels. The reserved class.methods in brackets apply to Java programs

- LOGIN (NEConnection.connect)
- DIALUP
- HANGUP
- LOGOFF (NEConnection.disconnect)

Resending Completed ASDLs

The Resend Completed ASDLs function (also known as the Recent Change Retransmission (RCR) function enables you to resend successfully completed ASDL commands in their original provisioning sequence to an NE. This feature is intended for use in situations in which a NE has been recovered to a known state and ASAP is requested to re-activate all successful commands to the NE from that known state onwards.

This mechanism assumes that ASAP is the sole system provisioning such NEs. If not, there is no guarantee that the provisioning commands are entered to the NE in the same provisioning sequence because the multiple systems may not be synchronized in their updates.

SARM provides this functionality in the following ways:

- SARM maintains a log of all successfully completed ASDL commands to each NE.
- Upon your request, the SARM reloads all completed ASDL commands that are successfully completed on the NE within the specified time period.
- The SARM then re-provisions these ASDL commands serially, in their original provisioning sequence.
- Errors arising from this process are logged for you to view.

To resend completed ASDLs, use option 11. Resend Completed ASDLs of asap_utils. You can invoke asap_utils by typing the following at the $ASAP_base prompt:

```
asap_utils [-P ctrl_password] [option]
```

where:

- [-P ctrl_password] is the password for the control database
- [option] is the asap_utils option you want to invoke. For the Resend Completed ASDLs option, type 11.

> **Note:** You can only use asap_utils to resend completed ASDLs; you cannot send RCR requests using the Order Control Application (OCA) client.

The arguments for this option include the target NE, the start and end dates/times (in 'Mmm dd yyyy hh:mm[AM/PM]' format, for example 'Dec 31 2004 11:59PM').

When the you invoke the Resend Completed ASDLs command from asap_utils, ASAP determines all of the ASDL commands that have successfully completed on the network element and the sequence in which they were originally executed.

Specifically, if rollback is not being used, then RCR functionality retrieves all ASDLs with a state of ASDL_NEP_COMPLETE. These are ASDLs that have successfully executed on the network element, as shown in the following example.

Table 7–2 Inclusion of ASDLs (Rollback not in use) in RCR Order

CSDL	ASDL	Description	Final ASDL State	Include in RCR Order?
CSDL1	ASDL1	ASDL completes	ASDL_NEP_COMPLETE	Yes
CSDL1	ASDL2	ASDL completes	ASDL_NEP_COMPLETE	Yes
CSDL1	ASDL3	ASDL completes	ASDL_NEP_COMPLETE	Yes
CSDL2	ASDL4	ASDL completes	ASDL_NEP_COMPLETE	Yes
CSDL2	ASDL5	ASDL completes	ASDL_NEP_COMPLETE	Yes
CSDL2	ASDL6	ASDL fails	ASDL_NEP_FAIL	No

If rollback is configured, then the RCR functionality retrieves all ASDLs with a state of ASDL_NEP_COMPLETE or ASDL_NEP_RBACK_FAIL. ASDLs with a state of ASDL_NEP_RBACK_FAIL are those ASDLs that successfully executed on the switch but whose rollback ASDL has failed (rollback was triggered due to a subsequent ASDL failure on the order). The following example shows which ASDLs falling within the specified time period that would be included in an RCR order:

Table 7–3 Inclusion of ASDLs (Rollback in use) in RCR order

CSDL	ASDL	Description	Final ASDL State	Include in RCR Order?
CSDL1	ASDL1	Rollback of ASDL completes	ASDL_NEP_RBACK_COMP	No
CSDL1	ASDL2	Rollback of ASDL fails	ASDL_NEP_RBACK_FAIL	Yes
CSDL1	ASDL3	Rollback of ASDL completes	ASDL_NEP_RBACK_COMP	No
CSDL1	ASDL4	Failure of ASDL triggers rollback	ASDL_NEP_FAIL	No

The final ASDL state is not the state of the rollback ASDL itself, but the final state of the "forward" ASDL that is being rolled back.

ASAP then automatically generates a new RCR work order ID that contains a single CSDL that maps to all of the ASDLs that were successfully executed against the NE.

ASAP automatically submits and performs the RCR work order.

All ASDL commands are executed in the sequence in which they were originally executed until the work order is complete.

If during the execution of the RCR work order an ASDL exits with a hard failure, ASAP will interpret this as a delayed failure, allowing the entire RCR work order to finish executing before generating a work order failure event.

RCR work orders start with an RCR prefix (for example, RCR-031125184951). When executing this option, ASAP displays the new work order name before returning to the UNIX prompt.

Configuring NEPs

NEP configuration information is held in static tables in the ASAP Control server database and the SARM database. To change configuration information, use the Service Activation Configuration Tool, or SQLplus sessions to invoke stored procedures.

This section outlines the procedures for configuring an NEP using stored procedures.

For instructions on using the Service Activation Configuration Tool, see Chapter 2, "Configuring ASAP Servers."

Adding an NEP

To add an NEP to the system, perform the following steps:

1. Export the current database using the **export_tool.sh** (see "Transforming ASAP Database Configurations or Service Models into XML"). For example:

```
export_tool.sh -m -p abc -t config
```

2. Open the **transOut2_config.xml** file using a text editor.

```
gedit transOut2_config.xml
```

3. Remove all elements except for the existing NEP

```
<activationConfig
xmlns="http://www.metasolv.com/ServiceActivation/2003/ActivationConfig"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.metasolv.com/ServiceActivation/2003/ActivationCo
nfig
M:\hlam_view\ASAP_base\ASAP\jmx\xsd\ActivationConfig.xsd">

<nepServer name="NEP_envid" xsi:type="NEPServerType">
        <description>NEP Server</description>
        <system>envid</system>
        <territory>envid</territory>
        <diagnosticFilename>NEP_envid.diag</diagnosticFilename>
        <diagnosticLevel>LOW_LEVEL</diagnosticLevel>
        <autoStart>true</autoStart>
        <controlServer>CTRLenvid</controlServer>
        <interfaceHostname>
                <hostname>test.can.com</hostname>
        </interfaceHostname>
        <interfacePort>40013</interfacePort>
        <secondaryPool/>
        <jinterpreterPort>40014</jinterpreterPort>
        <program>asc_nep</program>
        <enableJInterpreter>true</enableJInterpreter>
</nepServer>

</activationConfig>
```

   Where *envid* is the environment ID for your ASAP instance.

4. Modify the NEP element with new values where required. For example:

```
<activationConfig
xmlns="http://www.metasolv.com/ServiceActivation/2003/ActivationConfig"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.metasolv.com/ServiceActivation/2003/ActivationCo
nfig
M:\hlam_view\ASAP_base\ASAP\jmx\xsd\ActivationConfig.xsd">

<nepServer name="NEP_2envid" xsi:type="NEPServerType">
        <description>NEP Server</description>
        <system>envid</system>
        <territory>envid</territory>
        <diagnosticFilename>NEP_2envid.diag</diagnosticFilename>
        <diagnosticLevel>LOW_LEVEL</diagnosticLevel>
        <autoStart>true</autoStart>
        <controlServer>CTRLenvid</controlServer>
        <interfaceHostname>
                <hostname>test.can.com</hostname>
        </interfaceHostname>
        <interfacePort>40023</interfacePort>
        <secondaryPool/>
        <jinterpreterPort>40024</jinterpreterPort>
        <program>asc_nep</program>
        <enableJInterpreter>true</enableJInterpreter>
</nepServer>

</activationConfig>
```

5. Save the file.
6. Update the file using **sactConfig** as described in "Running the SACT Scripts".

Deleting an NEP

To delete an NEP, perform the following steps:

1. Export the current database using the **export_tool.sh** (see "Transforming ASAP Database Configurations or Service Models into XML"). For example:

   ```
   export_tool.sh -m -p abc -t config
   ```

2. Open the **transOut2_config.xml** file using a text editor.

   ```
   gedit transOut2_config.xml
   ```

3. Remove all elements except for the NEP you want to delete.

   ```
   <activationConfig
   xmlns="http://www.metasolv.com/ServiceActivation/2003/ActivationConfig"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.metasolv.com/ServiceActivation/2003/ActivationCo
   nfig
   M:\hlam_view\ASAP_base\ASAP\jmx\xsd\ActivationConfig.xsd">

   <nepServer name="NEP_2envid" xsi:type="NEPServerType">
           <description>NEP Server</description>
           <system>envid</system>
           <territory>envid</territory>
           <diagnosticFilename>NEP_2envid.diag</diagnosticFilename>
           <diagnosticLevel>LOW_LEVEL</diagnosticLevel>
           <autoStart>true</autoStart>
           <controlServer>CTRLenvid</controlServer>
           <interfaceHostname>
   ```

```
                <hostname>test.can.com</hostname>
        </interfaceHostname>
        <interfacePort>40023</interfacePort>
        <secondaryPool/>
        <jinterpreterPort>40024</jinterpreterPort>
        <program>asc_nep</program>
        <enableJInterpreter>true</enableJInterpreter>
</nepServer>

</activationConfig>
```

Where *envid* is the environment ID for your ASAP instance.

4. Save the file.
5. Update the file using **sactUnconfig** as described in "Running the SACT Scripts".

Adding Configuration Parameters to an NEP

To add configuration parameters to an NEP, perform the following steps:

1. Export the current database using the **export_tool.sh** (see "Transforming ASAP Database Configurations or Service Models into XML"). For example:

   ```
   export_tool.sh -m -p abc -t config
   ```

2. Open the **transOut2_config.xml** file using a text editor.

   ```
   gedit transOut2_config.xml
   ```

3. Remove all elements except for the NEP you want to add configuration parameters to. For example:

   ```
   <activationConfig
   xmlns="http://www.metasolv.com/ServiceActivation/2003/ActivationConfig"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.metasolv.com/ServiceActivation/2003/ActivationCo
   nfig
   M:\hlam_view\ASAP_base\ASAP\jmx\xsd\ActivationConfig.xsd">

   <nepServer name="NEP_envid" xsi:type="NEPServerType">
           <description>NEP Server</description>
           <system>envid</system>
           <territory>envid</territory>
           <diagnosticFilename>NEP_envid.diag</diagnosticFilename>
           <diagnosticLevel>LOW_LEVEL</diagnosticLevel>
           <autoStart>true</autoStart>
           <controlServer>CTRLenvid</controlServer>
           <interfaceHostname>
                   <hostname>test.can.com</hostname>
           </interfaceHostname>
           <interfacePort>40013</interfacePort>
           <secondaryPool/>
           <jinterpreterPort>40014</jinterpreterPort>
           <program>asc_nep</program>
           <enableJInterpreter>true</enableJInterpreter>
   </nepServer>

   </activationConfig>
   ```

   Where *envid* is the environment ID for your ASAP instance.

4. Add the configuration parameters. For example:

```
<activationConfig
xmlns="http://www.metasolv.com/ServiceActivation/2003/ActivationConfig"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.metasolv.com/ServiceActivation/2003/ActivationCo
nfig
M:\hlam_view\ASAP_base\ASAP\jmx\xsd\ActivationConfig.xsd">

<nepServer name="NEP_envid" xsi:type="NEPServerType">
        <description>NEP Server</description>
        <system>envid</system>
        <territory>envid</territory>
        <diagnosticFilename>NEP_envid.diag</diagnosticFilename>
        <diagnosticLevel>LOW_LEVEL</diagnosticLevel>
        <configurationParameters>
             <configurationParameter xsi:type="AUTOMATIC_BLACKOUT_CHECK">
                 <value>1</value>
             </configurationParameter>
             <configurationParameter xsi:type="CACHE_BLACKOUT_TABLE">
                 <value>1</value>
             </configurationParameter>
             <configurationParameter xsi:type="LOAD_JCLASS_FROM_DB">
                 <value>0</value>
             </configurationParameter>
        </configurationParameters>
        <autoStart>true</autoStart>
        <controlServer>CTRLenvid</controlServer>
        <interfaceHostname>
                <hostname>test.can.com</hostname>
        </interfaceHostname>
        <interfacePort>40013</interfacePort>
        <secondaryPool/>
        <jinterpreterPort>40014</jinterpreterPort>
        <program>asc_nep</program>
        <enableJInterpreter>true</enableJInterpreter>
</nepServer>

</activationConfig>
```

5. Save the file.
6. Update the file using **sactConfigNR** as described in "Running the SACT Scripts".

JNEP Logging

JNEP uses Log4j to manage system log messages. For details on Log4j, see *ASAP System Administrator's Guide*.

Configuring Resource Pools and Resource Pool Devices

tbl_resource_pool is a static table that defines the collection of command processors (devices) that the NEP uses to establish connections to NEs. Groups of command processors can be contained in a resource pool. Each NE configuration determines a primary resource pool that contains one or more devices the NEP uses to connect to that NE. These devices are not used to connect to other NEs. Each NEP has an auxiliary resource pool that contains devices used by the NEP to establish connections to any NE managed by the NEP. These primary and auxiliary resource pools are defined in this table.

You must create resource pools and add resource pool devices by populating the **tbl_rerouce_pool** table before you can add command processors to it. You can create Java processors using Design Studio when you create Java cartridges.

This section outlines the procedures for creating, deleting, or editing resource pools and devices using the SACT. For instructions about using the Service Activation Configuration Tool, see Chapter 2, "Configuring ASAP Servers."

To define, delete, and list resource pools and devices using stored procedures, see Appendix , "Configuring Resource Pools Using Stored Procedures".

Adding a Resource Pool and Device

To add a resource pool and device to the system, perform the following steps:

1. Export the current database using the **export_tool.sh** (see "Transforming ASAP Database Configurations or Service Models into XML"). For example:

   ```
   export_tool.sh -m -p abc -t config
   ```

2. Open the **transOut2_config.xml** file using a text editor.

   ```
   gedit transOut2_config.xml
   ```

3. Remove all elements except for one of the default connection pools.

   ```
   <activationConfig
   xmlns="http://www.metasolv.com/ServiceActivation/2003/ActivationConfig"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.metasolv.com/ServiceActivation/2003/ActivationCo
   nfig
   M:\hlam_view\ASAP_base\ASAP\jmx\xsd\ActivationConfig.xsd">
   <connectionPool name="POOL">
           <device name="dev1">
                   <environment>DEVELOPMENT</environment>
                   <lineType>TELNET_CONNECTION</lineType>
           </device>
   </connectionPool>

   </activationConfig>
   ```

   Where *envid* is the environment ID for your ASAP instance.

4. Modify the NEP element with new values where required. You can also create additional devices by adding **device name** elements. For example:

   ```
   <activationConfig
   xmlns="http://www.metasolv.com/ServiceActivation/2003/ActivationConfig"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.metasolv.com/ServiceActivation/2003/ActivationCo
   nfig
   M:\hlam_view\ASAP_base\ASAP\jmx\xsd\ActivationConfig.xsd">
    <connectionPool name="POOL2">
           <device name="dev2">
                   <environment>DEVELOPMENT</environment>
                   <lineType>SOCKET_CONNECTION</lineType>
           </device>
           <device name="dev3">
                   <environment>DEVELOPMENT</environment>
                   <lineType>SOCKET_CONNECTION</lineType>
           </device>
   </connectionPool>
   ```

```
</activationConfig>
```

5. Save the file.
6. Update the file using **sactConfig** as described in "Running the SACT Scripts".

Deleting a Resource Pool and Device

To delete an resource pool, perform the following steps:

1. Export the current database using the **export_tool.sh** (see "Transforming ASAP Database Configurations or Service Models into XML"). For example:

```
export_tool.sh -m -p abc -t config
```

2. Open the **transOut2_config.xml** file using a text editor.

```
gedit transOut2_config.xml
```

3. Remove all elements except for the NEP you want to delete.

```
<activationConfig
xmlns="http://www.metasolv.com/ServiceActivation/2003/ActivationConfig"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.metasolv.com/ServiceActivation/2003/ActivationCo
nfig
M:\hlam_view\ASAP_base\ASAP\jmx\xsd\ActivationConfig.xsd">
 <connectionPool name="POOL2">
        <device name="dev2">
                <environment>DEVELOPMENT</environment>
                <lineType>SOCKET_CONNECTION</lineType>
        </device>
</connectionPool>

</activationConfig>
```

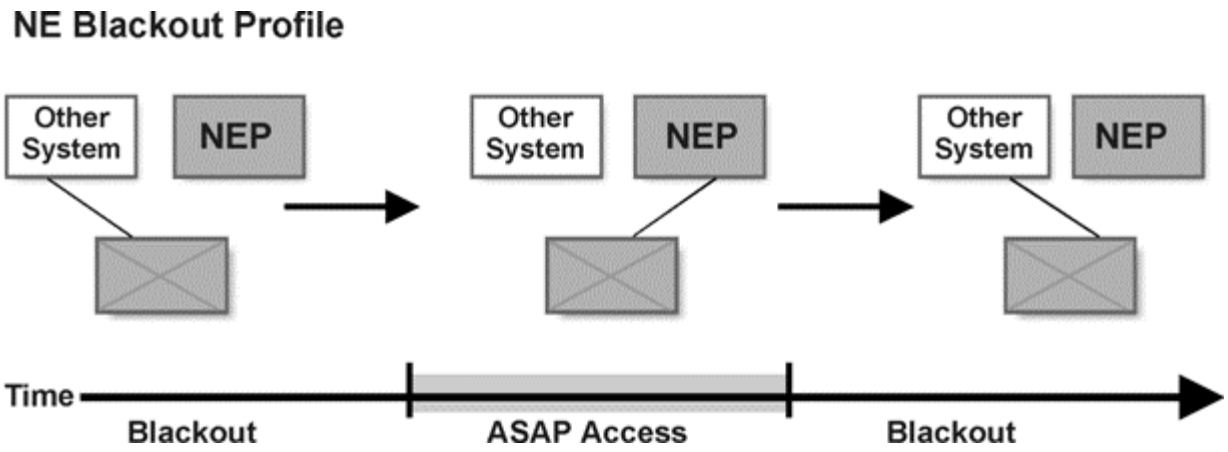
4. Save the file.
5. Update the file using **sactUnconfig** as described in "Running the SACT Scripts".

Configuring NE Blackout Periods

If ASAP must share a port to an NE with another system (as illustrated in the figure below) or if regular NE maintenance must be performed, you can define the NE blackout period during which time the NEP will not connect to that NE.

Figure 7–6 shows an NE blackout profile.

Figure 7–6 NE Blackout Periods



The NE Blackout feature automates ASAP NE Access blackout at scheduled times so that the ASAP System Administrator does not have to disable and enable the NE manually before and after the blackout period. When the NEP detects a blackout period during a check of a database table while executing the **BLACKOUT** State Table action, the NE is put into Maintenance Mode automatically.

You can configure the blackout period in two ways:

- **Weekly Periodic (Static)** – A day of the week, start time, and end time are specified. For example, you can set regular maintenance on the NE to be performed every Sunday between 2:00 a.m. and 4:00 a.m.
- **Absolute Date and Time (Dynamic)** – An absolute date and time for both the start and end periods are specified rather than a day of the week. For example, April 21, 2003 23:59 and April 22, 2003 1:59.

Table 7–4 displays the blackout information you can configure.

Table 7–4 Blackout Configuration

Field	Explanation
Host NE	The identifier of the host NE.
Day of Week	The day of the week the blackout period starts and ends. If NULL, the blackout period is an Absolute Date and Time.
Start Time	The start of the blackout period.
End Time	The end of the blackout period.
Description	An explanation or reason for the blackout period.

Note that if the dayname entry is present, then the configuration is considered to be Static. The implementation will check for the start time and end time alone, and assume that both fall on the same day. When configuring a blackout period that spans from one day to the next (for example from 22:00 until 01:00 the next day) you must configure two separate lines in **tbl_blackout**: one for 22:00:00 till 23:59:59 and another from 00:00:00 till 01:00:00.

Use the following stored procedures to define, list, and delete blackout definitions.

- **SSP_add_blackout** – This procedure configures the static and dynamic blackout periods for a specific network element host.

- **SSP_list_blackout** – This procedure lists blackout periods for a specific network element host.
- **SSP_del_blackout** – This procedure removes blackout periods for a specific network element host.

For more information on these stored procedures, refer to *ASAP Developer's Guide*.

Blackout information is maintained in tbl_blackout. For more information on **tbl_blackout**, refer to *ASAP Developer's Guide*.

Note: NE blackout periods can be configured with XML using the SACT or the **asapConfig** utility.

Checking NE Blackout Periods

In previous versions of ASAP, executing the BLACKOUT action function is the only way to determine the NE blackout period. The BLACKOUT action function cannot be executed within certain predefined state table programs such as LOGIN, LOGOUT, DIALUP, and HANGUP. Consequently, the physical connection between ASAP and the NE is required to check NE blackout period.

You can now check the NE blackout period without having a physical connection with the NE.

If the AUTOMATIC_BLACKOUT_CHECK variable in the ASAP.cfg is enabled, the NEP checks the NE blackout period in two places without having a physical connection with NE:

- Before connecting with the NE
- Before processing an ASDL request

When the NEP detects the NE blackout period, the NEP disconnects every existing connection with the NE and notifies the SARM that the NE is in maintenance mode. Then the NEP checks the NE blackout period at the interval, in seconds, as configured in the ASAP.cfg file.

The BLACKOUT action function can be used to check the NE blackout period when AUTOMATIC_BLACKOUT_CHECK is set to 0. If AUTOMATIC_BLACKOUT_CHECK is set to 1, then the BLACKOUT state table action function will not do any processing. It will always return successfully and continue processing the remainder of the state table.

The stored procedure **SSP_check_blackout** enables you to check whether or not the specified network element is currently blacked out.

For more information on this stored procedures, refer to *ASAP Developer's Guide*.

Configuration Parameters for NE Blackout

Table 7–5 describes the ASAP configuration parameters that control automatic NE blackout checking functionality.

Table 7–5 NE Blackout Configuration Parameters

Variable	Default	Description
AUTOMATIC_BLACKOUT_CHECK	0	The boolean type variable controlling automatic NE blackout checking. ■ 0 - disable ■ 1 - enable
CACHE_BLACKOUT_TABLE	1	This variable controls whether the NEP caches the NE blackout table information in the memory or not. This variable works only when the AUTOMATIC_BLACKOUT_CHECK configuration variable is set to 1. ■ 1 - enable caching ■ 0 - disable caching CACHE_BLACKOUT_TABLE =1 should not be used if you want to change blackout configuration on a regular basis.

8

Managing the Admin Server

This chapter provides information about managing the Admin server.

About Managing Admin Servers

The Admin server offers some system monitoring capabilities using administrative remote procedure calls (RPCs) to retrieve information from each server in the system. The Admin server acts as the collection focal point for all of this information, which can be obtained using the ASAP utility script, **asap_utils**. See Appendix A, "asap_utils" for more information.

9

Configuration Parameters

This chapter provides information about ASAP configuration parameters.

About ASAP Configuration Parameters

Every Oracle Communications ASAP application, whether client or server, references configuration files to determine configuration information. ASAP has a global file, **ASAP.cfg**, that maintains global configuration information, as well as component-specific information that may overwrite the globally-defined data. If a configuration parameter has not been defined in this configuration file, the application provides a suitable default value.

The **ASAP.cfg** file is located in the *ASAP_home*/**config** directory (where *ASAP_home* is the location of your ASAP server installation directory).

Every ASAP application client and server uses an ASAP API. The API provides considerable functionality to the application process, including configuration parameter management, system event generation, performance and process parameters, diagnostics, and database administrative functions.

Each parameter grouping provides a definition of the default character string value and a brief description of the configuration parameter settings.

Global parameters can usually be specified once in the global configuration section (beginning with the comment **#### Common API Configuration Parameters**).

Local parameters should be specified in the local configuration section (beginning with the first application server listed in square brackets [] followed by a comment, for example, **[CTRL] #### Control Server Configuration Parameters**).

Note: To start ASAP, you must define the server application name within the brackets [ ] in the header for each Server Configuration Parameters section in the **ASAP.cfg** file. You must define the server application name for each section, for example, the CTRL, OCA, SRP, SARM, NEP, and ADM. Ensure there are no empty brackets [ ] in any of the sections in the Server Configuration Parameters; otherwise, system errors occur.

Note: The maximum length for a configuration parameter value is 128 characters.

Configuration parameters are grouped as follows:

- **Global Configuration Parameters** – Parameters appearing at the beginning of the **ASAP.cfg** file before the application-specific sections and providing ASAP-wide parameter values.
- **Application-specific Configuration Parameters** – Parameters residing within the **ASAP.cfg** file in sections delimited by [*ApplName*], where *ApplName* represents the application name.

Determining Configuration Parameters

To determine the configuration parameter values, use the **ASC_get_config_param**() API call or in the case of object oriented libraries the **Config::get_config_param()** API call. You can specify the parameter values in the application-specific section or the global section of the **ASAP.cfg** file. If they are not specified, the default value is used.

Configuration parameters are read from the configuration file upon application startup and are referenced from a cache within the application's memory from that point onwards. Therefore, the application only has to reference the configuration parameter by means of this API call once.

- Application-specific section of **ASAP.cfg** – If the parameter value is specified in this section, its value overrides the parameter values specified in the global section. The application-specific section is determined to be the first section in the file which matches a substring of the application name. For example, an [SRP] section would match SRP_TST1, SRP_TST2 and SRP_PRD1 whereas an [SRP_P] section would only match SRP_PRD1. This is used to specify separate production configurations to those in test environments. In addition, if the SRP_PRD1 application is determining a configuration parameter in a configuration file in which there are both [SRP] and [SRP_P] sections, then the first of these two sections in the file is assumed to be the application-specific section. For this reason, you must specify the application-specific sections before the general ones. For example, you would place the [SRP_P] section prior to the [SRP] section.
- Global section of **ASAP.cfg** – If the parameter value is specified in this section but not in the application-specific section, the value of the global parameter is assumed.
- If a parameter is not specified in either the application-specific or global sections, the default value passed to the API call is assumed.

Configuration Parameter Scope

The configuration parameters are grouped in sections that correspond to their scope within the system. For each parameter, the default character string value is detailed together with a brief description of the configuration parameter settings. In addition, the configuration section in which this parameter is usually placed is identified. However, you can choose to place the configuration parameters in different sections.

- **Local** – The parameter should be specified in the local configuration section (for example, APPL_USERID has to be explicitly set for each application process).
- **Global** – The parameter can usually be specified once in the global configuration section (for example, APPL_SQL_SERVER) because these settings may be system wide.

Environment Variable Support

To support the insertion of environment variables within the configuration files, special processing is performed whenever the configuration parameter begins with the "$" character. An example appears below:

```
APPL_USERID = $USERID
```

The logic is as follows:

- If the configuration parameter value you specify begins with "$", the API determines whether the UNIX environment variable USERID is defined in the environment. If it is defined, the value of this environment variable is used for the configuration parameter, APPL_USERID.
- If the environment is not defined, the configuration parameter USERID is set as usual in the application; that is, to $*USERID*.

This facility allows the inclusion of environment variables within the configuration files. This eases the configuration effort required to maintain a number of ASAP environments especially during testing phases.

UNIX Environment Variables

The configuration parameter design allows for multiple environment variables in the right side of the equation. The following examples show this configuration parameter design:

```
PASS_FILE = $LOGDIR/pass_file.tmp
SRP_WO_FILE = $LOGDIR/work_orders/$SRP_NAME.$WO_EXT
```

Common API Configuration Parameters

The API common to both application servers and clients is represented by the Common API library (libasc), as well as configuration parameters common to both the Client (libclient) and Server (libcontrol) APIs. The configuration parameters in the following sections are used by the common components of servers and clients.

Logical-to-Network Application Name Mapping

You can specify the configuration parameters to allow an application process (client or server) to determine a network application name from the application's logical name, which usually resides within the database. The network application name is used to define the application to the network. This capability is only used when the logical application name is different from the network name.

You can globally define these configuration parameters for the system. The definition of these configuration parameters are required as an application process reading a logical name from the database references them to determine the network application name. If this is not specified, the network application name will default to the logical application name.

For example, the Service Activation Request Manager (SARM) application server accesses the configuration file to determine the network server names of all Service Request Processors (SRPs) and Network Element Processors (NEPs) to which it may establish network connections. Another example is the Control server, which requires the network name of each application server that it establishes network connections to.

To perform logical to network application name mapping, see Table 9–1.

Table 9–1 Logical to Network Application Name Mapping

Default Value	Config Section	Description
Generic Process Name	Global	The GENERIC configuration parameter represents the logical name of an ASAP application process. It is set equal to the network process name in this environment. For example, in the production environment, the configuration file may include entries as follows: ■ SARM=SARM_P ■ NEP01=NEP01_P ■ NEP02=NEP02_P Each application process that is to be spawned by the Control server must be defined as above if network names different from logical names are employed, because the Control server reads the logical names from the database and then uses these configuration file settings to determine the network names.

Table 9–2 lists and describes the logical-to-network application name mapping parameters and values.

Table 9–2 Logical-to-Network Application Name Mapping Parameters, and Values

Parameter	Description	Value
MASTER_CONTROL	Determines which Control server is the master Control server in the ASAP system.	Default = $*MASTER_CONTROL*.

ASAP Monitoring Parameters

Every Host machine in ASAP is assigned a Control server to monitor and control the ASAP process for that machine. The Admin server also provides some system monitoring capabilities using administrative Remote Procedure Calls (RPCs).

Table 9–3 lists and describes the ASAP monitoring parameters and values used to monitor the server performance.

Table 9–3 ASAP Monitoring Parameters, and Values

Parameter	Description	Value
SYSMON_ALWAYS_ON	Boolean flag. If set to 1, activates system monitoring upon startup. Default = 0.	Default = $*MASTER_CONTROL*.
SYSMON_LOG_ENABLED	Records work order logging information. This feature significantly increases the memory and CPU overhead.	Default = 0.
SYSMON_DUMP_ TRANSACTIONS	Dumps transaction details to a file. This feature significantly increases the memory and CPU overhead.	Default = 0.
SYSMON_SHOW_ DESCRIPTIONS	Shows descriptions of various sections of the generated report.	Default = 0.
SYSMON_DELIMITED_ OUTPUT	Generates ASCII formatted file (0) or comma-delimited spreadsheet file (1).	Default = 0.

Connection Pool Manager and Debugging Tools

Use the parameters in this section to set up the Connection Pool Manager (CPM) to automatically tune connection pools and log internal application server pools statistics.

Table 9–4 lists and describes the CPM and debugging tools parameters and values.

Table 9–4 Connection Pool Manager and Debugging Tools Parameters, and Values

Parameter	Description	Value
ASAP_DEVELOPMENT_ MODE	Boolean flag. If set to 1, it is used during the development process to aid in debugging and tuning the system. It should be set to 0 in production environments.	Default = 0.
CONNECT_TO_CTRL	Boolean flag. If set to 1, it instructs the client application to open a connection to the Control server to log system events.	Default = 1.
CPM_OPTIMIZE_POOLS	Boolean flag. If set to 1, it enables auto-tuning of connection pools by the CPM. The algorithm that CPM_ OPTIMIZE_POOLS uses does not guarantee that the oldest unused connection in the pool is closed first. This flag must not be used in a large scale production environment.	Default = 0.
DEBUGGER_ON	Enables the Interpreter State Table debugger. If set, it initializes the Interpreter debugging facilities.	Default = 1.
DIAG_LINE_FLUSH	Boolean flag. If set to 1, it determines whether the diagnostic file output is flushed to disk at the end of each diagnostic line. If set to 0, the diagnostics are only flushed to the diagnostic file if a PROGRAM or SANITY level diagnostic message is written, or if the I/O buffer is flushed by the operating system. You can change this diagnostic parameter by adjusting this configuration parameter or changing the diagnostic line flush flag of the application server by using the **diag_ line_flush()** API RPC/Registered Procedure to the particular server.	Default = 1. This option should be set to 0 in a production environment because it causes considerable performance overhead in disk activity.
LANGUAGE_DUMP_ON	Boolean flag. If set to 1, it determines whether the language buffer being transmitted or received should be logged as a low-level diagnostic in the server's diagnostic log file. The primary use is for debugging such language buffers and is, therefore, generally used in non-production environments.	Default = 1.
RPC_SHOW_OUTGOING_ PARAMS	N/A	Default = 1.
SYSMON_ALWAYS_ON	Activates system monitoring upon startup.	Default = 1.

Application Logical to Network Application Name Mapping

This section describes application logical to network application name mapping.

Table 9–5 lists and describes application logical to network application name mapping of parameters and values.

Table 9–5 Application Logical to Network Application Name Mapping of Parameters, and Values

Parameter	Description	Value
SARM	The logical name of the SARM server.	Default = $*SARM*.
SRP_EMUL	The logical name of the SRP Emulator.	Default = $*SRP_EMUL*.

SQL Server Security-Related Parameters

The following parameters are associated with establishing connections to a SQL Server using various user ID/password combinations to determine the default database within the SQL Server. The SQL Server names rarely need to be specified, but the user ID and password combination are likely be explicitly set in either the local or global configuration file.

Table 9–6 lists and describes the SQL Server security-related parameters and values.

Table 9–6 SQL Server Security-Related Parameters, and Values

Parameter	Description	Value
APPL_USERID	The primary SQL Server user ID the application process uses to connect to the APPL_SQL_SERVER SQL Server. The user ID defaults to the primary application database, and therefore, the application does not need database names because the definition of the user in the SQL Server specifies that user's default database. Maximum length is 20 alphanumeric characters.	Default = $*APPL_USER*.
APPL_SQL_SERVER	The SQL Server in which the primary application database resides. If you specify this, the database can reside on a separate SQL Server from other ASAP databases if required. Maximum length eight alphanumeric characters.	Default = $*DSQUERY*.
CONTROL_USERID	The SQL Server user ID the application process uses to connect to the Control database in the CONTROL_SQL_SERVER SQL Server. Maximum length is 20 alphanumeric characters.	Default = $*CTRL_USER*.
CONTROL_SQL_SERVER	The SQL Server where the Control database resides. If specified, the Control database can reside on a separate SQL Server from the application databases, if required. Maximum length is eight alphanumeric characters.	Default = $*DSQUERY*.

RPC-Related Parameters

The following parameters configure the number of retries and the time interval between each RPC retry. The default parameter values are usually sufficient.

Table 9–7 lists and describes the RPC-related parameters and values.

Table 9–7 RPC-related Parameters, and Values

Parameter	Description	Value
RPC_ERROR_SLEEP	The time interval, in seconds, of the RPC retries when an error condition other than database deadlock is encountered (for example, transaction logs full). This time interval should be large enough to allow the system administrator time to determine the problem and resolve the problem between retries.	Default = 300.
RPC_RETRY_COUNT	The number of times the RPC is retried by the RPC API functions. If after the number of retries the RPC is still unsuccessful, the calling function returns a failure condition, and the API issues a system event.	Default = 5.
RPC_RETRY_SLEEP	The sleep interval, in seconds, for the RPC retries for the database deadlock condition.	Default = 1.
RPC_SHOW_OUTGOING_ PARAMS	N/A	Default = 1.

Network Connection-Related Parameters

The following parameters are referenced whenever an application process opens a network connection to either an application server or SQL Server. The default settings are usually sufficient and rarely require modifications.

Table 9–8 lists and describes the network connection-related parameters and values.

Table 9–8 Network Connection-Related Parameters, and Values

Parameter	Description	Value
MAX_CONN_PER_CONTEXT	The number of Client and Server (CS) library connections that may be opened per context. It may be specified if using a global context; that is, USE_GLOBAL_ CONTEXT = 1.	Default = 25.
MAX_CONNECT_RETRY	The maximum number of retry attempts an application will perform when attempting to asynchronously connect to a TCP/IP server using the **UNIX connect()** API call. These retries may be necessary as the application can be interrupted during the connection attempt. This parameter is generally not set to a value other than its default.	Default = 10.
SOCK_REUSEADDR	Boolean flag. If set to 1, will set the re-use address TCP/IP option on a socket when acting as a TCP/IP socket listener.	Default = 1.
USE_GLOBAL_CONTEXT	This specifies whether each Client (CT)-Library connection should use the global context or allocate a context for each connection. See also MAX_CONN_PER_ CONTEXT.	Default = 0.

Application Diagnostics-Related Parameters

There are two sets of files maintained by ASAP applications for monitoring and troubleshooting purposes: log files and diagnostic files. These files are created by ASAP daily or whenever an application starts up.

Table 9–9 lists and describes the application diagnostics-related parameters and values.

Table 9–9 Application Diagnostics-Related Parameters, and Values

Parameter	Description	Value
CONNECT_ERROR_ LOGGING	Boolean flag. If set to 1, causes applications to log messages to their diagnostic files upon unsuccessful connection attempts to servers.	Default = 0.
DIAG_FILE_COUNT	When the current diagnostic file associated with an ASAP server reaches its maximum size (see MAX_DIAG_FILE), it creates a new current diagnostic file, and rolls over the old diagnostic file with a time stamped files. The DIAG_FILE_ COUNT parameter specifies the number of time stamped diagnostic files to maintain in addition to the current file. In addition, if the ASAP server or process associated with the diagnostic files stops and restarts, ASAP creates new time stamped files, keeping the old time stamped files without updating them.	Default = 1.
DIAG_LINE_FLUSH	Boolean flag. Determines whether the diagnostic file output is flushed to disk at the end of each diagnostic line. If set to 0, the diagnostics are only flushed to the diagnostic file if a PROGRAM or SANITY level diagnostic message is written, or if the I/O buffer is flushed by the operating system. You can change this diagnostic parameter by either adjusting this configuration parameter or changing the diagnostic line flush flag of the application server by using the **diag_line_ flush()** API RPC/Registered Procedure to the particular server.	Default = 0. This option should be set to 0 in a production environment because the option causes considerable performance overhead in disk activity.
MAX_DIAG_FILE	Application process diagnostic file size in megabytes. It is located in the $*LOGDIR/yymmdd* directory, where $*LOGDIR* is $*ASAP_BASE*/DATA/logs. After this file size is reached, this file can be copied to a time stamped file and a new diagnostic file opened. You can modify the diagnostic file growth by either adjusting this configuration parameter or modifying the diagnostic level of the application server (statically in the Control database or, if a server process, dynamically by means of the **diag_level()** API RPC/Registered Procedure). For more information on diagnostic levels, see *ASAP System Administrator's Guide*.	Default = 3.
OS_ERROR_LOGGING	Boolean flag. If set to 1, logs operating system client messages to the diagnostic file. This should be set to 0 in most implementations.	Default = 0.

Self-Balancing Binary Tree-Related Parameters

The following parameter is referenced whenever an application process initializes.

Table 9–10 lists and describes self-balancing binary tree-related parameters and values.

Table 9–10 Self-Balancing Binary Tree-Related Parameters, and Values

Parameter	Description	Value
SBT_ELEMENTS_PER_NODE	The number of self-balancing elements per node within the ASAP self-balancing binary trees. This should generally not be set to a value other than the default. This parameter is referenced whenever an application process initializes.	Default = 3.

Server API Configuration Parameters

This section describes the server API configuration parameters.

Sybase Open Server Parameters

The Sybase Open server configuration parameters can be configured by the server API as the Open server application is starting. The default settings are sufficient for most requirements. However, some specialized application servers may require the tuning of specific parameters.

Table 9–11 lists and describes Sybase open server parameters and values.

Table 9–11 Sybase Open Server Parameters, and Values

Parameter	Description	Value
API_CHECK	Boolean flag. If set to 1, it performs validation of Open server library arguments and state checking.	Default = 0.
ATTN_ON_ DISCONNECT	Boolean flag. If set to 1, it calls the Open server's SRV_ ATTENTION handler when a client disconnects.	Default = 0.
DEFERRED_ QUEUE_SIZE	Open server deferred event queue size.	Default = 1024.
LOG_SIZE	The maximum size, in bytes, of the Open server log file. Open server moves the current contents of the log file to *currentfilename*_**old** and truncates the current log to 0 bytes.	Default = 4194304. The numbers and sizes of the ASAP diagnostic files are controlled by the MAX_DIAG_FILE and DIAG_FILE_SIZE configuration parameters respectively.

Table 9–11 (Cont.) Sybase Open Server Parameters, and Values

Parameter	Description	Value

MAX_ CONNECTIONS	This parameter is used for tuning. The maximum number of physical network client connections the Open server accepts. It should be set high enough to accommodate all connections from other ASAP servers in addition to ad hoc connections from client programs such as sqlplus. The actual number of connections possible depends on the platform dependencies and resources.
	This parameter sets the maximum number of client connections that are accepted by a component server. The actual number is based on platform dependencies and resources. The configured value of MAX_CONNECTIONS must be high enough to accommodate all connections from other ASAP servers, as well as spontaneous connections from client-built programs, such as sqlplus.
	If the configured number of connections is exceeded, the Open server returns error message 16133.
	`> > 173815059:1:PROG:Server Info:1039: main.cSERVER: Information: Error 16133 Severity 10 State 'Configuration of 10 connections has been exceeded, connection rejected'`
	You can run the following sample code from the diagnostic directory to locate error messages in the log file:
	`export TARGET= "Error 16133"`
	`echo "searching for $TARGET (MAX_CONNECTIONS)"`
	`grep "$TARGET" *.diag*`
	Changes in the value of MAX_CONNECTIONS affect the MAX_ MSGQUEUES and the MAX_SERVER_PROCS parameters.
		Default = 30.
		Recommended initial values are based on system size:
		■ Small : 30
		■ Medium : 50
		■ Large : 80

MAX_ MSGQUEUES	The number of thread message queues available to allocate when the Open server starts up. This number does not include message queues created for internal use by the Open server run-time system. Message queues are created for each thread spawned by a component and for each client connection. The configured value of MAX_MSGQUEUES must be greater than or equal to the sum of the values of MAX_THREADS and MAX_ CONNECTIONS. Changes in the value of this parameter affect the MAX_ MSGPOOL parameter. Larger values are required only for the SARM section of the configuration file. The small value can be used for all other ASAP servers regardless of the size of the ASAP system.	Default = 278 Recommended initial values are based on system size: ■ Small : 278 ■ Medium : 428 ■ Large : 628
MAX_MSGPOOL	This parameter is used for tuning. The number of message structures available to the Open server. This parameter sets the number of message structures to be allocated at the time of start up. This number must always be greater than or equal to the sum of MAX_MSGQUEUES times 256.	Default = 71168. Recommended initial values are based on system size: ■ Small : 71168 ■ Medium : 109568 ■ Large : 160768

Table 9–11 (Cont.) Sybase Open Server Parameters, and Values

Parameter	Description	Value
MAX_MUTEXES	The number of mutually exclusive semaphores available to the application. Determine this number as follows: 110 + the number of NEPs + the number of SRPs + MAX_WO_MGRS (refer to **ASAP.cfg**). At a minimum, the number of mutexes would be 117 (110 + 1 NEP + 1 SRP + 5 work order managers). In this case, Oracle recommends that you accept the default. If you are operating multiple NEPs, SRPs, and work order managers, you may want to increase the default value. See *ASAP Developer's Guide* for information on mutexes.	Default = 128.
MAX_REMBUF	The window size used in server-to-server communications and indicates the maximum number of packets that can be outstanding on a logical subchannel before an acknowledgment is required.	Default = 15.
MAX_SERVER_PROCS	This parameter is used for tuning. This parameter sets the maximum number of thread IDs for the application server. The server API maintains an array of administrative thread information for each element, which is indexed using thread IDs. If the thread IDs in the server exceed the value of MAX_SERVER_PROCS, the server terminates. Set MAX_SERVER_PROCS equal to MAX_THREADS plus MAX_CONNECTIONS plus MAX_REMSITES plus 68 (the number of reserved Sybase internal threads). Larger values are required only for the SARM section of the configuration file. The smaller value can usually be used for all other ASAP servers regardless of the size of the ASAP system. In addition, you can set the Shrink Frequency value to non-zero (to allow shrinking) for the RPCConnectionPool in WebLogic Server. See "Socket Connections" to see how to access the **Configuration** tab. The **Shrink Frequency** parameter can be found on the **Connection Pool** sub-tab, under **Advanced**. In the WebLogic Server configuration, set RPC connection to SARM from dynamic control to static connection to disable shrinking.	Default = 512

Table 9–11 (Cont.) Sybase Open Server Parameters, and Values

Parameter	Description	Value

MAX_THREADS	This parameter is used for tuning. The maximum number of threads that may be spawned in the application (including the server API). It does not include threads spawned internally within the Sybase Open server library.
	The value of MAX_THREADS should be less than or equal to the value of MAX_SERVER_PROCS.
	This parameter sets the maximum number of threads in the ASAP component.
	Larger values are only required for the SARM section of the configuration file. Small values can usually be used for all other ASAP component servers regardless of the size of the ASAP system.
	The value of MAX_THREADS must be greater than MAX_CONNECTIONS because each incoming client connection will spawn a thread.
	If the number of threads that the ASAP server attempts to spawn exceeds the configured value of this parameter, then the Open server returns the following error message 16115:
	`>> 173532841:7:PROG:Server Info :1039: main.c`
	`SERVER: Information: Error 16115 Severity 10 State 0 'Could not start thread'`
	`>> 173532843:7:SHUT:System Event :158: asc_spawn.c`
	`ASAP System Event: SYS_TERM`
	`Error: Unable to Spawn Service Thread WO Prov 1 - Insufficient Resources`
	You can run the following sample code from the diagnostic directory to locate error messages in the log file:
	`export TARGET= "Error 16115"`
	`echo "searching for $TARGET (MAX_THREADS)"`
	`Grep "$TARGET" *.diag*`
	Changes in the value of MAX_THREADS affect the MAX_MSGQUEUES and the MAX_SERVER_PROCS parameters.
		Default = 150.
		Recommended initial values are based on system size:
		■ Small : 150
		■ Medium : 300
		■ Large : 500

MAX_USEREVENTS	The maximum number of user events that an Open server may define. These are not generally used in ASAP applications.	Default = 0.
MAX_REMSITES	The maximum number of Open server site handlers that can run at a time. Each site handler can support multiple connections to a single remote SQL Server. When a remote server attempts to log into the Open server and there are no available site handlers, the connection request fails.	Default = 10.
NET_BUF_SIZE	The maximum size of the network I/O buffer to be used by client applications. The actual size of the network buffer used is determined at login time. If a smaller size is requested, Open server will not resize the memory buffer.	Default = 2048.
NET_TRACE_FILE	File to which Sybase Net library will write trace information if so configured.	Default = NULL.
PRE_EMPTIVE	Boolean flag. If set to 1, it causes Open server to use pre-emptive thread scheduling. Support for pre-emptive scheduling is platform-specific.	Default = 0.
SRV_ATTENTION	N/A	Default = 0.

Table 9–11 (Cont.) Sybase Open Server Parameters, and Values

Parameter	Description	Value
STACK_SIZE	The size (in MB) of the stack allocated for each thread. This size can be increased should a thread in an Open server run out of stack space. Increasing the thread stack size will impact the amount of memory required by the server.	Default = 256000.
TIME_SLICE	The number of clock ticks an active thread consumes before the time slice callback routine is called. The callback routine for this state transition is called when a thread has run for a time slice determined by the TIME_SLICE, VIRTUAL_CLK_RATE, and VIRTUAL_TIMER Open server properties.	Default = 10.
TRUNCATE_LOG	Boolean flag. If set to 1, it causes the Open server to truncate its server log file upon start up. **Note:** It does not affect the ASAP diagnostic file.	Default = 0.
USE_SRV_LANG	Boolean flag. If set to 1, it causes the Open server's national language to be used for Open server error messages. If set to 0, the client's national language is used for error messages.	Default = 0.
VIRTUAL_CLK_ RATE	The Open server virtual clock rate, in microseconds per tick.	Default = 1000000.
VIRTUAL_TIMER	Boolean flag. If set to 1, it enables the Open server virtual timer.	Default = 0.

Sybase Open Server Debugging Trace Flag Parameters

The following configuration parameters are included to facilitate dynamic enabling of the Open server diagnostics within the Open server library. They are referenced only when you require detailed debugging output. Therefore, the default values for these flags are sufficient in most environments.

Table 9–12 lists and describes the Sybase Open Server debugging trace flag parameters and values.

Table 9–12 Sybase Open Server Debugging Trace Flag Parameters, and Values

Parameter	Description	Value
SRV_TR_ATTN	Boolean Trace flag. If set to 1, the Open server writes information about attention signals from the client to standard error and to the server logfile.	Default = 0.
SRV_TR_DEFQUEUE	Boolean Trace flag. If set to 1, the Open server writes information about deferred queues to standard error and the server log file.	Default = 0.
SRV_TR_EVENT	Boolean Trace flag. If set to 1, the Open server writes information about event information to standard error and the server log file.	Default = 0.
SRV_TR_MSGQ	Boolean Trace flag. If set to 1, the Open server writes information about message queues to standard error and the server logfile.	Default = 0.
SRV_TR_NETDRIVER	Boolean Trace flag. If set to 1, the Open server writes information about network drivers to standard error and the server log file.	Default = 0.

Table 9–12 (Cont.) Sybase Open Server Debugging Trace Flag Parameters, and Values

Parameter	Description	Value
SRV_TR_NETREQ	Boolean Trace flag. If set to 1, the Open server writes information about network requirements to standard error and the server log file.	Default = 0.
SRV_TR_NETWAKE	Boolean Trace flag. If set to 1, the Open server writes information about network wake-ups to standard error and the server log file.	Default = 0.
SRV_TR_TDSDATA	Boolean Trace flag. If set to 1, the Open server writes Tabular Data Stream (TDS) packet contents in hexadecimal (HEX) and ASCII to standard error and the server logfile. This displays the actual data between the client and the server.	Default = 0.
SRV_TR_TDSHDR	Boolean Trace flag. If set to 1, the Open server writes protocol packet header information (such as packet type and length) to standard error and the server logfile.	Default = 0.

Application Server Memory Management Parameters

The following parameters facilitate the configuration of the application server memory management. To use this memory management capability, you must use the memory management API routines (in particular, the creation and use of memory pools) provided as part of the server API.

Table 9–13 lists and describes the application server memory management parameters and values.

Table 9–13 Application Server Memory Management Parameters, and Values

Parameter	Description	Value
MEMORY_LOGGING	Status flag. Determines whether to create diagnostic memory management RPCs (**mem_usage()** and **mem_stats()**) in the server as well as the degree of memory management diagnostic logging to be performed while the server is running.	Default = 0. The values are: ■ 0 – No logging and no RPCs ■ 1 – Install RPCs but no logging ■ 2 – Install RPCs and provide logging of all memory management outside the server initialization (that is, in steady state) ■ 3 – Install RPCs and provide logging of all memory management activity for the duration of the server
ASC_BLOCK##_POOL	Number of byte memory blocks to allocate within the memory pool for use by the application server. ## is a variable that represents: 16, 32, 64, and so forth, up to 1048576.	The **ASAP.cfg** file defines the following default values: ■ ASC_BLOCK16_POOL= 1024 # Num of 16 byte memory blocks in mem pool ■ ASC_BLOCK32_POOL= 1024 # Num of 32 byte memory blocks in mem pool ■ ASC_BLOCK64_POOL= 1024 # Num of 64 byte memory blocks in mem pool ■ ASC_BLOCK96_POOL= 1024 # Num of 96 byte memory blocks in mem pool ■ ASC_BLOCK128_POOL= 512 # Num of 128 byte memory blocks in mem pool ■ ASC_BLOCK256_POOL= 256 # Num of 256 byte memory blocks in mem pool ■ ASC_BLOCK512_POOL= 128 # Num of 512 byte memory blocks in mem pool ■ ASC_BLOCK1024_POOL= 64 # Num of 1024 byte memory blocks in mem pool ■ ASC_BLOCK2048_POOL= 32 # Num of 2048 byte memory blocks in mem pool ■ ASC_BLOCK4096_POOL= 16 # Num of 4096 byte memory blocks in mem pool ■ ASC_BLOCK8192_POOL= 8 # Num of 8192 byte memory blocks in mem pool

Client Library Parameters

The APPL_POOL_SIZE and CONTROL_POOL_SIZE parameters relate to various Client library attributes that can be specified within the application server.

Note: ASAP no longer supports DB library as a client API.

The number of client library connections that the application server opens to its primary application database in the APPL_SQL_SERVER SQL Server with user APPL_USERID is calculated based on the number of threads running in the application.

Table 9–14 lists and describes the client library parameters and values.

Table 9–14 Client Library Parameters, and Values

Parameter	Description	Value

CONTROL_POOL_SIZE	This parameter is used for tuning. This parameter sets the number of CT-Library connections to the Control database.
	If the configured number is not sufficient, the following error message is logged:
	`>> 181540.988:41:PROG:ASC cppalloc :153: ctlib.c Waiting for Connection from 'Control Pool', (2 In Use). Auto-tuning will increase connection pool if problems persist.`
	You can run the following sample code from the diagnostic directory to locate error messages in the log file:
	`export TARGET= "Waiting for connection from 'Control Pool'"`
	`echo "searching for $TARGET (APPL_POOL_SIZE)"`
	`grep "$TARGET" *.diag*`
		Default = 2.
		Recommended initial values are based on system size:
		■ Small : 5
		■ Medium : 7
		■ Large : 10

APPL_POOL_SIZE	This parameter is used for tuning. This parameter sets the number of CT-Library connections to the component database that you are tuning.
	Larger system sizes are required only for the SARM section of the configuration file. The smaller value can usually be used for all other ASAP servers regardless of the size of the ASAP system.
	If the configured number is not sufficient, the following error message is logged:
	`>> 181540.988:41:PROG:ASC cppalloc :153: ctlib.c Waiting for Connection from 'Application Pool', (2 In Use). Auto-tuning will increase connection pool if problems persist.`
	You can run the following sample code from the diagnostic directory to locate error messages in the log file:
	`export TARGET= "Waiting for connection from 'Application Pool'"`
	`echo "searching for $TARGET (APPL_POOL_SIZE)"`
	`grep "$TARGET" *.diag*`
		Default = 2.
		Recommended initial values are based on system size:
		■ Small : 5
		■ Medium : 25
		■ Large : 100

ASAP_IS_ALIVE_INTERVAL	The Control server can also be configured to issue "Keep Alive" notifications to an external operations center signifying that ASAP is functioning correctly. The absence of this periodic notification may indicate an ASAP outage. ASAP_IS_ALIVE_INTERVAL is the check-time interval, in seconds, during which application servers validate existing connections to the SQL Server. This parameter also determines the time period between connection retries between application servers in the event a server is unavailable for some reason.	Default = 60.

DB Library Parameters

This section describes the DB library parameters.

Table 9–15 lists and describes the DB library parameters and values.

Table 9–15 DB Library Parameters, and Values

Parameter	Description	Value
DBERROR_TIME_INTERVAL	Threshold time frame. See also NUM_DBERROR_ALLOWED.	Default = 60.
MAX_DBPROCS	The maximum number of database processes per application.	Default = 25.
NUM_DBERROR_ALLOWED	DB library error threshold. If more than this number of DB library errors occur in a specified time interval, the application server will stop. This is to avoid recursive error situations. See also DBERROR_TIME_INTERVAL.	Default = 40.

Poll Management Parameters

The following parameters are used by the poll manager thread in the server API, which provides thread-safe polling functionality. The poll manager thread monitors the network connections on behalf of all the threads in the ASAP application server. The server API call, **ASC_poll()**, sends poll requests to the poll manager and returns the same return values as the native, **srv_poll()** Sybase call.

Table 9–16 lists and describes the poll management parameters and values.

Table 9–16 Poll Management Parameters, and Values

Parameter	Description	Value
MAX_CONNECTIONS_TO_POLL	The initial number of network connections for which the poll manager polls. This configuration parameter is used to determine the initial poll list size. This list is automatically resized as additional connections are added to the poll list.	Default = 1.
MAX_POLL_REQUESTS	This specifies the maximum initial number of poll requests to the server API poll manager thread at one time. If more than this number of requests are received, the poll manager automatically resizes the poll request list.	Default = 1.
MAX_POLL_RETRY	This is the number of retries that the server API poll manager will attempt should the Sybase **srv_poll()** call be interrupted in the **ASC_poll()** server API call.	Default = 10.

Database Administration Parameters

You can use the following parameters to configure various aspects of the database administration thread in each application server. This calls a stored procedure in the database and passes it a configurable parameter before recompiling all the stored procedures in that application server's default database (the database defaulted to using the APPL_SQL_SERVER and APPL_USERID login information). It also updates the SQL Server statistics for all indexes on user-defined tables in the database.

Table 9–17 lists and describes the **DB_ADMIN_ON** parameter and values.

Table 9–17 DB_ADMIN_ON Parameter and Values

Parameter	Description	Value
DB_ADMIN_ON	Boolean flag. If set to 1, it enables the database administration thread operation in the application server. This can be disabled in particular servers in situations where multiple servers share the same application database (for example, multiple NEPs) and then only one server is required to perform this database administration.	Default = 0.

IPC Diagnostic Parameters

This section describes the IPC diagnostic parameters.

Table 9–18 lists and describes the IPC Diagnostic parameters and values.

Table 9–18 IPC Diagnostic Parameters and Values

Parameter	Description	Value
LANGUAGE_ DUMP_ON	Boolean flag. If set to 1, it determines whether the language buffer being transmitted or received should be logged as a low-level diagnostic in the server's diagnostic log file. The primary use is for debugging such language buffers, and is therefore, generally used in non-production environments. Enables the logging of diagnostic details of language requests being transmitted to and from application servers.	Default = 1.
REGISTERED_ PROCS	Defines the behavior of a particular API call to create a registered/remote procedure call in an application server. The parameters enable diagnostic output of the RPCs and language requests being transmitted to and from application servers. Boolean flag. If set to 1, it determines whether procedures added with the **add_appl_rpc()** API call are to be added as Sybase registered procedures or as Sybase remote procedures. The advantage to adding them as registered procedures is that there is less overhead in determining the procedure handler because the Sybase Open server library searches the registered procedure handlers before the remote procedure handlers. The disadvantage of using registered procedures is that they do not allow optional parameters. Therefore, any such procedures should be added using the API function, **add_rpc()**, which adds them as remote procedures regardless of this configuration setting. In general, to register a procedure with fixed number of parameters as a registered procedure, call **ASC_define_rpc()**. To register a procedure with variable number of parameters as an RPC, call **add_rpc()**.	Default = 0.

Security-Related Parameters

The following parameters describe security-related parameters.

Table 9–19 lists and describes the Security-Related parameters and values.

Table 9–19 Security-Related Parameters and Values

Parameter	Description	Value
CLIENT_SECURITY_ON	Determines whether client processes connecting to server processes have their user IDs and passwords verified against the SQL Server (APPL_SQL_SERVER SQL). If this connection fails, the client is denied access to the server.	Default = 0.
NE_DIALOG_OFF	Controls the NE dialog message in the diagnostic file.	Default = 0. Possible values are: ▪ 0 – NE dialog messages appear in the diagnostic file. ▪ 1 – Secures the NE dialog. No NE dialog messages appear in the diagnostic file.

High-Availability Parameters

The following parameters identify the system and territory used in ASAP.

Table 9–20 lists and describes the high-availability parameters and values.

Table 9–20 High-Availability Parameters and Values

Parameter	Description	Value
SYSTEM_NAME	Name of the current system.	Default = $*ASAP_SYSTEM*.
TERRITORY_NAME	Name of the current territory.	Default = $*ASAP_TERRITORY*.

Application Server Performance Parameters

Every application server logs process performance parameters which are maintained in memory within the application. The parameters are also written to the database at an interval configured using the PERF_POLL_PERIOD parameter.

Table 9–21 lists and describes the application server performance parameter and value.

Table 9–21 Application Server Performance Parameter and Value

Parameter	Description	Value
PERF_POLL_PERIOD	Time period, in seconds, that the application servers wait to update the Control database with periodic performance information related to the operation of the application server.	Default = 600.

Client API Configuration Parameters

This section describes client API configuration parameters.

Client Application Signal Handling

This section describes client application signal handling.

Table 9–22 lists and describes the client application signal handling parameter and value.

Table 9–22 Client Application Signal Handling Parameter and Value

Parameter	Description	Value
SIG_IGNORED	Boolean flag. If set to 1, it instructs the client application to ignore any received signals that are not explicitly handled by the application. If set to 0, reception of such a signal results in the termination of the client application.	Default = 1.

SRP API Parameters

This section describes SRP API parameters.

SARM Connectivity Parameters

This section describes SARM connectivity parameters.

Table 9–23 lists and describes the SARM connectivity parameters and values.

Table 9–23 SARM Connectivity Parameters and Values

Parameter	Description	Value
DUMP_WO_PATH	The location of the dump file for work orders processed by the SRP Emulator. This parameter works in conjunction with DUMP_WO_FLAG, which specifies whether the work orders processed by the SRP Emulator should be dumped to a dump file.	Default = **/tmp**.
MAX_SARM_DRIVER	The number of SARM driver threads to spawn within the SRP library. These threads send orders to the SARM.	Default = 5.
MAX_WO_MGRS	The number of work order manager threads to be started by the SARM.	Default = 5.
SARM	The logical name of the SARM server to which the SRP will establish network connections.	Default = $*SARM*.
SRP_SEND_GPARMS	Boolean flag. If set to 1, it transmits global parameters on the order to the SARM as global parameters. This parameter is provided for backward compatibility with some existing SRPs.	Default = 1.

Loopback Testing Parameters

The following parameter provides loopback delay functionality in the SRP upon reception of work order event notifications.

Table 9–24 lists and describes the loopback testing parameter and value.

Table 9–24 Loopback Testing Parameter and Value

Parameter	Description	Value
WO_MGR_DELAY	Artificial loopback time delay which, if not zero, results in a pause for the configured number of seconds after SRP processes a work order event notification from the SARM. This is used primarily in testing and should not be configured in a non-test environment.	Default = 0.

Interpreter Operation

The following parameters relate directly to the operation of the Interpreter State Tables.

Table 9–25 lists and describes the interpreter operation parameters and values.

Table 9–25 Interpreter Operation Parameters and Values

Parameter	Description	Value
INSTRUCTIONS_ PER_SLICE	The number of State Table instructions that are consecutively run by the Interpreter before the Interpreter explicitly suspends the thread to let other threads run in a non pre-emptive environment. This is generally only a concern when the Interpreter is running in loopback mode because there is no I/O that would otherwise suspend the thread.	Default = 10.
MAX_STACK_ DEPTH	The maximum depth of the stack in the Interpreter State Tables. For complicated or recursive state tables, this stack depth size may need to be increased to accommodate the largest anticipated stack size. Each Interpreter instance in an application server has its own State Table stack, and therefore, the memory requirements for each Interpreter is multiplied by the number of configured Interpreters within the application server to determine process memory requirements. Each unit increase in the stack depth parameter results in roughly eight extra bytes per Interpreter. If an NEP has 10 Interpreters or command processors and defines its stack depth at 100, the memory requirements would be 8 KB (10 x 100 x 8).	Default = 20.
STRING_ LENGTH_CHECK	Boolean flag. If set to 1, it performs extensive checking of the length of State Table variables. This is useful in situations in which long variables are being extensively manipulated in State Tables and allows the library to perform the validation rather than the State Tables themselves.	Default = 0.
SWITCH_ OPTIMIZATION	Boolean flag. If set to 1, it enables State Table optimizations to be made to the State Tables to improve performance. Switch/case/default statements are load-time compiled into a map that is used when the SWITCH action is called. In this mode of operation, the CASE values are treated as static strings and the CASE, DEFAULT, and ENDSWITCH actions are never called because the SWITCH statements provide all the functionality with the map. Optimization can be enabled in two ways: ▪ Global configuration parameter SWITCH_OPTIMIZATION can be set to 1. ▪ SWITCH_OPT action can be inserted into the State Table as the first action to provide selective optimization by State Table. **Note:** User-defined actions that use the SWITCH_VALUE do not work when optimization is enabled and will cause compilation errors.	Default = 0.

SQL Server Connectivity

Use the following parameters to configure the Interpreter's access to the core ASAP database, which contains the State Tables, the Interpreter's local database (for example, the NEP database if the Interpreter is running in the NEP), and the number of network connections to open to each database.

Table 9–26 lists and describes the SQL Server connectivity parameters and values.

Table 9–26 SQL Server Connectivity Parameters and Values

Parameter	Description	Value
CORE_SQL_ SERVER	The SQL Server where the core ASAP database resides. This allows distribution of the core ASAP database to a separate SQL Server, if required. Maximum length is eight alphanumeric characters.	Default = $*DSQUERY*.
CORE_USERID	The user ID for CORE_SQL_SERVER.	Default = $*SARM_USER*.

MAX_CMD_ DBPROCS	This parameter sets the number of connections to the Interpreter database, where the State Tables use additional tables and stored procedures.
	If the configured number is not sufficient, the following error message is logged:
	`181540.988:71:PROG:Alloc DBPROC:888: cmd_utils.c`
	`Waiting for Command DBPROCESS to become Available`
	You can run the following sample code from the diagnostic directory to locate error messages in the log file:
	`export TARGET= "Waiting for Command DBPROCESS"`
	`echo "searching for $TARGET (MAX_CMD_DBPROCS)"`
	`grep "$TARGET" *.diag*`
		Default = 1.
		Recommended initial values are based on system size:
		■ Small : 1
		■ Medium :3
		■ Large : 5

MAX_CORE_ CPPROCS	The number of Client library network connections the Interpreter establishes to the core database. These connections can then be accessed by action functions within the Interpreter in order to interact with the core database. The NEP library uses these connections to write information directly to the core database. The SRP library uses these connections to access data directly in the core database.	Default = 2.

MAX_CORE_ DBPROCS	This parameter sets the number of connections to the SARM database.
	If the configured number is not sufficient, the following error message is logged:
	`111921.013:42:PROG:Alloc DBPROC :904: cmd_utils.c`
	`Waiting for Core DBPROCESS to become Available`
	You can run the following sample code from the diagnostic directory to locate error messages in the log file:
	`export TARGET= "Waiting for Core DBPROCESS"`
	`echo "searching for $TARGET (MAX_CORE_DBPROCS)"`
	`grep "$TARGET" *.diag*`
		Default = 2.
		Recommended initial values are based on system size:
		■ Small : 2
		■ Medium :3
		■ Large : 5

NEP_SQL_ SERVER	The SQL Server to which the Interpreter provides database access. For an interpreter in an NEP, this is the SQL Server where the NEP database resides. For an interpreter within an SRP, this is the SQL Server where the SRP database resides. Maximum length is eight alphanumeric characters.	Default = $*DSQUERY*.
NEP_USERID	The user ID the Interpreter uses to open network connections to the SQL Server in order to access the Interpreter database. For an interpreter in an NEP, this is the NEP user ID. For an interpreter in an SRP, this is the SRP user ID. The default database for this user is the location of any tables, stored procedures, etc., required by the Interpreter in its processing. Maximum length is 20 alphanumeric characters.	Default = $*NEP_USER*.

State Table Debugger Support

This section describes the State Table debugger support.

Table 9–27 lists and describes the State Table Debugger support parameter and values.

Table 9–27 State Table Debugger Support Parameter and Value

Parameter	Description	Value
DEBUGGER_ON	Boolean flag. You can use the DEBUGGER_ON parameter to enable the Interpreter State Table debugger. If set to 1, it initializes the Interpreter debugging facilities. Refer to the relevant ASAP specification for details about the Interpreter debugger.	Default = 0.

Loopback Support

The following parameters describe how you can configure the Interpreter to run in loopback mode. They are used primarily in system testing.

Table 9–28 lists and describes the loopback support parameters and values.

Table 9–28 Loopback Support Parameters and Values

Parameter	Description	Value
LOOPBACK_ DELAY	If the Interpreter is running in loopback mode, this value represents the number of seconds the interpreter waits before returning a successful Atomic Service Description Layer (ASDL) status to mimic real run-time conditions. **Note:** This is a global parameter over all ASDLs. Specific ASDL settings may be configured through the ASDL response table in the core database.	Default = 5.
LOOPBACK_ON	Boolean flag. If set to 1, it denotes the Interpreter loopback operational status. In addition, the library accesses the ASDL response table in the core database to determine the post-processing emulation action to take. If this parameter appears in the global section of the **ASAP.cfg** file, it sets the global loopback status. Alternatively, it can be put within NEP server sections to manage loopback settings for each NEP server independently. If there is no LOOPBACK_ON parameter configured in **ASAP.cfg**, each NEP server assumes that LOOPBACK_ON is not set (that is, LOOPBACK_ON=0) unless it is specified to be on for that NEP server. Using **asap_utils** option 37, you can dynamically set the value of LOOPBACK_ ON for individual NEs. Changes made using **asap_utils** take effect without the need to restart the ASAP server. You can also query the current settings of LOOPBACK_ON. See options 36 and 37 in "NEP Utilities". A LOOPBACK_ON field is located in the SARM database table tbl_ne_config to control the LOOPBACK mode for each NE individually; There are three possible values for the field: ■ Y – NE is set to LOOPBACK ON ■ N – NE is set to LOOPBACK OFF ■ G – NE's loopback mode depends on the LOOPBACK_ON parameter in **ASAP.cfg**	Default = 1.

Table 9–29 lists all possible NE loopback values according to the values in **ASAP.cfg** and tbl_ne_config.

Table 9–29 NE Loopback Values

ASAP.cfg (LOOPBACK_ON)	tbl_ne_config (loopback_on)	NE LOOPBACK_ON ON/OFF
0 or Not defined	G	OFF
1	G	ON
0 or Not defined	Y	ON
1	Y	ON
0 or Not defined	N	OFF
1	N	OFF

NEP API Parameters

This section describes the NEP API parameters.

ASDL Processing Parameters

The following parameters are used to configure specific aspects of the NEP ASDL processing.

Table 9–30 lists and describes the ASDL processing parameter and values.

Table 9–30 ASDL Processing Parameters and Values

Parameter	Description	Value
ASAP_STATS_ ON	Boolean flag. If set to 1, the NEP saves ASAP statistics logged by the State Tables in the core database statistics table.	Default = 0.
DEFAULT_ ASDL_ESTIM	The default estimate (in seconds) for an ASDL to be processed. This value is used by the NEP to initially estimate the processing times of ASDLs in the NEP.	Default = 10.
NE_CMD_LOG_ ON	Boolean flag. If set to 1, it indicates whether the NEP should save each Man-Machine Language (MML) command issued by the SEND State Table action in the core database for user viewing.	Default = 1.

Connectivity Parameters

The following parameters relate directly to the NEP connectivity logic.

Table 9–31 lists and describes the ASDL processing parameter and values.

Table 9–31 ASDL Processing Parameters and Values

Parameter	Description	Value
CONNECT_ FAIL_DELAY	Time interval, in seconds, the NEP waits between login retries after establishing a successful connection to an external system.	Default = 120.
NEP_ MAINTENANCE _INTERVAL	Time interval, in seconds, before the NEP retries the NE connection after the NEP or State Tables have determined that the NE is in maintenance mode.	Default = 600.
NEP_PORT_ BIND_ INTERVAL	Time interval, in seconds, the NEP waits between attempts to bind to an NEP port in order to establish a connection to an external system. This generally occurs when there are fewer ports available than connection requests and the bind attempt fails. In such circumstances, the binding operation is retried on the assumption that a port will become available.	Default = 60.

Table 9–31 (Cont.) ASDL Processing Parameters and Values

Parameter	Description	Value
NEP_PORT_ CONNECT_ INTERVAL	Time interval, in seconds, between primary connection attempts to an external system. If the NEP is requested to establish a primary connection and that connect attempt fails, this is the time interval the NEP will wait between attempts to establish the primary connection.	Default = 60.
PORT_ENABLE_ TIMER	The time interval, in seconds, after the disabling of an NEP port that the NEP will automatically re-enable the port. A port can be disabled by the NEP if the connect attempt to the external system fails. This avoids manual intervention to re-enable the port in such circumstances.	Default = 600.
SARM	The logical name of the SARM server to which the NEP opens network connections.	Default = $*SARM*.

Switch Direct Parameters

The following parameters relate directly to the NEP support for Switch Direct processing.

Table 9–32 lists and describes the switch direct parameters and values.

Table 9–32 Switch Direct Parameters and Values

Parameter	Description	Value
NEP_HOST_ IPADDR	This is the NEP IP address upon which the NEP is running. It must be set to support Switch Direct functionality. See also SWD_SESSIONS_ SUPPORTED.	Default = $*NEP_HOST_ IPADDR*.
NEP_HOST_ NAME	Host name of the machine on which the NEP is running. See also SWD_ SESSIONS_SUPPORTED.	Default = $*NEP _HOST_NAME*.
SWD_ SESSIONS_ SUPPORTED	Boolean flag. If set to 1, it enables switch direct functionality in the NEP.	Default = 0.

NE Communication API Parameters

This section describes the NE communication API parameters.

Device Driver Support

The following parameters are used to specify the various device drivers that are licensed to the client.

Table 9–33 lists and describes the device driver support parameters and values.

Table 9–33 Switch Direct Parameters and Values

Parameter	Description	Value
DCE_IF_ SUPPORTED	Boolean flag. If set to 1, it enables the DCE option.	Default = 1.
FTP_IF_ SUPPORTED	Boolean flag. If set to 1, it enables the FTP device driver in the communications interface.	Default = 1.
LDAP_IF_ SUPPORTED	Indicates whether LDAP is supported or not.	Default = 1.
SERIAL_IF_ SUPPORTED	Boolean flag. If set to 1, it enables the serial device driver in the communications interface.	Default = 1.

Table 9–33 (Cont.) Switch Direct Parameters and Values

Parameter	Description	Value
SNMP_IF_ SUPPORTED	Boolean flag. Specifies whether the NEP SNMP Option should be enabled. If set to 1, it enables the NEP SNMP option.	Default = 0.
SOCKET_IF_ SUPPORTED	Boolean flag. If set to 1, it enables the TCP/IP socket device driver in the communications interface.	Default = 1.
TELNET_IF_ SUPPORTED	Boolean flag. If set to 1, it enables the TCP/IP telnet device driver in the communications interface. If telnet is supported, the ALLOW_TELNET_EXTENDED_ CHAR_SET Boolean flag specifies whether to print extended ASCII characters.	Default = 1. Possible values are: ■ 1 – All characters (1-255) will print except the NULL character. ■ 0 – Only standard ASCII characters (32-127) are printable.
X25_IF_ SUPPORTED	Boolean flag. If set to 1, it enables the generic device driver in the communications interface.	Default = 1.

Terminal Communication Support

The MAX_GR_CONNECTIONS and MPM_READ_BUF_SIZE parameters are required if the communication to the NE is terminal-based.

Table 9–34 lists and describes the terminal communication support parameters and values.

Table 9–34 Terminal Communication Support Parameters and Values

Parameter	Description	Value
MAX_GR_ CONNECTIONS	Identifies the maximum number of connections that the Generic Router (GR) thread will be polling (including the connection used by the command processors to communicate with the GR). This parameter is used by the GR to set up the initial data structures for polling. If more command processor connection requests are received, the queue is automatically resized.	Default = 20.
MPM_READ_ BUF_SIZE	Specifies the size of the buffer, in bytes, used by the GR thread to read data from terminal based ports. This value can be tuned depending upon the length of the communication.	Default = 512.

Serial Device Driver Support

This section describes the serial device driver support.

Table 9–35 lists and describes the serial device driver support parameters and values.

Table 9–35 Serial Device Driver Support Parameters and Values

Parameter	Description	Value
IGNORE_ MODEM_ST	Boolean flag. If set to 1, it instructs the serial device driver to treat the serial connection as a local direct connection with no modem control. If set to 0, modem control (dial-up) is assumed.	Default = 0.

Generic EDD API Parameters

An external device driver (EDD) is a client application that resides outside the NEP and provides the communication mechanism with external systems. An EDD links in the generic EDD API, libgedd, as well as the client API, libclient, and the common API, libasc.

> **Note:** These parameters are only available to Solaris based ASAP installations when you install the optional SNMP component. For more information about installing this component, see the *ASAP Installation Guide*.

Table 9–36 lists and describes the generic EDD API parameters and values.

Table 9–36 Serial Device Driver Support Parameters and Values

Parameter	Description	Value
HOST_NAME	The host name of the machine that the NEP (that the EDD is communicating with) is on.	Default = $*HOST_NAME*.
IO_TIMEOUT	The I/O timeout, in seconds, the EDD waits before timing out from an I/O write operation to the NEP.	Default = 180.
SERVER_ IPADDR	The IP address of the host machine upon which the EDD is listening for incoming connections.	Default = $*SERVER_ IPADDR*.
SERVER_PORT	The IP port number upon which the EDD is listening for incoming connections from the NEP.	Default = $*SERVER_PORT*.
SNMP_VERSION	Specifies the SNMP version, which will be used for the current EDD. Available versions are V1, V2C, V3.	Default = V1.

CSOL API Parameters

The following parameters are used by the CSOL library Work Order Query component:

Table 9–37 lists and describes the CSOL API parameters and values.

Table 9–37 CSOL API Parameters and Values

Parameter	Description	Value
MAX_QUERY_ RESULTS	A limiting parameter on query result sets. Query results for work orders, CSDL history responses, ASDL history responses and such are limited to this configurable maximum so as not to overload the caller with a potentially large result set. Large result sets can require a lot of memory and CPU resources to maintain and manage.	Default = 250. The default value of 250 is also the maximum value.
QUERY_POOL_ SIZE	Number of database connections to the query database to maintain in the connection pool. The number of simultaneous work order queries supported by the database. The queries exceeding the number will wait until a database connection is available.	Default = 7.
QUERY_SQL_ SERVER	Name of the database server, either Sybase SQL Server or an Oracle database instance, where the QUERY_USERID is located.	Default = $*DSQUERY*.
QUERY_USERID	Database user ID for the C++ library Work Order query component. Normally this is the login name for the SARM database.	Default = $*SARM_USER*.

Auditing Level Parameter

This section describes the auditing level parameter.

Table 9–38 lists and describes the auditing level parameter and values.

Table 9–38 Auditing Level Parameter and Value

Parameter	Description	Value
WO_AUDIT_LEVEL	ASAP provides the ability to audit all transactions involved in provisioning a work order. All work order auditing messages are stored in an audit table in the SARM database (tbl_wo_audit). The user can query work order audit trail records through Order Control Application (OCA). Set the audit level for transactions performed by the work order. See *ASAP Developer's Guide* for more information on ASAP auditing features. If you want to log error messages (SRQ_ERROR_EVENTS) for service requests events (srq_evt) in tbl_srq_log, you must set WO_AUDIT_LEVEL to 2.	Default = 1. Possible values are: ■ 0 – No auditing occurs. No information is placed in the tbl_wo_audit table. ■ 1 – There is one audit entry per work order as it is traced through the system. ■ 2 – Provides all functions of level 1 plus the audit level entries for all error states. ■ 3 – Provides all functions of level 2 plus it tracks the provisioning of a work order through the entire provisioning process. For example, when the ASDL was started, when it was placed in the pending queue, where in the pending queue it is, when it was sent to the NEP, etc. ■ 4 – All events are inserted into the tbl_wo_audit table. This level is intended to debug the work order auditing process. Audit level 4 should not be used in production environments because this level of auditing may result in degraded performance.

WebLogic Server Parameters

This section describes the WebLogic server parameters.

Table 9–39 lists and describes the WebLogic server parameters and values.

Table 9–39 WebLogic Server Parameters and Values

Parameter	Description	Value
BEA_CONN_TYPE	Sets the type of connection that the WebLogic server is expecting (mandatory).	Default = http. Possible values are: ■ http ■ https
BEA_WLS_HOST	Sets the host name or IP address of the system that the WebLogic server is running on. (mandatory).	Default = localhost.
BEA_WLS_PORT	Sets the port number for the WebLogic server. (mandatory).	Default = 7001.
BEA_WLS_USER	Sets the name of the WebLogic admin user login. (mandatory).	Default = system.
SECURITY_SERVICE	Sets the name of the Security Web service deployed in the WebLogic server. (mandatory).	Default = security.

Control Server Configuration Parameters

This section describes the Control server configuration parameters.

Table 9–40 lists and describes the Control server parameter and values.

Table 9–40 Control Server Parameter and Value

Parameter	Description	Value
APPL_ USERID	The primary SQL Server user ID the application process uses to connect to the APPL_SQL_SERVER SQL Server. The user ID defaults to the primary application database, and therefore, the application does not need database names because the definition of the user in the SQL Server specifies that user's default database. Maximum length is 20 alphanumeric characters.	Default = $*CTRL_USER*.

Control Server Alarm Generation

The Control server controls the generation of system alarms by means of the closely coupled fork agent client process. The ALARM_RETRIES and ALARM_TIME_ INTERVAL configuration parameters control specific aspects of the alarm program generation.

Table 9–41 lists and describes the Control server alarm parameters and values.

Table 9–41 Control Server Alarm Parameters and Values

Parameter	Description	Value
ALARM_ RETRIES	The number of times the Control server will attempt to generate an alarm program.	Default = 3.
ALARM_ TIME_ INTERVAL	The time interval, in seconds, between unsuccessful alarm program generations.	Default = 10.

Control Server Database and File System Monitoring

Table 9–42 lists and describes the Control server database and file system monitoring parameters and values.

Table 9–42 Control Server Database and File System Monitoring Parameters and Values

Parameter	Description	Value
DB_MONITOR_ TIME	The time interval, in minutes, when the Control server monitors the database and logs segment sizes and then issues a system event if they exceed a predetermined threshold value. This functionality can be enabled within the Control server by populating the database threshold monitoring table in the Control database.	Default = 10.
FS_MONITOR_TIME	The time interval, in minutes, when the Control server monitors the UNIX file system sizes and issues a system event if they exceed a predetermined threshold value. This functionality can be enabled within the Control server by populating the file system threshold monitoring table in the Control database.	Default = 5.

Table 9–42 (Cont.) Control Server Database and File System Monitoring Parameters and Values

Parameter	Description	Value
RESTART_ ATTEMPTS	The number of times the Control server attempts to restart an ASAP application.	Default = 5.
RESTART_DELAY	The time delay, in seconds, that the Control server waits between attempts to restart an ASAP application.	Default = 60.
RESTART_RESET_ DELAY	The time delay, in seconds, the Control server waits to reset its counters after the last attempt to restart an ASAP application.	Default = 600.

Fork Agent Process Generation Configuration

You can configure the fork agent to retry process generation a number of times before returning a failure. The frequency and number of such attempts is determined by the FORK_ATTEMPTS_INTERVAL and FORK_ATTEMPTS_MAX parameters.

Table 9–43 lists and describes the fork agent process generation configuration parameters and values.

Table 9–43 Fork Agent Process Generation Configuration Parameters and Values

Parameter	Description	Value
FORK_ATTEMPTS_ INTERVAL	The time interval, in seconds, between process generation attempts.	Default = 0.
FORK_ATTEMPTS_ MAX	The maximum number of times the fork agent attempts to generate a new application, either an ASAP application or an alarm program.	Default = 1.
MSGSND_RETRIES	Controls the number of retries if the submission fails. The Control server passes messages to the fork agent using message queues. If the system temporarily cannot submit messages to the queue, the Control server can be configured to retry the submission. Changing the MSGSND_RETRIES and MSGSND_RETRY_DELAY parameters should be done only after diagnostics show that the Control server stops.	Default = 0.
MSGSND_RETRY_ DELAY	Specifies, in milliseconds, the delay between two retries.	Default = 5.

Control Server Database Administration Parameters

You can use these parameters to configure various aspects of the database administration thread in each application server. This calls a stored procedure in the database and passes it a configurable parameter before recompiling all the stored procedures in that application server's default database (the database defaulted to using the **APPL_SQL_SERVER**, and **APPL_USERID** login information). It also updates the SQL Server statistics for all indexes on user-defined tables in the database.

Table 9–44 lists and describes the Control Server database administration parameters and values.

Table 9–44 Control Server Database Administration Parameters and Values

Parameter	Description	Value
DB_ADMIN_ON	Boolean flag. If set to 1, it enables the database administration thread operation in the application server. This can be disabled in particular servers in situations where multiple servers share the same application database and then only one server is required to perform this database administration.	Default = 0.
DB_ADMIN_PROC_ PARAM	The integer parameter passed to the database administration procedure. For example, this can specify a purge interval for a particular database.	Default = 100.
DB_ADMIN_PROC	The procedure the database administration thread calls at a specified time in the day. This procedure could perform many tasks, including archiving and purging of dynamic data. All stored procedures in the database are recompiled and the statistics updated after this administration procedure has been called.	Default = CSP_ db_admin.
DB_ADMIN_TIME	The number of minutes after midnight when the database administration tasks are to be performed. This is usually performed at a time of low system activity.	Default = 300.
DB_PCT_ANALYZE	This parameter applies to Oracle Database only. It is used to update statistics on all user-defined tables. The updates are done when the database administrations tasks are performed. (See also DB_ ADMIN_TIME.) This parameter is used to optimize the database query performance. The Oracle SQL statement is "analyze table table_name estimate statistics sample DB_PCT_ANALYZE percent". See the discussion on the Analyze command in *Oracle SQL Reference Manual*.	Default = 20.

SRP Emulator Server Configuration Parameters

The SRP Emulator is a generic SRP used for performance bench marking, system testing, and the prototyping of new SRPs.

This application server links in libsrp, libinterpret, libcontrol, and libasc. Therefore, it requires the libsrp, libinterpret, libcontrol, and libasc configuration parameter definitions in addition to the ones outlined below.

Note: DUMP_WO_FLAG is specific to the SRP Emulator server and not the SRP.

The SRP Emulator Server can have multiple instances of the following:

- **SARM Driver Threads** – Configurable with the **MAX_SARM_DRIVER** parameter
- **WO Manager Threads** – Configurable with the **MAX_WO_MGRS** parameter
- **SARM RPC Handler Threads** – The connections threads spawned when the SARM open connections to the SRP
- **WO Translation Threads** – Increasing the number of WO translation threads leads to a faster load of work orders in the SRP. To increase the number of WO translation threads you must connect more clients to the SRP because the event-driven threads dedicated to these clients are the WO translation threads themselves.

Table 9–45 lists and describes the SRP Emulator server configuration parameters and values.

Table 9–45 SRP Emulator Server Configuration Parameters and Values

Parameter	Description	Value
APPL_USERID	The primary SQL Server user ID the application process uses to connect to the APPL_SQL_SERVER SQL Server. The user ID defaults to the primary application database, and therefore, the application does not need database names because the definition of the user in the SQL Server specifies that user's default database. Maximum length is 20 alphanumeric characters.	Default = $*SRP_USER*.
DUMP_WO_FLAG	Boolean flag that specifies whether the work orders processed by the SRP Emulator should be dumped to a dump file located in the **/tmp** directory.	Default = 0.
NEP_USERID	The user ID the Interpreter uses to open network connections to the SQL Server in order to access the Interpreter database. For an interpreter in an NEP, this is the NEP user ID. For an interpreter in an SRP, this is the SRP user ID. The default database for this user is the location of any tables, stored procedures, etc., required by the Interpreter in its processing. Maximum length is 20 alphanumeric characters.	Default = $*SRP_USER*.
SAVE_SARM_DATA	Boolean flag that controls the saving of order information by the SRP Emulator when a work order completion or failure is received by the emulator. If set to 1, the emulator will query the SARM for all information about the completed or failed order.	Default = 0.
SRP_EMUL_ORDER_ID	The single character prefix for all work orders created by the SRP Emulator.	Default = A.
USE_RAW_WO_IDS	Boolean flag. If set to 1, dictates that the SRP Emulator use the work order IDs present in the database. If set to 0, the SRP will generate a numerical prefix to the order number and transmit this derived order number to the SARM. This allows the same order definition to be transmitted many times to the SARM by the emulator.	Default = 0.
ZERO_PAD_WO_IDS	Boolean flag. If set to 1, pads the generated work order IDs with zeros.	Default = 0.

SARM Server Configuration Parameters

This section describes SARM server configuration parameter.

Table 9–46 lists and describes the SARM server configuration parameter and values.

Table 9–46 SARM Server Configuration Parameter and Value

Parameter	Description	Value
APPL_USERID	The primary SQL Server user ID the application process uses to connect to the APPL_SQL_SERVER SQL Server. The user ID defaults to the primary application database, and therefore, the application does not need database names because the definition of the user in the SQL Server specifies that user's default database. Maximum length is 20 alphanumeric characters.	Default = $*SARM_USER*.

Mask for WO ID Generation

This section describes mask for WO ID generation.

Table 9–47 lists and describes the mask for WO ID parameter and values.

Table 9–47 Mask for WO ID Parameter and Value

Parameter	Description	Value
OCA_WOID_MASK	Format string that uses a single integer placeholder (%d) to set a mask for generation of work order IDs. The integer field is populated by a unique integer from the SARM.	The default value results in work order IDs beginning with OCA-00000001.Default=OCA-{0,number,*******}.

Configuration for VNO External Validation

The following parameters are used to configure Virtual Network Operator (VNO) status.

Table 9–48 lists and describes the configuration for VNO external validation parameters and values.

Table 9–48 Configuration for VNO External Validation Parameters and Values

Parameter	Description	Value
VNO_ENABLED	This VNO configuration parameter is designed to activate the OCA client to perform VNO security checking on OCA users and to un-authorize users who are not members of any VNO group in WebLogic Server security. The value of this configuration parameter is obtained by the OCA SRP server and is provided to the OCA client.	Default = 0.
VNO_ID_DEFAULT	This parameter controls the specification of a default VNO ID at the OCA SRP. If VNO_ID_DEFAULT is not set, no default VNO ID will be added to work orders, even if they do not include a VNO ID. If VNO_ID_DEFAULT is set to the VNO ID, the VNO ID specified will be added to work orders that do not already have a VNO ID specified. When a default VNO ID is added to an order and the order is submitted successfully, a record is inserted to tbl_usr_wo_prop in the SARM database for that order, with VNO ID in the name field and the default value in the value field.	Default = no value
VNO_ID_STRIP	This parameter controls whether the VNO ID is stripped. If VNO_ID_STRIP is set to 0, the VNO ID will not be stripped before returning the information upstream. If VNO_ID_STRIP is set to 1, the information being returned upstream is filtered by removing the VNO ID. The OCA SRP maintains the VNO ID in its memory and databases for this order. For example, tbl_usr_wo_prop still contains the VNO ID for the order.	Default = 0.
EXTERNAL_ VALIDATION_JNDI	This parameter configures the WO extended validation server. It works only when VNO_ENABLED is set to 1. To enable WO extended validation, set the full JNDI path of external Validation EJB. To disable WO extended validation with VNO filter enabled, do not define this parameter. An external validation sample can be found in $*ASAP_BASE*/**oca_sys_if/sample/validationServer/**. With the sample validation logic, if a work order contains the CSDL C-ADD_POTS_LINE and its parameter DN is not equal to 6742727, the InvalidOrderException message is thrown.	Default = abc/Comp/SampleExternalOrderValidation

SARM Work Order Processing

These following parameters are relevant to the processing of work orders within the SARM.

Table 9–49 lists and describes the SARM work order processing parameters and values.

Table 9–49 SARM Work Order Processing Parameters and Values

Parameter	Description	Value
ASDL_TIMEOUTS	Each ASDL has a timeout value that governs how long it can remain in the pending queue without being successfully provisioned. All ASDLs within the same work order must have the same timeout value. When the work order has timed out and failed, the final status is WO_TIME_OUT. Boolean flag. For SARM to support ASDL timeout, set the following: ■ ASDL_TIMEOUTS to 1 in **ASAP.cfg** (Default = 0) ■ asdl_timeout to desired value on the work order Each ASDL can time out based on the configuration parameters. The timer must be set for each ASDL, and it starts once the ASDL is in pending queue. If the value of the ASDL_TIMEOUTS is 0 (zero), the ASDL timeout feature is not used, regardless of the asdl_timeout value. Timeout and retry attributes are configurable at the ASDL level, at the NE level, at the work-order level, and at the system level. If no ASDL timeout parameters are defined, other settings may apply. See *ASAP Cartridge Development Guide*. The way the ASDL timeout is applied to the first ASDL differs from the way it applies to all subsequent ASDLs. If an ASDL timeout occurs during the provisioning of the first ASDL, a grace period is given. For the second, and subsequent ASDLs, no grace period is given. The value of the grace period is the same as the asdl_timeout value. When the first ASDL times out during provisioning, the ASDL does not fail and the work order remains in an In Progress status. The SARM sends an event notification to the SRP, gives the timed out ASDL a grace period, and continues provisioning. If the same ASDL times out again, the SARM fails the ASDL. Then the work order fails. When both the work order and the ASDL timeouts are set for a particular work order, only the first occurrence of timeout is processed and the other is ignored. The following notification events are triggered to inform the SRP of the state of the work order for both the work order and ASDL timeouts: ■ Executing : Informs the SRP of first ASDL timeout that occurred during provisioning. ■ Failed : Informs the SRP of the work order failure.	Default = 0.
BATCH_SLEEP_ INTERVAL	The time period, in seconds, between SARM database queries for future dated orders that: ■ Are ready to be provisioned ■ Have had a state change (for example, from HELD to INITIAL) ■ Can now be provisioned due to the completion of a PARENT order.	Default = 60.
CORE_SQL_SERVER	The SQL Server in which the SARM database resides. This is the same as APPL_SQL_SERVER for the SARM. Maximum length is eight alphanumeric characters.	Default = $*DSQUERY*.

Table 9–49 (Cont.) SARM Work Order Processing Parameters and Values

Parameter	Description	Value
EXP_ONLY_COMP_ DEFINED_PARM	This parameter controls the evaluation logic of ASDL spawning expressions within the SARM. If set to 1, the (sub)expression returns false if a label is not provided, except for DEF/NOTDEF cases. If set to 0, the legacy logic is used. When a label is not provided, the label name is used as the value of the label in the expression.	Default=0.
FAIL_WO_ON_ EMPTY_CSDL	Boolean flag. If set to 1, the CSDL triggers a work order failure if it fails to spawn any ASDLs.	Default=0.
INPROC_CHK_INT	The time interval, in seconds, the SARM waits between checks of the internal queues to determine whether there are any orders in progress beyond the specified in progress threshold. See also MAX_IN_QUEUE_TM.	Default = 900.
MAX_IN_QUEUE_ TM	This is the time interval, in seconds, for which the SARM allows a work order to be in progress before calling the WOINPROC system event, which notifies the user that at least one work order is still in progress beyond the defined threshold. The order is not automatically failed by this mechanism. For example, a work order that requires provisioning at two different NEs, one of which is presently unavailable, could result in the SARM issuing a system event. See also INPROC_CHK_INT.	Default = 3600.
MAX_ORDERS_IN_ GM_Q	Limits the number of work orders the SARM keeps in memory. During each BATCH_SLEEP_INTERVAL, the Batch Handler attempts to load work orders from the SARM database that are due for provisioning. The Batch Handler does not fetch work orders from the SARM database when the number of new or reloaded messages in the Group Manager common queue exceeds MAX_ORDER_IN_GM_Q. You must correlate MAX_ORDER_IN_GM_Q with BATCH_ SLEEP_INTERVAL. To do this, use the following formula: BATCH_SLEEP_INTERVAL (in seconds) x <max.# of WO per sec.>= MAX_ORDER_IN_GM_Q where: <max.# of WO per sec.> is the maximum number of work orders that the ASAP instance can provision. Examples of typical values: ■ BATCH_SLEEP_INTERVAL: 200 (seconds) ■ <max.# of WO per sec.>: 3.5 WO/s ■ MAX_ORDER_IN_GM_Q: 800 (>200 x 3.5) Each time you add 100 to any of the above values, MAX_ ORDER_IN_GM_Q can increase the amount of memory the SARM uses by up to 1 MB. A smaller value can affect performance during peak hours. If you set a smaller value, the SARM and NEPs request fewer work orders from the SARM database.	Default = 0.

Table 9–49 (Cont.) SARM Work Order Processing Parameters and Values

Parameter	Description	Value
MAX_ORDERS_IN_ PROGRESS	The maximum number of orders that the SARM allows to be in progress at any given time. This is to limit the memory requirements of the SARM should there be large numbers of orders in progress for long periods of time. If zero, this check is disabled. In production environments, Oracle recommends that you set this parameter to 0 because there will then be no upper limit on memory consumption. However, you can protect SARM from running out of memory by setting this value to exceed standard or expected work order loads.	Default = 40.
NUM_TIMES_RETRY	The number of times the SARM sends an ASDL to the NEP to be processed after the NEP returned it with a Fail but Retry status. A work order is failed when the number of retries equals the value specified for NUM_TIMES_RETRY.	Default = 5.
ORDER_TIMEOUT	The number of seconds a particular work order can remain in progress before the SARM fails the order. The work order timer starts after the work order has been submitted and started. This threshold can be exceeded if, for example, the connection to an NE is interrupted after the connection has been established. You can set the system-wide parameter ORDER_TIMEOUT in the **ASAP.cfg** file. If you are using the SRP Emulator, you can also set the wo_timeout value in the tbl_wo_def table in the SRP database. The order timeout behavior is governed by two parameters: the wo_timeout parameter on the work order and the ORDER_TIMEOUT configuration parameter in **ASAP.cfg**.	Default = 0. Possible values are: ■ If wo_timeout has a value greater than 1, it is used. ■ If wo_timeout has a value of 0, work orders do not time out. ■ If wo_timeout has a value less than 0, ORDER_TIMEOUT is used. ■ If wo_timeout has a value less than 0 and ORDER TIMEOUT has a value of 0 or less, work orders do not time out.
RETRY_TIME_ INTERVAL	Time, in seconds, the SARM waits between retries of an ASDL command returned to the SARM with 'Fail but Retry' status.	Default = 120.
SECURITY_CHECK	Boolean flag. Indicates whether security checking is enabled in the SARM. If set to 1, whenever the SARM receives the WO from the SRP, it checks to see if the user ID and password passed on the work order are present in the SARM database user/password table. If so, the work order is accepted, otherwise it is rejected. If security checking is 0, the work order is accepted by default.	Default = 0.

Table 9–49 (Cont.) SARM Work Order Processing Parameters and Values

Parameter	Description	Value
WO_HANDLER_ TIMEOUT	The time interval, in seconds, the SARM waits for the transmission of a work order from the SRP to the SARM before aborting the transaction. If the SRP were to terminate when it is transmitting work orders to the SARM, this condition might occur.	Default = 60.
WO_TIME_ ESTIMATE_ON	Boolean flag. If set to 1, it instructs the SARM to calculate the rough time estimate for completing the work order. If WO estimate notifications back to the SRP are enabled, this parameter should be set. If no WO estimate notifications are required, this parameter should not be set.	Default = 0.
WO_TIMEOUT_ EVENT_TWICE	Boolean flag. If set to 1, then two timeout events are generated, whenever a work order gets times out. If set to 0, then only one timeout event is raised, whenever a work order gets times out. This parameter must be added manually to set it to 0. Otherwise ASAP uses the default setting of 1.	Default = 1

SARM Thread Configuration Management

The following parameters are relevant to the configuration and number of threads within the SARM.

Table 9–50 lists and describes the thread configuration management parameters and values.

Table 9–50 Thread Configuration Management Parameters and Values

Parameter	Description	Value
MAX_GROUP_ MGRS	The number of group manager threads to be started by the SARM upon startup.	Default = 5.
MAX_PROVISION_ HANDLERS	The number of provisioning handler threads to be started by the SARM.	Default = 5.
SRP_READY_WAIT_ TIME	This is the time required by the SRP drivers to complete initializing before SARM starts processing work orders. If this parameter is not present, SARM may not wait for the SRP drivers to complete initializing and may begin processing work orders earlier.	Default = 0.
MAX_SRP_DRIVERS	The number of SRP driver threads to be started by the SARM to communicate with each SRP server.	Default = 2.
MAX_WO_ HANDLERS	The number of work order handler threads to be started by the SARM.	Default = 5.
MAX_WO_MGRS	The number of work order manager threads to be started by the SARM.	Default = 5.

SARM Message Pool Size

This section describes the parameter used for SARM message pool size.

Table 9–51 lists and describes the SARM message pool size parameters and values.

Table 9–51 SARM Message Pool Size Parameters and Values

Parameter	Description	Value
MAX_MSGPOOL	This parameter is used for tuning. The number of message structures available to the Open server. This parameter sets the number of message structures to be allocated at the time of start up. This number must always be greater than or equal to the value of MAX_MSGQUEUES times 256. The SARM requires a larger message pool size approximately 50% larger than the message pool size configured for Sybase Open servers, see "SARM Message Pool Size".	Default = 106752. Recommended initial values are based on system size: ■ Small : 106752 ■ Medium :164352 ■ Large : 241152

SARM Batch Error Thresholds

SARM batch error thresholds are configured using the BATCH_DELETE_DELAY and BATCH_THRESHOLD SARM parameters.

Table 9–52 lists and describes the SARM batch error thresholds parameters and values.

Table 9–52 SARM Batch Error Thresholds Parameters and Values

Parameter	Description	Value
BATCH_DELETE_ DELAY	After all the orders in the batch group are processed, the SARM does not delete the batch group until the time interval that is specified by the configuration parameter expires. Orders are released from the batch group upon completion or batch failure. Any order in the same batch group that is sent after the time interval expires is not treated as part of the batch. You can specify an error threshold for the number of failures that can occur in the batch before the SARM stops activation of that batch. The batch failure threshold value is based on the SARM configuration parameter and is not configurable by the SRP using the Batch Order Delay fail threshold. The time interval, in seconds, after the processing of the last order in the batch, that the SARM waits to remove the batch details from memory.	Default = 60.
BATCH_ THRESHOLD	Defines the maximum number of errors that can occur within a batch of loosely coupled requests. Batch orders are defined by means of the SRP batch order property that identified the batch to which the order belongs. Should the batch error threshold be exceeded for a particular batch, the batch as a whole will be suspended. This parameter is set for all batch work orders. You cannot set this parameter on a per work order basis.	Default = 0.

SARM International Messages

The LANGUAGE_OF_MSG and MESSAGE_NUMBERS parameters provide support for SARM internationalized messages.

Table 9–53 lists and describes the SARM international messages parameters and values.

Table 9–53 SARM International Messages Parameters and Values

Parameter	Description	Value
DB_MONITOR_ TIME	After all the orders in the batch group are processed, the SARM does not delete the batch group until the time interval that is specified by the configuration parameter expires. The time interval, in minutes, when the Control server monitors the database and logs segment sizes and then issues a system event if they exceed a predetermined threshold value. This functionality can be enabled within the Control server by populating the database threshold monitoring table in the Control database. Default = 20.	Default = 60.
LANGUAGE_OF_ MSG	The language code to determine the language in which the Service Request (SRQ) Log messages are written to the database. These messages are detailed in message conversion tables in the SARM database and allow substitution of data fields in their expansion. To use a language other than the default, insert all the relevant messages into this table with the new language code, while ensuring the same placeholder positions within the data string. Any new messages added to the SARM are added under the default language.	Default = USA.
MAX_MAINT_ INTERVAL		Default = 0.
MESSAGE_ NUMBERS	Boolean flag. If set to 1, it causes the SARM to write only the message IDs from the message conversion table to the SARM database. This is used primarily in system testing and should be set to 0 in all other environments.	Default = 0.
SOURCE_ROUTING		Default = 1.

OCA Work Order Entry

This section describes the OCA work order entry.

Table 9–54 lists and describes the OCA work order entry parameters and values.

Table 9–54 OCA Work Order Entry Parameters and Values

Parameter	Description	Value
USER_UNID_1	Must be set to OCA to enable the OCA_WOID_MASK parameter.	Default = OCA.

UNID Manager

This section describes the UNID manager.

Table 9–55 lists and describes the UNID manager parameters and values.

Table 9–55 UNID Manager Parameters and Values

Parameter	Description	Value
ASDL_LOG_UNID_ JUMP_INTERVAL	Number of ASDL LOG UNIDs to cache per work order. Reduces number of SLAVE to MASTER UNID fetch RPCs called.	Default = 5.
UNID_JUMP_ INTERVAL	Number of unique integers to cache in the master SARM. The larger the value, the fewer stored procedures are called to the SARM databases. This can cause UNIDs to be unused in the event that the SARM is shut down.	Default = 1000.

SARM Switch Direct

These parameters provides direct switch access capability, which must be configured in both the SARM and the NEP.

Table 9–56 lists and describes the SARM switch direct parameters and values.

Table 9–56 SARM Switch Direct Parameters and Values

Parameter	Description	Value
SWD_HOST_ IPADDR	Used for Switch Direct support. This is the Host IP Address for the machine where the SARM is running.	Default = $*SWD_HOST_ IPADDR*.
SWD_HOST_NAME	Used for Switch Direct support. This is the Host Name for the machine where SARM is running.	Default = $*SWD_HOST_ NAME*.
SWD_IDLE_ TIMEOUT	Used for Switch Direct support. This is the idle timeout period, in seconds, for automatically disconnecting the Switch Direct Interface (SWD) session if no activity is detected.	Default = 120.
SWD_LISTEN_PORT	Used for Switch Direct support. The TCP/IP port number for the SARM server for Switch Direct support. The SWD client connects to this port.	Default = $*SWD_ LISTEN_PORT*.
SWD_SESSIONS_ SUPPORTED	Boolean flag. If set to 1, it determines whether Switch Direct sessions are supported by the SARM. This parameter is also used by NEP applications.	Default = 0.

Admin Server Parameters

This section describes the admin server parameters.

Table 9–57 lists and describes the Admin server parameters and values.

Table 9–57 Admin Server Parameters and Values

Parameter	Description	Value
CORE_SQL_SERVER	The SQL Server in which the SARM database resides. This is the same as APPL_SQL_SERVER for the SARM. Maximum length is eight alphanumeric characters.	Default = $*DSQUERY*.
CORE_USERID	The user ID for the CORE_SQL_SERVER.	Default = $*SARM_USER*.
SARM	The logical name of the SARM server to which the SRP will establish network connections.	Default = $*SARM*.

Persistent ADM Data in SARM

This section describes persistent ADM data in SARM.

Table 9–58 lists and describes the Persistent ADM data in SARM parameters and values.

Table 9–58 Persistent ADM Data in SARM Parameters and Values

Parameter	Description	Value
ADM_SQL_SVR	The database where ADM tables reside.	Default = $*DSQUERY*.
ADM_USER	The database user ID for ADM_SQL_SVR.	Default = $*ADM_USER*.
LOAD_ PERSISTENT_DATA	Boolean flag. If set to 1, it loads the last known data from the ADMIN database into the SARM's memory. This prevents jumps or gaps in the statistical data.	Default = 1.

Socket Connections

This section describes the socket connections.

Table 9–59 lists and describes the socket connections parameters and values.

Table 9–59 Socket Connections Parameters and Values

Parameter	Description	Value
MAX_CLIENT_CONN	This parameter defines the maximum number of socket connections that the SARM can create to receive RPC requests from the SRP. This value can be tuned to meet the actual need of the connections. One connection corresponds to one thread in SARM, so this value should be less than MAX_SERVER_PROCS (default = 512) Also, the maximum capacity of the connection pool is defined in the jconnector in the JSRP and should not exceed the value of MAX_CLIENT_CONN. To configure the Maximum Capacity parameter through the WebLogic Server Administration Console: 1. In the **Oracle WebLogic Server 11g Administration Console**, click **Lock & Edit** if not already clicked. See the Administration Console Online Help for more information. 2. In the **Domain Structure** panel of the **Change Center** in the **Oracle WebLogic Server 11g Administration Console**, expand **Services**. 3. In the **Domain Structure** panel expand **JDBC** and then select **Data Sources**. The Summary of JDBC Data Sources screen appears. 4. Click the name of the data source to be configured. 5. Select the **Configuration** tab, and the **Connection Pool** sub-tab. 6. Set **Maximum Capacity** to 50. 7. Click **Save**. 8. Click **Release Configuration**. You can determine whether the maximum connections threshold is reached through the SARM diagnostic file.	Default = 100.

Database Administration Parameters for the SARM DB

Use the following parameters to configure various aspects of the database administration thread in each application server. This calls a stored procedure in the database and passes it a configurable parameter before recompiling all the stored procedures in that application server's default database (the database defaulted to using the **APPL_SQL_SERVER**, and **APPL_USERID** login information). It also updates the SQL Server statistics for all indexes on user-defined tables in the database.

Table 9–60 lists and describes the SARM database administration parameters and values.

Table 9–60 SARM Database Administration Parameters and Values

Parameter	Description	Value
DB_ADMIN_ON	Boolean flag. If set to 1, it enables the database administration thread operation in the application server. This can be disabled in particular servers in situations where multiple servers share the same application database and then only one server is required to perform this database administration.	Default = 0.
DB_ADMIN_PROC_ PARAM	The integer parameter passed to the database administration procedure. For example, this can specify a purge interval for a particular database.	Default = 100.
DB_ADMIN_PROC	The procedure the database administration thread calls at a specified time in the day. This procedure could perform many tasks, including archiving and purging of dynamic data. All stored procedures in the database are recompiled and the statistics updated after this administration procedure has been called.	Default = SSP_db_admin.
DB_ADMIN_TIME	The number of minutes after midnight when the database administration tasks are to be performed. This is usually performed at a time of low system activity.	Default = 300.
DB_PCT_ANALYZE	This parameter applies to Oracle Database only. It is used to update statistics on all user-defined tables. The updates are done when the database administrations tasks are performed. (See also DB_ADMIN_TIME.) This parameter is used to optimize the database query performance. The Oracle SQL statement is "analyze table table_name estimate statistics sample DB_PCT_ANALYZE percent". See the discussion on the Analyze command in *Oracle SQL Reference Manual*.	Default = 20.

NEP Server Configuration Parameters

This section describes NEP server configuration parameters.

Table 9–61 lists and describes the NEP server configuration parameters and values.

Table 9–61 NEP Server Configuration Parameters and Values

Parameter	Description	Value
APPL_USERID	The primary SQL Server user ID the application process uses to connect to the APPL_SQL_SERVER SQL Server. The user ID defaults to the primary application database, and therefore, the application does not need database names because the definition of the user in the SQL Server specifies that user's default database. Maximum length is 20 alphanumeric characters.	Default = $*NEP_USER*.
AUTOMATIC_ BLACKOUT_CHECK	If set to greater than 0, the NEP will check for NE blackout prior to connection and State Table execution.	Default = 0.
CACHE_ BLACKOUT_TABLE	If set to greater than 0, the NEP will cache the blackout table and check blackouts using this cached table. If set to 0, the table will not be cached and the database will be queried when blackout is checked. This parameter influences only automatic blackout checking.	Default = 1.
LOAD_JCLASS_ FROM_DB	Indicates that the provisioning and connection classes should be loaded from: ■ Database (1) ■ Local file system (0)	Default = 0.
NEP_USERID	The user ID the Interpreter uses to open network connections to the SQL Server to access the Interpreter database. For an interpreter in an NEP, this is the NEP user ID. For an interpreter in an SRP, this is the SRP user ID. The default database for this user is the location of any tables, stored procedures, etc., required by the Interpreter in its processing. Maximum length is 20 alphanumeric characters.	Default = $*NEP_USER*.
RESPONSE_LOG_ FILE_APPEND	If set to 0, the Java-enabled NEP (JNEP) does not append a new response log to the existing log file with the same name when the **startResponseLog()** method is called. This parameter is used in only JNEP and does not take affect the response log in the State Table.	Default = 1, could be absent in the **ASAP.cfg** file.
PRE_EMPTIVE	Boolean flag. If set to 1, you can enable this flag in the NEP to allow the NEP to use pre-emptive threading, which improves NEP efficiency. Note that pre-emptive threading is CPU-intensive.	Default = 0.

ADM Server Configuration Parameters

This section describes the ADM server configuration parameters.

Table 9–62 lists and describes the ADM server configuration parameters and values.

Table 9–62 ADM Server Configuration Parameters and Values

Parameter	Description	Value
APPL_USERID	The primary SQL Server user ID the application process uses to connect to the APPL_SQL_SERVER SQL Server. The user ID defaults to the primary application database, and therefore, the application does not need database names because the definition of the user in the SQL Server specifies that user's default database. Maximum length is 20 alphanumeric characters.	Default = $*ADM_USER*.
COPY_DOWN_DATA	Boolean flag. If set to 1, writes information to the performance-related tables with the value of D in the Record Type field.	Default = 0.
POLL_TIMER_ASDL	Polling time, in minutes, for querying ASDL-related statistics from the SARM.	Default = 30.
POLL_TIMER_CSDL	Polling time, in minutes, for querying CSDL-related statistics from the SARM.	Default = 30.
POLL_TIMER_NE	Polling time, in minutes, for querying network-element-related statistics from the SARM.	Default = 30.
POLL_TIMER_NE_ ASDL	Polling time, in minutes, for querying network-element/ASDL-related statistics from the SARM.	Default = 30.
POLL_TIMER_ ORDER	Polling time, in minutes, for querying order-related statistics from the SARM.	Default = 30.

Database Administration Parameters for the ADMIN Database

Use the following parameters to configure various aspects of the database administration thread in each application server. This calls a stored procedure in the database and passes it a configurable parameter before recompiling all the stored procedures in that application server's default database (the database defaulted to using **APPL_SQL_SERVER**, and **APPL_USERID** login information). It also updates the SQL Server statistics for all indexes on user-defined tables in the database.

Table 9–63 lists and describes the Admin database administration parameters and values.

Table 9–63 Admin Database Administration Parameters and Values

Parameter	Description	Value
DB_ADMIN_ON	Boolean flag. If set to 1, it enables the database administration thread operation in the application server. This can be disabled in particular servers in situations where multiple servers share the same application database and then only one server is required to perform this database administration.	Default = 0.
DB_ADMIN_PROC_ PARAM	The integer parameter passed to the database administration procedure. For example, this can specify a purge interval for a particular database.	Default = 100.

Table 9–63 (Cont.) Admin Database Administration Parameters and Values

Parameter	Description	Value
DB_ADMIN_PROC	The procedure the database administration thread calls at a specified time in the day. This procedure could perform many tasks, including archiving and purging of dynamic data. All stored procedures in the database are recompiled and the statistics updated after this administration procedure has been called.	Default = PSP_db_admin.
DB_ADMIN_TIME	The number of minutes after midnight when the database administration tasks are to be performed. This is usually performed at a time of low system activity.	Default = 300.
DB_PCT_ANALYZE	This parameter applies to Oracle only. It is used to update statistics on all user-defined tables. The updates are done when the database administrations tasks are performed. (See also DB_ADMIN_TIME.) This parameter is used to optimize the database query performance. The Oracle SQL statement is "analyze table table_name estimate statistics sample DB_PCT_ANALYZE percent". See the discussion on the Analyze command in *Oracle SQL Reference Manual*.	Default = 20.

Generic EDD API Parameters

This section describes generic EDD API parameters.

Table 9–64 lists and describes the generic EDD API parameters and values.

Table 9–64 Generic EDD API Parameters and Values

Parameter	Description	Value
APPL_USERID	The primary SQL Server user ID the application process uses to connect to the APPL_SQL_SERVER SQL Server. The user ID defaults to the primary application database, and therefore, the application does not need database names because the definition of the user in the SQL Server specifies that user's default database. Maximum length 20 alphanumeric characters.	Default = $*CTRL_USER*.

BX25_EDD Configuration Parameters

External device driver applications link in libgedd, libclient, and libasc. Therefore, they require the libgedd, libclient, and libasc configuration parameter definitions in addition to the ones outlined below.

Table 9–65 lists and describes the BX25_EDD configuration parameter and values.

Table 9–65 BX25_EDD Configuration Parameter and Value

Parameter	Description	Value
MAX_WAIT_BLOCK_TIME	Number of seconds the X25 port monitor waits for data before it times out from this condition. In production, this value should be set for only a few seconds.	Default = 1.

PADEDD Configuration Parameters

This section describes the PADEDD configuration parameters.

Table 9–66 lists and describes the PADEDD configuration parameter and values.

Table 9–66 PADEDD Configuration Parameter and Value

Parameter	Description	Value
MAX_WAIT_BLOCK_TIME	The blocking time of the X.2X EDD to receive data from outside of X.2X EDD.	Default = 1.

UTILITY Configuration Parameters

This section describes the UTILITY configuration parameters.

Table 9–67 lists and describes the UTILITY configuration parameters and values.

Table 9–67 UTILITY Configuration Parameters and Values

Parameter	Description	Value
CONTROL_SQL_SERVER	The SQL Server where the Control database resides. If specified, the Control database can reside on a separate SQL Server from the application databases, if required. Maximum length is eight alphanumeric characters.	Default = $*DSQUERY*.
CONTROL_USERID	The SQL Server user ID the application process uses to connect to the Control database in the CONTROL_SQL_SERVER SQL Server. Maximum length is 20 alphanumeric characters.	Default = $*CTRL_USER*.

Login Information for the SARM Database

This section describes the login information for SARM database.

Table 9–68 lists and describes the login information for the SARM database parameters and values.

Table 9–68 Login Information for the SARM Database Parameters and Values

Parameter	Description	Value
SARM_USER	SARM user ID.	Default = $*SARM_USER*.
SARM_SQL_SVR	SARM server where the database resides.	Default = $*DSQUERY*.
SARM_POOL_SIZE	Number of database connections to the SARM database to maintain in the connection pool.	Default = 7.

Login Information for ADM Database

This section describes login information for ADM DB.

Table 9–69 lists and describes the login information for ADM database parameters and values.

Table 9–69 Login Information for the SARM Database Parameters and Values

Parameter	Description	Value
ADM_USER	ADM user ID.	Default = $*ADM_USER*.
ADM_SQL_SVR	N/A	Default = $*DSQUERY*.
ADM_POOL_SIZE	Number of database connections to the ADMIN database to maintain in the connection pool.	Default = 7.

A

asap_utils

This section explains the **asap_utils** functions. Because this is a UNIX and Linux program you can add or remove functions.

To invoke asap_utils:

1. Type the following at the $ASAP_base prompt:

   ```
   asap_utils [-P ctrl_password] [option]
   ```
   where:

   - [-P ctrl_password] is the password for the control database.
   - [option] is the asap_utils option you want to invoke. For example, for the Resend Completed ASDLs option, you would type 11.

   Note: If any **asap_utils** command fails, diagnostic information can be found in the *ASAP_home*/**DATA/logs/***yyyymmdd*/**UTILITY.diag** (where *yyymmmdd* is the year, month, and date of the logs) for the current date when the command was executed. For more information about diagnostic files, see *ASAP System Administrator's Guide*.

asap_utils Functions

This section describes each of the asap_utils functions.

SARM Utilities

1. Service Requests in DB

This lists the SRQs currently resident in the ASAP database. It details the order ID, status, priority, due date, parent order, batch group, etc.

2. In Proc Requests Summary

This lists the number of requests currently in progress in the ASAP database (this is determined to be orders in a Loading or In Progress state).

3. In Proc Requests Details

This lists details of any work orders currently in progress within the SARM.

4. Work Order Queue Summary

This details the number of orders in each of the SARM order queues and includes:

- Ready Queue – Orders currently in progress.
- Rollback Queue – Orders currently being rolled back.

- Auto Held Queue – Orders that are being held by the SARM and not released for some reason.

5. Work Order Queue Details

This provides the order details of each order in the SARM work order queues. Such queues are global to the SARM.

6. Work Order Lock States

This lists the orders in progress and their respective lock states. Generally, an in progress order will have a local lock. Only in the high availability configuration will orders be remotely locked.

7. ASDL/NE Queue Summary

This is one of the most commonly used utilities. It provides summary details about each NE in the system including:

- NE, technology and software load
- NEP managing the NE
- Current state of the NE Down, Connecting, Available, Maintenance, Disabled
- Whether the thoughput of the NE is throttled to the configured throughput value
- The configured throughput value
- Time estimate (sec.) for ASDL processing to that NE
- Number of ASDLs pending to that NE (in a prioritized queue)
- Number of ASDLs currently in progress, the number of connections open to that NE
- The number of ASDLs waiting to be retried to that NE.

8. ASDL/NE Queue Details

This provides details about each ASDL in the Pending, In Progress and Retry ASDL queues for each NE in the system.

9. HA Summary/Details

This option provides both a detailed and summary view of the ASAP high availability operation.

The ASAP configuration parameter, ASAP_HIGH_AVAIL, must be set for these RPCs to be installed; that is, the ASAP territory must be in HA mode.

10. Enable/Disable Entire NE

This utility facilitates the enabling/disabling of an entire NE from the SARM perspective. If a NE is disabled, then the SARM will hold all ASDLs queued to that NE in the ASDL pending queue for that NE.

In general, a NE is disabled for administrative purposes. It must be manually enabled for ASDLs to be transmitted to the NEP managing the NE. The current status of a NE may be viewed using the **30. List Device States** option.

11. Resend Completed ASDLs

This option allows you to resend recent changes/completed ASDL commands, to an NE. This allows ASAP to provide some basic retransmission logic to re-provision completed ASDLs to a particular NE. It is intended for use in situations in which a NE has been recovered to a known state and ASAP is requested to re-input all successful commands to the NE from that known state onwards.

The arguments for this option include the target NE, the start and end dates/times. This option returns the number of the work order created within SARM, and resends all ASDLs in their original provisioning sequence.

12. List NEs over Err Thresh

This option allows you to view any NE/ASDL combinations that have been automatically disabled after the consecutive error threshold for that NE/ASDL was exceeded. The error threshold for a NE and ASDL combination may be specified in the ASAP error threshold database table.

The state of a NE as displayed by the **30. List Device States** option may be available even though there may be several ASDLs that have been logically disabled to that NE by means of this facility.

13. Enable NE/ASDL over Thresh

This option allows you to logically enable NE/ASDL combinations that were logically disabled by the error threshold mechanism.

18. Set NE instance throughput

This option allows you to set the NE instance throughput value, which controls the minimum time in milliseconds an ASDL/transaction takes on an NE. A value of 0 disables NE instance throughput.

Admin Server Utilities

20. WO Stats

This option presents pseudo real time statistical information relating to SARM work order processing.

21. CSDL Stats

This option presents pseudo real time statistical information relating to SARM CSDL processing.

22. ASDL Stats

This option presents pseudo real time statistical information relating to SARM ASDL processing.

23. NE Stats

This option presents pseudo real time statistical information relating to SARM NE processing.

24. NE/ASDL Stats

This option presents pseudo real time statistical information relating to SARM NE/ASDL processing.

In particular, this provides information about the ASDL user exit codes from the NEP state tables and facilitates statistical collection of types of soft errors, hard errors, etc.

NEP Utilities

30. List Device States

This option details the state of each communication device in the selected NEP.

It includes the following:

- resource pool to which the device belongs

- device type (for example, T – Telnet device)
- device status (Enabled/Disabled)
- bind status of the device (Free/Bound to a NE)
- host NE to which the device last communicated

31. Enable/Disable NE Devices

This option allows you to enable/disable specific communication devices in the NEP used to communicate with external NEs.

There is also an ASAP automatic device re-enabling facility that automatically re-enables a disabled device after a specific period of time.

32. Enable/Disable Screen Diags

This option allows you to capture the return data stream from a terminal emulation session such as a Telnet session and append the output to a UNIX file.

You may then “tail -f” the file to see the ASAP interaction with the terminal session in real time.

This may only be used in terminal emulation sessions, not message based sessions.

If you provide only a file name for the log file, the log file will be stored in the default directory for line/screen diagnostics – $LOGDIR/ne_logs. If you provide a full path with file name, the log file will be stored at the location you specified.

$LOGDIR/ne_logs is the default directory for line/screen diagnostic files. However, if you specify a file path, such as './dms_log' or '../dms_log1', the dms_log will be created under $LOGDIR/ne_logs, and dms_log1 will be created in the $LOGDIR.

33. Enable/Disable NE Line Diags

This option allows you to capture all data passing on the communication “line” between the NEP and the NE while provisioning the NE. It is sometimes useful for diagnosing specific character sequences.

If you provide only a file name for the log file, the log file will be stored in the default directory for line/screen diagnostics – $LOGDIR/ne_logs. If you provide a full path with file name, the log file will be stored in the location you specified.

$LOGDIR/ne_logs is the default directory for line/screen diagnostic files. However, if you specify a file path, such as './dms_log' or '../dms_log1', the dms_log will be created under $LOGDIR/ne_logs, and dms_log1 will be created in the $LOGDIR.

34. Capture NE VS Screen to File

This option allows you to capture a snapshot of a terminal session screen to a file for review. This option is only available for state table based cartridges.

35. Enable/Disable EDD Diags

This option controls a series of external device driver diagnostic capabilities including the following:

- start_dump – Starts logging of all communication for this generic device
- stop_dump – Stops logging of all communication for this generic device
- dump_info – Prints information about the EDD devices to the specified file
- start_dump_all – Starts communication logging for all generic devices
- stop_dump_all – Stop communication logging for all generic devices

36. List NE Loopback Information

Using this option, all the network elements configured within an NEP server can be listed:

```
Specify NEP: [NEP_HT9]
Host NE   NE loop back
-------------------------------------------------------------
ROME        OFF [from NE config]
BEIJING     ON  [Global]
TORONTO     OFF [from NE config]
DYN_DALLAS  OFF [from NE config] [Dynamic Template]
```

As shown above, for dynamic NE configurations, only the dynamic template is listed. (When changing the loopback state for dynamic NE configurations, only the template is modified as the template represents a group of dynamic network elements.)

In the list above:

- OFF – Network element loopback is set to OFF
- ON – Network element loopback is set to ON
- Global – The Network element loopback setting is taken from the global loopback parameter (i.e. LOOPBACK_ON parameter in the ASAP.cfg file). The value OFF is used if the global loopback parameter LOOPBACK_ON is not specified in the ASAP.cfg file.
- [from NE config] – The network element loopback state is configured to be either ON or OFF independently of the value of the global loopback parameter LOOPBACK_ON.
- [Dynamic Template] – The network element is a template for dynamic network elements.

37. Set NE Loopback

Use this option to set the loopback state of network elements to ON, OFF or Global.

When you use this option, you are first notified with the following prompt:

```
Before changing the loopback state of an NE, it is suggested to disable the NE.
Would you like to continue ('Y') or not ('N')? [N] Y
```

The loopback state of a network element can be turned ON or OFF or set to Global at any time without having to stop and restart the ASAP Server. However, once a specific connection is established, its loopback state will be preserved. In order for the new loopback state to be used, the connection must be released and re-established. Any new connections will employ the updated loopback configuration for that network element.

Loopback parameters:

- OFF – Set the Network element loopback to OFF
- ON – Set Network element loopback to ON
- Global – The Network element loopback setting is taken from the global loopback parameter (i.e. LOOPBACK_ON parameter in the ASAP.cfg file). The value OFF is used if the global loopback parameter LOOPBACK_ON is not specified in the ASAP.cfg file.

When changing the loopback state for dynamic NE configurations, only the template is modified as the template represents a group of dynamic network elements.

The '%' can be used as a wildcard. Alone, it denotes all network elements (%). It can also denote a group of network elements (e.g. A%, AA%, AB%, ABC%).

Execution example:

```
Before changing the loopback state of an NE, it is suggested to disable the NE.
Would you like to continue ('Y') or not ('N')? [N] Y
Specify NEP: [NEP_HT9]
Specify NE name: [BEIJING] T%
Specify NE loopback operation: ON ('Y') or OFF ('N') or Global('G'): [G] Y
```

In state table programming, the network element state can be checked by referring to the value of the LOOPBACK_ON parameter as shown in the following state table fragment:

```
100     IF_THEN         '%LOOPBACK_ON == 0'
110         LOG         'NE is ALIVE (NO loopback)'
120         SEND        'ls -l /tmp'
130         SENDKEY     'ENT'                           3
140         ASDL_EXIT   'SUCCEED:Successful Non Loopback State Table'
150     ENDIF           ''
160    LOG         'NE is in LOOPBACK mode....'
```

In Java classes of the Java-enabled NEP server, the loopback state can be checked by inspecting the value of the LOOPBACK_ON parameter passed from the NEP server to the JNEP server (Java interpreter) during connection time as a connection parameter or as an order parameter at the beginning of ASDL execution. For example:

```
>> 193902.659:Connection handler srvtodev07-58877:LOW
:com.mslv.activation.jinterpreter.JInterpConnection:
```

Communication Parameters passed:

```
DEVICE = JTEL#1
LOOPBACK_ON = 1
...
>> 193902.659:Connection handler srvtodev07-58877:LOW
:com.mslv.activation.jinterpreter.ClassManager:
...
Received invocation request for telnet_demo.TelnetProcessor.prov
>> 193902.804:Connection handler srvtodev07-58877:LOW
:com.mslv.activation.jinterpreter.JInterpConnection:
```

Communication Parameters passed:

```
HOST_IPADDR = 127.0.0.1
HOST_NAME = localhost
HOST_PASSWORD = desert1
HOST_USERID = sunen808
LOGIN_PROMPT = login:
OPEN_TIMEOUT = 5
PASSWORD_PROMPT = Password:
PORT = 23
PROMPT = $
READ_TIMEOUT = 5
VSTYPE = vt100
VS_LENGTH = 24
VS_WIDTH = 80
WRITE_TIMEOUT = 5
```

Order Parameters passed:

```
ACT_FUNC_SEC = 0
ASDL_CMD = ASDL_TELNET
ASDL_UNID = 6
CSDL_CMD = CSDL_TELNET
CSDL_SEQ_NO = 5
DEVICE = JTEL#1
```

```
DIAL_NO =
HOSTCLLI = TEL_HOST
IS_ROLLBACK = NO
LOOPBACK_ON = 1
...
```

60. Clear Alarms

This option allows you to enter an alarm code and a number of hours as parameters. Alarms matching the alarm code that are older than the number of hours specified are cleared.

Use "%" as a parameter to clear all alarms.

Technical Utilities

The following options are not applicable for the C++ SRP servers.

100. View Server Thread Listing

This option displays details of all threads within the application server including:

- Thread ID (spid) and status
- Login name and hostname of incoming connection threads
- Thread name (cmd)

101. View Server Memory Statistics

This option provides statistics about the internal server memory management subsystem including the following:

- Memory blocks available
- Block sizes
- Number of such memory blocks currently in use
- High water mark of the memory block usage

These statistics only refer to use of the ASC memory management routines.

102. View Server Memory Usage

This option provides detailed memory usage statistics for the ASC memory management routines. The output of this RPC is appended to the application diagnostic file.

The configuration parameter, MEMORY_LOGGING, must be >= 1 for this RPC to be installed.

> **Note:** In production systems, this configuration parameter should not be set as it imposes a minor performance overhead.

103. Flush State Table Cache

This option will request the target application to flush its memory cache of state tables. This is usually performed to ensure that updated copies of state tables in the database are read into the cache when next executed. This allows dynamic loading of updated state tables.

104. View Server Thread Status

This option displays details about internal application threads. This includes the following:

- Start time of each thread
- Number of thread context switches since thread startup
- CPU time used by each thread
- If applicable, thread message queue details for each thread

105. View Server Thread Sleep Status

This options details the sleep and alarm requests currently being managed by the API including the following:

- Time to wake up
- Request type (Sleep, Alarm, or Poll)
- Thread queue name and ID to which alarm notifications are to be sent
- Socket file descriptor to which a message is to be written if the request is a poll request

106. Initiate Server DB Admin Proc

This option will initiate the database administration procedures to the primary database of the selected server. This routine performs the following steps:

- Invokes the DB Admin stored procedure, if defined
- Updates all database statistics on user-defined entities in this database
- Recompiles all SQL entities in the database to ensure they are optimized

This capability is only enabled if the DB_ADMIN_ON configuration parameter is set. In addition, the behavior of this option is controlled by several configuration parameters including DB_ADMIN_TIME, DB_ADMIN_PROC and DB_ADMIN_PROC_PARAM.

107. Change Server Diag Level

This option allows you to dynamically change the diagnostic level of an application server when it is running. For more information on diagnostic levels, see *ASAP System Administrator's Guide*.

This change only has effect as long as the server is running. Upon restart, it assumes the diagnostic level specified in the database configuration.

The System monitoring tool is not available to C++ SRPs.

108. Change Server Diag Line Flush

This option allows you to specify whether or not each diagnostic message written to the diagnostic file should be physically flushed to disk after each write.

In production systems, diagnostics must not be explicitly flushed after each write as it imposes a performance overhead.

109. Real-time System Monitoring

The ASAP Utility Script (asap_utils) is a menu that provides access from UNIX to a set of monitoring utilities for ASAP. You can access sysmon through the Real-time System Monitoring option (109) of the asap_utils menu. You can also monitor multiple servers at the same time by selecting the Real-time System Monitoring option (109) again. Once the data collection time period has passed, sysmon output files will be created in the ASAP systems diagnostic file directory.

Note: Option 101, View Server Msg Queue Statistics, available in previous versions of ASAP, as well as the configuration parameter DIAG_MSGQUEUES and the RPC diag_msgqueues, have been replaced by the functionality available from option 109, Real-time System Monitoring.

The system monitoring tool is not available to C++ SRPs.

Sysmon defaults to monitoring the SARM for 300 seconds. If you want to monitor a different ASAP component or change the length of the monitoring time, you can change the defaults.

```
----------------------
Tuning - Message Queue
----------------------
Description            Count   Total    Min   Max    Average  Mean   Deviation
------------           ------  ------   ----  -----  -------  -----  ----------
ASDL Provision Queue
message read wait time 1056   5995.5     0.9  62.2     5.7    15.1     10.2
messages sent (count)  1056
queue idle-time (ms)   1056   5971905.0  0.0  114374.2 5655.2 23775.0  19062.4
queue size (count)     1056   0.0        0.0  0.0      0.0    0.0 0.0
...

Group Manager Msg Q
message read wait time 2469   38896.8    2.5  290.5    15.8   61.5     48.0
messages sent (count)  2469
queue idle-time (ms)   2469   18707440.6 0.0  71880.3  7576.9 18294.2  119980.0
queue size (count)     2469   27184.0    0.0  27.1     7.2    1.5      4.5
...
```

For more information, refer to System monitoring output files, below.

110. Print System Monitor Report

This option allows you to print gathered information on a monitored ASAP component.

112. Load New Service Configuration into Cache

This option allows you to add new ASAP service definitions dynamically including all CSDL and ASDL-related configurations.

113. Load New NE Configuration into Cache

This option allows you to add ASAP network interface configurations dynamically including host NEs and NE resources.

114. Change JNEP Java Interpreter Diagnostic Level

To change the diagnostic level for JNEP, give the name of the related NEP server as follows:

Specify Server: [NEP_S11A]

Specify New Diag Level: (KERN, LOW, SANE, PROG) [SANE] PROG

115. Analyze Service Model & NEP Configuration Refresh

Performs a discrepancy analysis between the NE configuration in the cache and the configuration read from the database. Produces a discrepancy report which summarizes the discrepancies and includes the following fields: NEP, Host_CLLI,

Host_Update_Type, State, Estimate, Pending, In_Progress, Connect_Count, Retry_Count, NEP_Update_Type.

116. Service Model & Configuration Refresh

Refreshes the configuration from the database using the latest values without requiring a restart of ASAP. This command is similar to command 115, except in addition to producing the discrepancy report, it also applies the Service Model and NE configuration cache refresh.

117. List CSDL & ASDL version information and usage referencing

This command produces a report containing the current CSDL and ASDL version information and usage referencing count.

B

Stored Procedures (Deprecated)

This appendix provides information about stored procedures.

Configuring an SRP Using Stored Procedures

This section describes the basic SRP configuration steps using stored procedures. When configuring an SRP, you must first configure the basics before configuring a specific SRP type.

SRP configuration information is located in static tables in the ASAP Control server database and the SARM database.

To add an SRP to the system, perform the following steps:

- Adding the SRP to the ASAP Start-up Procedures
- Defining the SRP as an ASAP Component
- Adding the SRP to the SARM Database
- Registering the SRP
- Set Configuration Parameters by manually editing the **ASAP.cfg** file. See "Configuration Parameters"

Adding the SRP to the ASAP Start-up Procedures

The static table tbl_appl_proc contains ASAP application information. You must populate this table to identify the SRP as an ASAP application and add the SRP to the ASAP start-up procedure. The ASAP startup procedure uses tbl_appl_proc to determine what applications to start, and the startup sequence.

> **Note:** This procedure does not apply to the Java SRP and OCA SRP.

For more information on tbl_appl_proc, see the *ASAP Developer Reference*.

Use the following stored procedures to define, delete, and list ASAP client or server applications:

- CSP_new_appl – Defines a new ASAP client or server application.
- CSP_del_appl – Deletes an ASAP client or server application.
- CSP_list_appl – Lists ASAP application registration information for the specified application or for all applications in the Control database.

For more information on these stored procedures, see the *ASAP Developer Reference*.

Defining the SRP as an ASAP Component

The static table tbl_component contains a list of ASAP components for each ASAP territory and system. You must populate this table to add the SRP as an ASAP component.

For more information on tbl_component, see the *ASAP Developer Reference*.

Use the following stored procedures to define, delete, and list ASAP components in a territory:

- CSP_new_component – Defines an ASAP component within a territory.
- CSP_del_component – Deletes an ASAP component from a territory.
- CSP_list_component – Lists ASAP components.

For more information on these stored procedures, see the *ASAP Developer Reference*.

Adding the SRP to the SARM Database

The static table tbl_asap_srp defines an SRP in ASAP. You must populate this table to add the SRP to the SARM database. Any ASAP application process that communicates with the SARM as an SRP using the SRP API must be defined in this table.

For more information on tbl_asap_srp, see the *ASAP Developer Reference*.

Use the following stored procedures to define, delete, and list SRPs in the system:

- SSP_new_srp – Defines an SRP for the system in the SARM database.
- SSP_del_srp – Deletes an SRP from the SARM database.
- SSP_list_srp – Lists SRP definitions

For more information on these stored procedures, see the *ASAP Developer Reference*.

Registering the SRP

This step applies to C SRP APIs only.

For more information on configuring the OCA SRP, see "About JSRP, Web Service, and OCA SRP Components". For more information on configuring the C++ SRP, see "Using the C++ SRP API". For instructions on configuring the Java SRP, see "About JSRP, Web Service, and OCA SRP Components".

Sybase Open Client/Open Server comes bundled with ASAP and is installed automatically during ASAP installation.

The ASAP administrator or DBA must register the SRP by editing the Sybase interfaces file as follows:

- Add the ASAP component server names.
- Specify the service as “tcp” (on Oracle , specify “tcl tcp”).
- Assign port numbers for the servers.

Use the **dsedit** utility located in the $SYBASE/**SYBASE_OCS/bin** directory to perform these procedures. Before using **dsedit**, do the following:

- Log on as the **asap** user.
- Ensure that the $SYBASE variable points to the *ASAP_Home***/SYBASE** directory, where *ASAP_Home* is the directory in which ASAP is installed.
- Set the $DISPLAY variable by typing:

```
export DISPLAY=<your IP address>:0.0
```

- Ensure that the $LANG variable is set to English. If you are currently using a non-English language setting, you must temporarily set the $LANG variable to English by typing:

```
export LANG=C
```

Configuring NEPs Using Stored Procedures

The configuration of an NEP using stored procedures requires the following steps:

- Adding the NEP to ASAP Start-up Procedures
- Adding the NEP as an ASAP Component
- Adding the NEP to the SARM Database
- Adding the NEP to the Sybase Interfaces File
- Configuring Ports for the JInterpreter

These configuration procedures apply to all NEP core and optional components. These include EDD, SNMP, and generic communication protocols (such as Telnet, FTP, Serial, and socket).

Adding the NEP to ASAP Start-up Procedures

tbl_appl_proc is a static table that contains ASAP application information. ASAP uses this table to determine which applications to start, and the startup sequence. You must populate this table to add an NEP to the startup procedure.

Use the following stored procedures to define, list, and delete ASAP client or server applications:

- **CSP_new_appl** – This stored procedure defines a new ASAP client or server application.
- **CSP_list_appl** – This stored procedure lists ASAP application registration information for the specified appl_cd or all applications from the Control database.
- **CSP_del_appl** – This stored procedure deletes ASAP Application registration information from the Control database.

For more information on these stored procedures and “tbl_appl_proc”, refer to the *ASAP Developer Reference*.

Adding the NEP as an ASAP Component

tbl_component is a static table that contains a list of ASAP components for each ASAP territory and system. You must populate this table to add an NEP as an ASAP component.

Use the following stored procedures to define, list, and delete ASAP components in a territory are:

- **CSP_new_component** – This stored procedure defines an ASAP component in a territory.
- **CSP_list_component** – This stored procedure lists ASAP components.
- **CSP_del_component** – This stored procedure deletes an ASAP component.

For more information on these stored procedures and tbl_component, refer to the *ASAP Developer Reference*.

Adding the NEP to the SARM Database

tbl_nep is a static table that is referenced by the SARM and the NEP. This table maintains the relationship between the NEP and the secondary pool of devices that are used by the NEP to establish auxiliary connections to host NEs. Each NEP references this table upon start up to determine the secondary pool of devices available to all session managers within that NEP. It spawns a command processor thread for each device in the secondary pool of devices. You must populate this table to configure these connections.

Use the following stored procedures to define, list, and delete the auxiliary pool of devices or connections for an NEP:

- **SSP_new_nep** – This stored procedure defines a secondary (dialup) pool of devices or connections for a specified NEP in the SARM database.
- **SSP_del_nep** – This stored procedure deletes an NEP secondary pool definition from the SARM database.
- **SSP_list_nep** – This stored procedure lists NEP secondary pool definitions.

For more information on these stored procedures and "tbl_nep", refer to the *ASAP Developer Reference*.

Adding the NEP to the Sybase Interfaces File

You must add the NEP server to the Sybase interfaces file. For more information, refer to the *ASAP Installation Guide*.

Configuring Ports for the JInterpreter

You can enable or disable the JInterpreter for an NEP by defining or not defining the listener entry in tbl_listeners. If no $NEP_jlistener entry is configured for the server $NEP, the NEP is not Java-enabled. This configuration must be specified before startup; it is not runtime applicable.

During installation, ASAP defines a default NEP enabled with a JInterpreter.

Every NEP must maintain a dedicated connection to its JInterpreter (configured in tbl_listeners).

You can enable or disable the ports on the JInterpreter in one of two ways:

- Using the Service Activation Configuration Tool (SACT) – refer to Chapter 2, "Configuring ASAP Servers."
- Using the following stored procedures to define, list, and delete the auxiliary pool of devices or connections for an NEP:
  - **CSP_new_listener** – This stored procedure defines a listener entry for an NEP.
  - **CSP_del_listener** – This stored procedure deletes a listener entry for an NEP.
  - **CSP_get_listener** – This stored procedure lists listener entries for an NEP.

For more information on "tbl_listeners", refer to the *ASAP Developer Reference*.

Configuring Multiple JInterpreters

Each NEP has an entry in tbl_appl_proc. If the NEP is java-enabled, ensure that the following conditions are met:

- There should be a corresponding entry in $SYBASE/interfaces file. See "Adding the NEP to the Sybase Interfaces File".
- There should be a corresponding entry in tbl_listeners."Configuring Ports for the JInterpreter".
- There should be a unique JInterpreter script under *ASAP_Home*/**programs**.

  The JInterpreter script located in *ASAP_Home*/**programs** directory is a template. For each NEP, copy the JInterpreter script file in the format:

$NEP_ID_jinterpreter.

For example, if you have the following NEPs:

NEP_S123, NEP1S123, NEP2S123, NEP3S123

you must copy JInterpreter script for each NEP as follows:

cp JInterpreter NEP_S123_jinterpreter

cp JInterpreter NEP1S123_jinterpreter

cp JInterpreter NEP2S123_jinterpreter

cp JInterpreter NEP3S123_jinterpreter

Additional Considerations when Configuring Multiple JInterpreters

In the situation described above, where multiple JNEPs are deployed and the Jinterpreter_<NEP> files were created by making hard copies of the Jinterpreter, additional issues should be considered.

If you deploy using SAM/Studio the file updated is the Jinterpreter file. The updates are not propagated into the additional interpreter files (e.g. NEP_S123_jinterpreter, NEP1S123_jinterpreter, NEP2S123_jinterpreter, NEP3S123_jinterpreter from the example above.) Not all the NEPs would have knowledge of what has been deployed. This implies that changes to the Jinterpreter file would have to be manually propagated to these other files or they will have to be re-copied.

If you have some Class B JNEPs, they may connect to other databases, perform many lookups, perform many file management task, ftp transfers or other tasks. In this case, some but not all of your JNEPs may need different JVM startup parameters.

Consider whether it is the best choice to start all NEPs with high JVM allocation if only a small percentage of them needs it. It may make more sense for the majority to be configured as soft links to Jinterpreter and the small percentage that need the high JVM allocation to be standalone copies.

To simplify maintenance in a case where you have added JNEPs which are all the same, consider creating a soft link to the original Jinterpreter file.

Sample Scripts

Refer to *ASAP_Home***ASAP\samples\JeNEP\PLSQL** for a sample Oracle script that configures ASAP for JInterpreter connectivity.

Configuring Resource Pools Using Stored Procedures

Use the following stored procedures to define, delete, and list reserouce pools:

- **SSP_new_resource** – This stored procedure defines an NEP resource (“device”) to be used for NE access in the SARM database.
- **SSP_del_resource** – This stored procedure deletes an NEP resource record from the SARM database.
- **SSP_list_resource** – This stored procedure lists NEP resource records.

For more information on these stored procedures and tbl_resource_pool, refer to the *ASAP Developer Reference*.