Oracle® Sensor Edge Server

Guide

10g (10.1.3)

B28979-01

October 2006

ORACLE®

Oracle Sensor Edge Server Guide, 10g (10.1.3)

B28979-01

Primary Author: John Bassett

Contributing Author: Robin Clark, Joseph Garcia

Contributor: Samuelson Rehman, Anit Chakraborty, James Chase, Ron Caneel, Greg Grisco

Contents

3 Managing Oracle Sensor Edge Server

4 Using the Sensor Data Repository

5 Oracle Sensor Edge Mobile

6 Configuring Devices, Filter Instances, and Dispatchers

Preface

Oracle Sensor Edge Server Guide includes information on the Oracle Sensor Edge Server product, it's installation, and configuration. For information on developing Sensor Edge Server applications, see Sensor Edge Server Documentation on OTN.

Audience

This guide is for customers using Oracle Sensor Edge Server.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at

http://www.oracle.com/accessibility/

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, seven days a week. For TTY support, call 800.446.2398.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1 Installation

This chapter provides information on installing Oracle Sensor Edge Server. Topics include:

Hardware Requirements

The hardware requirements for Oracle Sensor Edge Server are the same as the hardware requirements for Oracle Application Server. Refer to the *Oracle Application Server 10g Installation Guide* for your platform for the specific hardware requirements.

Oracle Sensor Edge Mobile requires Pocket PC 2003 or later devices. Oracle Sensor Edge Mobile includes device drivers for Symbol 9000G and Intermec IP3 devices for RFID and barcode. Sensor Edge Mobile requires less than 150KB of program storage, and less than 1MB of memory.

Table 1–1 lists minimum memory and disk space requirements.

Table 1–1 Minimum Memory and Disk Space Requirements

	(OC4J and Oracle Sensor Edge Server)	Oracle Sensor Edge Server
Memory	512 MB	128 MB
Disk Space	550 MB	150 MB
TEMP Directory Space	256 MB	256 MB
Swap Space	1.5 GB	1.5 GB

Software Requirements

Oracle Sensor Edge Server requires the following:

- Unix, Linux, Windows 2000, Windows XP, or later
- Sensor Data Streams and Sensor Data Repository require an enterprise version of Oracle Database (version 9.2.0.6 or later). Oracle Database 10*g* Release 2 is recommended.
- To install and use Sensor Data Streams, Oracle Database must be running in ArchiveLog mode, with automatic archiving enabled. Refer to the appropriate version of the *Oracle Database Administrator's Guide* for instructions on changing to ArchiveLog mode, and for instructions on enabling automatic archiving. Note that these instructions vary depending on the Oracle Database version you are using, so be sure to consult the version of the *Oracle Database Administrator's Guide* that corresponds to the Oracle Database version you are using.
- Sun Java2 SDK, Standard Edition, version 1.4.2 or later
- Oracle Containers for J2EE (OC4J), version 10.1.3 or later

> **Note:** For backward compatibility with 10.1.2 OC4J instances:
>
> 10.1.2.x.x companion CD OUI (must install OC4J first)
>
> 10.1.2.x.x standalone version (unzip oc4j_extended.zip into a directory and install). See Installing OC4J below.

- J2EE Server and Process Management, 10.1.3 (Not with Oracle HTTP Server or WebCache)
- All other software requirements are the same as they are for Oracle Application Server

Additionally, Sensor Data Repository and Sensor Data Streams must have encoding set to UTF-8.

For Oracle Sensor Edge Mobile, the software requirements are as follows:

- Pocket PC 2003 or later
- Pocket IE 3.0.3 browser or later

Installing OC4J

Before you install Oracle Sensor Edge Server, you must install OC4J. In previous releases, OC4J was packaged as part of the Oracle Sensor Edge Server installation. In release 10.1.3, to make upgrading OC4J easier, it is not included as part of the Oracle Sensor Edge Server installation and must be installed before you install Oracle Sensor Edge Server.

You can install OC4J when you install Oracle Application Server. In the Oracle Universal Installer for Oracle Application Server, you must select **Advanced Installation Mode**, and then select the option for **J2EE Server and Process Management**. Refer to the *Oracle Application Server Installation Guide* for your platform for more information on installing Oracle Application Server.

Once you complete the OC4J installation, you are ready to install Oracle Sensor Edge Server.

If you are installing release 10.1.2 for backward compatibility, follow these installation steps:

1. After installing release 10.1.2.x.x companion CD OUI (or after unzipping oc4j_extended.zip), navigate to ORACLE_HOME/j2ee/home.
2. Run java -jar oc4j.jar -install.
3. When prompted for a password, enter an administrator password for the OC4J instance.This is the password that you will use to log in to the management UI, and it is also required for installation of Oracle Sensor Edge Server.

Upgrading or Installing Oracle Sensor Edge Server From a 10.1.2 OC4J Instance

If you already have a 10.1.2 instance of OC4J installed, you can install Oracle Sensor Edge Server by following the steps for "Installing Oracle Sensor Edge Server From a 10.1.3 OC4J Instance". During installation, Oracle Universal Installer will ask you to create the OC4J administrator password. This is the password used to install and administer OC4J.

If you already have a previous version of Oracle Sensor Edge Server installed, you will not be able to upgrade it. You must install the new version by following the steps for "Installing Oracle Sensor Edge Server From a 10.1.3 OC4J Instance". You will then need to manually recreate your Sensor Edge Server settings in the new version.

Installing Oracle Sensor Edge Server From a 10.1.3 OC4J Instance

To install Oracle Sensor Edge Server from a 10.1.3 instance of OC4J:

1. Insert the companion CD and start Oracle Universal Installer.
2. When the Welcome screen appears, click Next.
3. In the Specify File Locations screen, under Destination, enter a name for the installation and the path where you want to install Oracle Sensor Edge Server. Oracle Sensor Edge Server must use the same *oracle_home* as OC4J. Then click Next.
4. In the Select Installation Type screen, choose Sensor Edge Server and click Next.
5. In the J2EE Container (OC4J) Password screen, enter the password for the OC4J administrator, and click Next. This is the administrator password you defined when you installed OC4J.
6. In the Sensor Data Repository Installation screen, choose Yes if you want to connect the Sensor Edge Server to a Sensor Data Repository. If you choose Yes, you will be able to create a new Sensor Data Repository or connect to an existing one. If you choose No, skip to 10. If you choose No, and you later want to use Sensor Data Repository, you can manually configure it by following the steps in "Connecting to an Existing Sensor Data Repository in Release 10.1.2" or "Manually Configuring Sensor Data Repository and Sensor Data Streams in Release 10.1.3". After making your choice, click Next.
7. In the Sensor Data Repository Configuration screen, choose whether you want to connect to a new Sensor Data Repository or connect to an existing one, and click Next.
8. In the SDR Database Connection Information screen, enter the requested information for the Sensor Data Repository, and click Next.

9. In the Sensor Data Repository Password screen, specify the password you want to use for the Sensor Data Repository schema, and click Next. (This screen does not appear if you chose to connect to an existing Sensor Data Repository in 7.) The schema must be installed under the *edge* user in the database.
10. In the Sensor Data Streams Installation screen, choose Yes if you want to connect the Sensor Edge Server to Sensor Data Streams. If you choose Yes, you will be able to create a new Sensor Data Streams schema or connect to an existing one. If you choose No, skip to 14. If you choose No, and you later want to use Sensor Data Streams, you can manually configure it by following the steps in "Connecting to an Existing Sensor Data Streams in Release 10.1.2" or "Manually Configuring Sensor Data Repository and Sensor Data Streams in Release 10.1.3". After making your choice, click Next.
11. In the Connect to an Existing SDS screen, choose whether you want to create a new Sensor Data Streams schema or connect to an existing one, and click Next.
12. In the Sensor Data Streams Configuration screen, enter the database SYS password and TNS connection string, and click Next.
13. In the Sensor Data Streams Password screen, specify the password you want to use for Sensor Data Streams, and click Next. (This screen does not appear if you chose to connect to an existing Sensor Data Streams schema in 11.) The schema must be installed under the *edge* user in the database.
14. In the Summary screen, verify the installation options, and click Install.
15. In the Configuration Assistants screen, wait for the configuration assistants to finish, and click Next.
16. In the End of Installation screen, click Exit.

The Oracle Sensor Edge Server Welcome screen appears.

Examining the Installation Log Files

If there are errors during installation, you can examine the Oracle Sensor Edge Server log files located in `oracle_home/edge` (`createedgeuser_SDR.log`, `deploySDR.log`, `createedgeuser_SDS.log`), and the Oracle Universal Installer log files.

Uninstalling Oracle Sensor Edge Server

To uninstall Oracle Sensor Edge Server, you must use Oracle Universal Installer. In the Oracle Universal Installer Welcome screen, click Deinstall Products to uninstall.

Uninstalling Sensor Data Repository or Sensor Data Streams removes the temp files that were used to deploy the schemas, but the schemas themselves are not removed from the database.

Verifying Your Installation

To verify that your installation was successful, perform the following tasks:

- display and navigate through the screens of the Sensor Edge Server (SES) console
- test the database connection

Displaying the Sensor Edge Server Console

To display the SES console:

1. Determine the port number that the HTTP service is listening on (for example, 8888). If you do not know the port number, do the following:
   1. Open a command window, and change directories to *oracle_home*/opmn/bin.
   2. Type `opmnctl status -l`. The port number for the HTTP service is displayed.

   Note: To determine the port number for a 10.1.2 OC4J instance:
   1. Navigate to `ORACLE_HOME/j2ee/home/config`.
   2. Open the `http-web-site.xml` file.
   3. The `<web-site>` element includes an attribute called `port`. This attribute defines the port number.

   See *Oracle Application Server Containers for J2EE User's Guide* for more information.
2. Open a browser window and enter the URL for the Sensor Edge Server console in the form of http://localhost:*port_number*/edge/, where *port_number* is the port number for the http service.
3. On the Oracle Sensor Edge Server Login page, enter the username and password for the Sensor Edge Server administrator. The default username is oc4jadmin. For password, enter the password for the OC4J administrator (as specified when you installed OC4J).

Testing the Database Connection

You can test the database connection to verify that installation was successful. To connect to the Sensor Data Repository using SQL*Plus, do the following:

1. Configure tnsnames.ora (located in /network/admin/) to set up the TNS connection string for the database. For example:

```
BAR.US.ORACLE.COM =
   (DESCRIPTION =
      (ADDRESS_LIST =
         (ADDRESS = (PROTOCOL = TCP)(HOST = foo.us.oracle.com)(PORT = 1521))
      )
      (CONNECT_DATA =
         (SERVICE_NAME = BAR)
       )
    )
```

2. Connect to the schema, always using the username edge for sda_usr. For example, if the username is edge and the password is sda_pwd, enter:

```
sqlplus edge/sdr_pwd@BAR
```

3. To check if there are events coming in, verify that the number of rows in view edg_event_vw increases as events come in, as follows:

```
select count(*) from edg_event_vw
```

The count of events should increase as events are sent to the repository. In the case of a clean installation with no configuration, there should still be some events sent over, such as the edge server starting event.

Two other tables can be used for diagnostic purposes: edg_diag_tab (for checking device related error and status messages) and edg_log (for checking internal errors encountered in processing events).

To connect to the schema using SQL*Plus, do the following:

1. Configure tnsnames.ora to set up the TNS connection string for the database. For example:

```
BAR.US.ORACLE.COM =
   (DESCRIPTION =
      (ADDRESS_LIST =
         (ADDRESS = (PROTOCOL = TCP)(HOST = foo.us.oracle.com)(PORT = 1521)))
      )
      (CONNECT_DATA =
         (SERVICE_NAME = BAR)
       )
    )
```

2. Connect to the schema, always using the username edge for sds_usr. For example, if the username is edge and the password is sds_pwd, enter:

```
sqlplus edge/sds_pwd@BAR
```

   The sds_pwd is the password entered for Sensor Data Streams during installation.

3. To check if there are events coming in, verify that the number of rows in sda_ observations increases as events come in, as follows:

```
select count(*) from sda_observations
```

   The count of events should increase as events are sent to the repository.

Testing the Database Connection Using Enterprise Manager

For OC4J 10.1.3, you can test the JDBC connection to the Sensor Data Repository by examining the JNDI resource using OC4J Enterprise Manager as follows:

1. Login into OracleAS Enterprise Manager (http://localhost:*oc4j_web_port_number*/em/) by entering the OC4J administrator and password.
2. Click the *Members* tab and then click the OC4J instance (per default home).
3. Select *Administration*.
4. If needed, expand *Services*.
5. Click the *Go to Task* icon for *JDBC Resource*. The *JDBC Resources* page appears.
6. Click the *Test Connection* icon for *edge/SensorDataRepositoryDS*.
7. Click **Continue**.
8. Enter `select count (*) from edg_event_vw` in the *SQL Statement* field to verify that events are coming in and that the number of rows in the view `edg_event_vw` increase as these events come in.

   The event count should increase as events are sent to the Sensor Data Repository. In the case of a clean installation with no configuration, there should still be some events sent over, such as the Oracle Sensor Edge Server starting event.

To verify that events are coming in and that the number of rows in `sda_observations` increases as events come in, enter select count (*) from sda_observations in the *SQL Statement* field.

9. Click **Test**. The connection status displays under **Confirmation** (*Connection to "edge/SensorDataRepositoryDS" established successfully for all OC4J instances in the Group*.) An error displays if the connection has not been established.

Note: For OC4J 10.1.3, you can test the OC4J instance directly from OC4J Enterprise Manager. In addition, using the SES Console, you can set the log level of the Oracle Sensor Edge Server instance to *All* and then check the log at startup to ensure that it connects properly. The *View Log* page displays the errors from the Sensor Data Repository that indicate problems. For more information on setting the log level, see "Setting the General Information for the Oracle Sensor Edge Server Instance" and "Viewing Log Information".

For OC4J 10.1.2, you cannot test the connection using Enterprise Manager. However, if the log does not report errors from the Sensor Data Repository at startup, then the connection is successful. If the log level of the Oracle Sensor Edge Server instance is set to *All*, then the SES Console's *View Log* page displays any SQL errors.

Installing Oracle Sensor Edge Mobile

To install Oracle Sensor Edge Mobile:

1. Insert the companion CD and start Oracle Universal Installer.
2. When the Welcome screen appears, click Next.
3. In the Specify File Locations screen, under Destination, enter a name for the installation and the path where you want to install Oracle Sensor Edge Mobile. Oracle Sensor Edge Mobile must use the same *oracle_home* as OC4J. Then click Next.
4. In the Select Installation Type screen, choose Sensor Edge Mobile and click Next.
5. In the Summary screen, verify the installation options, and click Install.

The installer places the following .CAB files into *oracle_home*/edge:

- OracleEdgeMobile.PPC420_StrongARM-XScale.CAB: Installer for ARM or XScale devices. To run the installer on a Pocket PC device, follow the instructions under "Installing and Starting Oracle Sensor Edge Mobile on a Pocket PC Device".
- OracleEdgeMobile.PPC420_i686.CAB: Pocket PC emulator for use with Windows 2000/XP computers, to allow code testing when no Pocket PC device is available. To install the emulator, follow the instructions described in "Installing the Oracle Sensor Edge Mobile Emulator".

Installing and Starting Oracle Sensor Edge Mobile on a Pocket PC Device

To install Oracle Sensor Edge Mobile on a Pocket PC device:

1. Using ActiveSync, load OracleEdgeMobile.PPC420_StrongARM-XScale.CAB (located in *oracle_home*/edge) onto the Pocket PC device, and click on it to install it.

2. To start the Sensor Edge Mobile background service from the emulator screen, choose **Start**, then choose **Programs**, and then click the **Start EdgeMobile Service** shortcut. The background service should now be running.
3. To verify that the background service is running, open a File Explorer window, navigate to the \EdgeMobile\html_samples directory, and click admin.html. The Sensor Edge Mobile Administration page should appear as shown in Figure 1–1.

Figure 1–1 Sensor Edge Mobile Administration Page

4. Verify that the **System status** is "Running". If **System status** is not "Running", click **View log** to examine the log file for errors. The log file is shown in Figure 1–2.

Figure 1–2 Sensor Edge Mobile Log File

Changing the Default Device Configuration

By default, the Symbol 900G drivers are loaded on the Pocket PC device. The following additional drivers are provided:

- EdgeMobileConfig_INTERMEC.xml: Intermec drivers configuration file.

- EdgeMobileConfig_SYMBOL.xml: Symbol drivers configuration file.
- EdgeMobileConfig_SIMULATOR.xml: Simulator drivers configuration file.
- EdgeMobileConfig_KEYBOARD.xml: Keyboard dispatcher configuration file.

To load a different driver, rename the appropriate driver configuration file to EdgeMobileConfig.xml and restart the Sensor Edge Mobile service.

Reading RFID Tags

To read RFID tags, navigate to the \EdgeMobile\html_samples directory and open the file rfid_ops.html. If you are using the simulation driver, it generates tags just like a real device driver. The value of the simulated or real tag(s) should appear in the **RFID data** field.

Reading Barcode Data

To read Barcode data, navigate to the \EdgeMobile\html_samples directory and open the file barcode_ops.html. The value of the simulated or real bar codes should appear in the **Barcode data** field.

Shutting Down the Sensor Edge Mobile Service

To shut down the Sensor Edge Mobile service:

1. Navigate to the \EdgeMobile\html_samples directory and open the file admin.html.
2. Click **Shutdown System**.
3. Click **OK** in the Confirmation dialog box.
4. Verify that the **System status** is "Stopped".

Installing the Oracle Sensor Edge Mobile Emulator

If you are developing applications to use with Oracle Sensor Edge Mobile, you can run the Sensor Edge Mobile emulator on a Windows 2000 or Windows XP computer. Using the emulator allows you to test your code when no Pocket PC device is available.

To install the Oracle Sensor Edge Mobile emulator:

1. If you have not already done so, install Oracle Sensor Edge Mobile as described in "Installing Oracle Sensor Edge Mobile".
2. Download the SDK for Windows Mobile 2003-based Pocket PCs from this location:
   `http://www.microsoft.com/downloads/details.aspx?familyid=9996b314-0364-4623-9ede-0b5fbb133652&displaylang=en`
3. Install the SDK on a Windows 2000 or Windows XP computer. The Pocket PC emulator is part of the SDK.
4. To start the Pocket PC emulator from the Windows Start menu, choose Microsoft Pocket PC 2003 SDK, and then choose Pocket PC Emulator.
5. Once the emulator starts, choose Emulator, and then choose Folder Sharing.
6. Click Browse, and browse to an existing folder on your computer, and click OK.
7. In the Folder Sharing dialog box, click OK.

8. In the emulator window, choose **Start**, and then choose **Programs**.
9. Click the File Explorer icon.
10. In the File Explorer, select **My Device**.
11. Select the folder that you made a shared folder in Step 6.
12. Load the OracleEdgeMobile.PPC420_i686.CAB file into the shared folder and click on it in the emulator to install it. This file is located in *oracle_home*/edge.

Manually Configuring Sensor Data Repository and Sensor Data Streams in Release 10.1.3

If you want to use Sensor Data Repository or Sensor Data Streams in Release 10.1.3, but you did not configure them when you installed Sensor Edge Server, you can manually deploy them using the procedures described in this section. This section contains the following topics:

Manually Deploying Sensor Data Repository

If you have already deployed Sensor Data Streams, you can skip this section, because the Sensor Data Streams schema includes everything necessary to deploy Sensor Data Repository.

To manually deploy Sensor Data Repository, use the following procedure:

1. Create the edge user (schema) in your database to host the Sensor Data Repository schema. To do this, log in as system with SQL*Plus and run the `create_edg_sda_user.sql` script located in *oracle_home*/edge/stage/sql/10.1.3.
2. When you are prompted for an input value, enter the password you want to use for the edge schema.
3. Log in to SQL*Plus as the edge user with the password you just chose, and run the `edg_sda_with_edgeuser.sql` script located in *oracle_home*/edge/stage/sql/10.1.3.
4. Follow the procedure described in "Connecting Oracle Sensor Edge Server to Sensor Data Repository".

Connecting Oracle Sensor Edge Server to Sensor Data Repository

To manually connect Sensor Edge Server to Sensor Data Repository, use the following procedure:

1. Make sure that OC4J is running. To start OC4J, open a command window, and change directories to *oracle_home*/opmn/bin. Then type `opmnctl startall`.

Note: For OC4J 10.1.2, start OC4J by navigating to `ORACLE_HOME/j2ee/home` and then running `Java_HOME/bin/java -jar oc4j.jar`. JDK 1.4.2 should already be installed on the machine.

---

2. Open a browser window and enter the URL for Oracle Enterprise Manager in the form of http://localhost:*oc4j_web_port_number*/em/. Note that the OC4J web port may have an Apache listener, in which case you must use the port Apache is listening on.
3. Log in to Oracle Enterprise Manager using the OC4J password you entered in Oracle Universal Installer.
4. Navigate to the OC4J Home page, and click **Administration**.
5. Scroll down to JMX, click the icon next to **System MBean Browser**, and navigate to oc4j>J2EEServer>standalone>J2EEApplication>default.
6. Click **Operations**, then click **createNativeDataSource**. You use this screen to create the data source to connect to the database on which Sensor Data Repository schema is installed.
7. Enter the following values:
   - dataSourceName: edge/SensorDataRepositoryDS
   - user: edge
   - password: enter the password you chose when you ran the `create_edg_user.sql` script
   - jndiLocation: edge/SensorDataRepositoryDS
   - loginTimeout: 20
   - dataSourceClass: oracle.jdbc.pool.OracleDataSource
   - URL: enter the jdbc URL to your database. The URL should look something like `jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)(HOST=yourhost.us.oracle.com)(PORT=1521)))(CONNECT_DATA=(SERVICE_NAME=your_service_name)))`
8. Log in to the Sensor Edge Server administration console.
9. On the first page, for **Use Archive**, select Yes.
10. Restart OC4J.

Manually Deploying Sensor Data Streams

Before manually deploying Sensor Data Streams, Oracle Database must be running in ArchiveLog mode, with automatic archiving enabled. Refer to the appropriate version of the *Oracle Database Administrator's Guide* for instructions on changing to ArchiveLog mode, and for instructions on enabling automatic archiving. Note that these instructions vary depending on the Oracle Database version you are using, so be sure to consult the version of the *Oracle Database Administrator's Guide* that corresponds to the Oracle Database version you are using.

To manually deploy Sensor Data Streams if you've already created a Sensor Data Repository, use the following procedure:

1. Use SQL*Plus to connect to the database as sysdba, by running `sqlplus /nolog`.
2. Run `sqlplus connect sys/`*`your_pwd@your_db`*` as sysdba`.
3. Run the script `grant_edg_user.sql`, located in *oracle_home*/edge/stage/sql/10.1.3.

4. Follow the procedure described in "Connecting Oracle Sensor Edge Server to Sensor Data Streams".

To manually deploy Sensor Data Streams if you have not already created a Sensor Data Repository, use the following procedure:

1. Run the script `create_edg_user.sql`, located in *oracle_home* `/edge/stage/sql/10.1.3`. When prompted, enter the password you want to use for the edge schema.
2. Run `sqlplus connect edge/`*`your_edge_user_pwd@your_db`*.
3. Run the script `edg_create_streams.sql`, located in *oracle_home* `/edge/stage/sql/10.1.3`.

   When running `edg_create_streams.sql`, you may encounter the following error:

```
drop role edg_user
          *
ERROR at line 1:
ORA-01919: role 'EDG_USER' does not exist
```

   You can ignore this error, as it notes that this is the first time an edge schema was installed on this database instance.

4. Follow the procedure described in "Connecting Oracle Sensor Edge Server to Sensor Data Streams".

Note: Installing Oracle Data Streams on an Oracle9*i* database instance results in the following error:

```
BEGIN DBMS_STREAMS_AUTH.GRANT_ADMIN_PRIVILEGE(grantee => 'edge');
END;

      *
ERROR at line 1:
ORA-06550: line 1, column 7:
PLS-00201: identifier 'DBMS_STREAMS_AUTH.GRANT_ADMIN_PRIVILEGE'
must be
declared
ORA-06550: line 1, column 7:
PL/SQL: Statement ignored
```

Because this is a Oracle 10*g* procedure that does not exist in Oracle9*i*, you can ignore this error.

Connecting Oracle Sensor Edge Server to Sensor Data Streams

To manually connect Sensor Edge Server to Sensor Data Streams, use the following procedure:

1. Make sure that OC4J is running. To start OC4J, open a command window, and change directories to *oracle_home* `/opmn/bin`. Then type `opmnctl startall`.

Note: For OC4J 10.1.2, start OC4J by navigating to `ORACLE_HOME/j2ee/home` and then running `Java_HOME/bin/java -jar oc4j.jar`. JDK 1.4.2 should already be installed on the machine.

---

2. Open a browser window and enter the URL for Oracle Enterprise Manager in the form of http://localhost:*oc4j_web_port_number*/em/. Note that the OC4J web port may have an Apache listener, in which case you must use the port Apache is listening on.
3. Log in to Oracle Enterprise Manager using the OC4J password you entered in Oracle Universal Installer.
4. Navigate to the OC4J Home page, and click **Administration**.
5. Scroll down to JMX, click the icon next to **System MBean Browser**, and navigate to oc4j>J2EEServer>standalone>J2EEApplication>default.
6. Click **Operations**, then click **createNativeDataSource**. You use this screen to create the data source to connect to the database on which Sensor Data Streams schema is installed.
7. Enter the following values:
   - dataSourceName: edge/StreamsDS
   - user: edge
   - password: enter the password you chose when you ran the `create_edg_user.sql` script
   - jndiLocation: edge/StreamsDS
   - loginTimeout: 20
   - dataSourceClass: oracle.jdbc.pool.OracleDataSource
   - URL: enter the jdbc URL to your database. The URL should look something like `jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)(HOST=yourhost.us.oracle.com)(PORT=1521)))(CONNECT_DATA=(SERVICE_NAME=your_service_name)))`
8. To configure Sensor Edge Server to use the streams dispatcher, log in to the Sensor Edge Server administration console.
9. Click **Change dispatcher**, and then select **Streams DispatcherV2**.
10. Restart OC4J. The streams dispatcher will attempt to connect to the Sensor Data Streams schema. For more information, see "Setting the Dispatcher for the Oracle Sensor Edge Server Instance" and "Starting and Stopping the Oracle Sensor Edge Server Instance".

Connecting to an Existing Sensor Data Repository in Release 10.1.2

If you want to use Sensor Data Repository, but you did not configure it when you installed Sensor Edge Server, you can manually connect to it using the following procedure:

1. Using a text editor, open the data-sources.xml file. This file is located in *oracle_home*/j2ee/home/config/.
2. Add the following xml entry:

```
<data-source
       class="com.evermind.sql.DriverManagerDataSource"
       name=""edge/SensorDataRepositoryDS"
       location=""edge/SensorDataRepositoryDS"
       xa-location=""edge/SensorDataRepositoryDSXA"
```

```
        ejb-location=""edge/SensorDataRepositoryDS"
        connection-driver="oracle.jdbc.driver.OracleDriver"
        username="edge"
        password="<password for sdr>"
        url="<jdbc url for database holding SDR>"
        inactivity-timeout="100"
    />
```

3. Save and close the data-sources.xml file.

Connecting to an Existing Sensor Data Streams in Release 10.1.2

To use Sensor Data Streams, Oracle Database must be running in ArchiveLog mode, with automatic archiving enabled. Refer to the appropriate version of the *Oracle Database Administrator's Guide* for instructions on changing to ArchiveLog mode, and for instructions on enabling automatic archiving.

If you want to use Sensor Data Streams, but you did not configure it when you installed Sensor Edge Server, you can manually connect to it using the following procedure:

1. Using a text editor, open the data-sources.xml file. This file is located in *oracle_home* / j2ee / home / config / .
2. Add the following xml entry:

```
<data-source
       class="com.evermind.sql.DriverManagerDataSource"
       name=""edge/StreamsDS"
       location=""edge/StreamsDS"
       xa-location=""edge/StreamsDSXA"
       ejb-location=""edge/StreamsDS"
       connection-driver="oracle.jdbc.driver.OracleDriver"
       username="edge"
       password="<password for sds>"
       url="<jdbc url for database holding SDS>"
       inactivity-timeout="100"
   />
```

3. Save and close the data-sources.xml file.

2

Introducing Oracle Sensor Edge Server

Oracle Sensor Edge Server is a middle tier component that integrates sensors and other types of command or response indication equipment with applications. Oracle Sensor Edge Server enables enterprises to incorporate information received from sensor devices into their I.T. infrastructure and business applications.

Oracle Sensor Edge Server receives event data from sensor devices and then normalizes this data by putting it in a common data format and filtering out excess information. The event data, which is now a normalized event message, is then sent to database applications.

This chapter provides an overview of Oracle Sensor Edge Server. It contains the following sections:

What's New in Release 10.1.3

This section summarizes the new and enhanced features for Oracle Sensor Edge Server for Release 10.1.3.

Release 10.1.3 includes the following new features and enhancements:

- Sensor Edge Server (SES) Console
- Oracle Sensor Edge Mobile
- Enhanced management through Oracle Application Server Control
- Improved performance
- Enhanced Sensor Data Repository
- Sensor Data Streams
- Electronic Product Code (EPC) compliance integration
- Transport layer
- Enhanced Security
- Deprecation of the Device Controller

Oracle Sensor Edge Server Console

You perform Oracle Sensor Edge Server administration, monitoring, and configuration using a new Sensor Edge Server Console, as shown in Figure 2–1. The tools provided by the console help to decrease deployment and maintenance costs. Using the console, you can:

- configure general server settings
- monitor server event statistics
- view and manage extensions such as dispatchers, filters, and drivers
- manage groups (logical collections of devices)
- configure new devices
- change server instance parameters

Figure 2–1 Oracle Sensor Edge Server Console

Some of the common tasks you can perform include:

- renaming a server instance
- changing the site name of the server
- changing instance parameters such as internal queue type, log level, and shutdown timeout
- configuring or changing the dispatcher
- performing group management such as creating a new group, renaming or deleting a group, adding a filter to a group, configuring a new group filter, and adding a device to a group
- configuring a new device
- adding or configuring a filter for a device

Oracle Sensor Edge Mobile

Designed for mobile workers in warehouses and on factory floors, Oracle Sensor Edge Mobile runs on handheld RFID readers that run on Pocket PC 2003 and later

platforms. Sensor Edge Mobile ships with Intermec and Symbol device support, and can be controlled through an ActiveX component. Sensor Edge Mobile interfaces with different types of sensors and devices, and feeds observation events to applications.

Enhanced Management Through Oracle Application Server Control

In this release, Java Management Extensions (JMX) interface has been integrated into the Application Server management console to increase accessibility and ease of management. The administrative tasks that are integrated into Application Server include startup, shutdown, process management, and recovery. These enhancements will help to decrease deployment and maintenance costs.

Improved Performance

Scalability and latency are crucial components for the real time performance required by many sensor-enabled applications. For this release, Oracle Sensor Edge Server provides improvements in scalability and latency through optimizations in the Database layer and the application server layer.

Enhanced Sensor Data Repository

In this release, the Sensor Data Repository (SDR) is specifically designed to store EPC and sensor data. The database schema stores status and diagnostic information from Sensor Edge Server, and serves as the single repository for sensor data.

The SDR is a database schema where the sensor events (RFID, temperature, location, etc.) obtained from Oracle Sensor Edge Server are archived. The events are normalized on their way into the SDR to facilitate business intelligence, such as reporting. The SDR can be enabled and configured so that Oracle Sensor Edge server can use it for storing event data and querying the event archive. Typically, multiple Sensor Edge Servers can share one SDR.

The SDR provides a set of database tables, views, and PL/SQL packages to facilitate the storage and retrieval of sensor events data.

Sensor Data Streams

Oracle Sensor Edge Server relies on plug-ins, called dispatchers, to communicate with applications and event distribution systems. The StreamsDispatcher is a plug-in that sends and receives events using Oracle Streams technology.

EPC Compliance Integration

This release includes EPC compliance features that were previously provided separately. This functionality allows customers to easily comply with the minimum EPC requirements and also provides a fully scalable infrastructure that can be applied to an entire enterprise.

Transport Layer

The transport layer simplifies development and maintenance for Oracle Sensor Edge Server applications. The transport layer provides an extensible development platform that simplifies device implementation and upgrading. The transport layer also resolves platform dependencies so that your Device Drivers can work with multiple operating systems.

Enhanced Security

By using Java Naming and Directory Interface (JNDI), Oracle Sensor Edge Server provides enhanced security. JNDI allows applications to work in conjunction with directory-specific security systems.

Deprecation of the Device Controller

The Device Controller is no longer necessary and has been removed from Oracle Sensor Edge Server for this release. The following devices that previously used the Device Controller can now operate without it:

- Patlite
- Intermec

Oracle Sensor Edge Server Overview

Oracle Sensor Edge Server manages and monitors the performance of the sensors that are integrated with your Information Infrastructure. Sensor Edge Server collects sensor information, filters it, and performs local sensor event processing. Oracle Sensor Edge Server then securely and reliably dispatches event data back to the central applications and databases.

For example, RFID readers use small transponders with embedded Electronic Serial Numbers (ESN) or memory, which transmit identifiers across one or more frequencies. A transponder can carry other information in its memory, and can be read or written to at a distance, without the need for line-of-sight contact. Oracle Sensor Edge Server interfaces with all the readers and sends normalized data back to the application server.

Oracle Sensor Edge Server provides the following features:

- Sensor Data Collection
- Sensor Data Filtering
- Sensor Data Dispatching
- Sensor Data Archive and Rules
- Sensor Server and Device Management
- Sensor Edge Mobile
- SES Console

Sensor Data Collection

Oracle Sensor Edge Server provides an extensible driver architecture that integrates with any sensor source such as Radio Frequency Identification (RFID) readers, printers, temperature, motion, pressure, or location, and any response devices such as light stacks, message boards, and sound systems. You can also implement custom components to plug in to the Sensor Edge Server architecture. You can implement the standard driver interface to Sensor Edge Server, or expose custom functionality of the sensor hardware through the Sensor Edge Server configuration interface.

Sensor Data Filtering

Data streaming in from sensors connected to Sensor Edge Server arrives in a wide range of formats, and includes unnecessary or redundant information. Before the

sensor data is passed on to enterprise applications, Sensor Edge Server performs data cleansing by filtering the data from individual sensors and groups of sensors. This process normalizes the sensor data into a consistent format. Groups of sensors can be treated as single logical entities for filtering purposes.

Sensor Data Dispatching

Dispatchers are plug-ins that enable two-way communication with applications. The main output of Oracle Sensor Edge Server is filtered data events. These data events are provided already minimized and normalized. They can be delivered in one of the out-of-the-box supported methods:

- Streams / AQ: Dispatching through Oracle Streams provides the most robust and flexible method of forwarding data. This is the only dispatching method that fully supports rule-based processes and agents-based technologies.
- Java Messaging Services (JMS): JMS is a standard messaging API that J2EE components use to communicate. The Oracle Sensor Edge Server provides a JMS Dispatcher to enable users to relay events to JMS topics.
- Web Services
- HTTP Post
- Other forwarding methods can be supported by writing a Custom Event Dispatcher.
- EPC PML (Physical Markup Language)
- Application Level Events (ALE)

Sensor Data Archive and Rules

Once the data has been filtered and dispatched, Oracle Sensor Edge Server uses the Oracle 10g database to provide sensor data archiving. You can use sensor data schemas tailored for storing sensor data captured through Sensor Edge Server. This archive provides uniform storage for sensor data for the entire enterprise, and provides a repository for data analysis.

Sensor Server and Device Management

To reduce maintenance costs for sensor infrastructure, Oracle Sensor Edge Server provides device management and monitoring from a single centralized location.

Sensor Edge Mobile

Oracle Sensor Edge Mobile is a Pocket PC application that runs on handheld devices such as RFID and barcode readers. Sensor Edge Mobile interfaces with different types of sensors and devices, and feeds observation events to applications. For more information see "Oracle Sensor Edge Mobile Architecture".

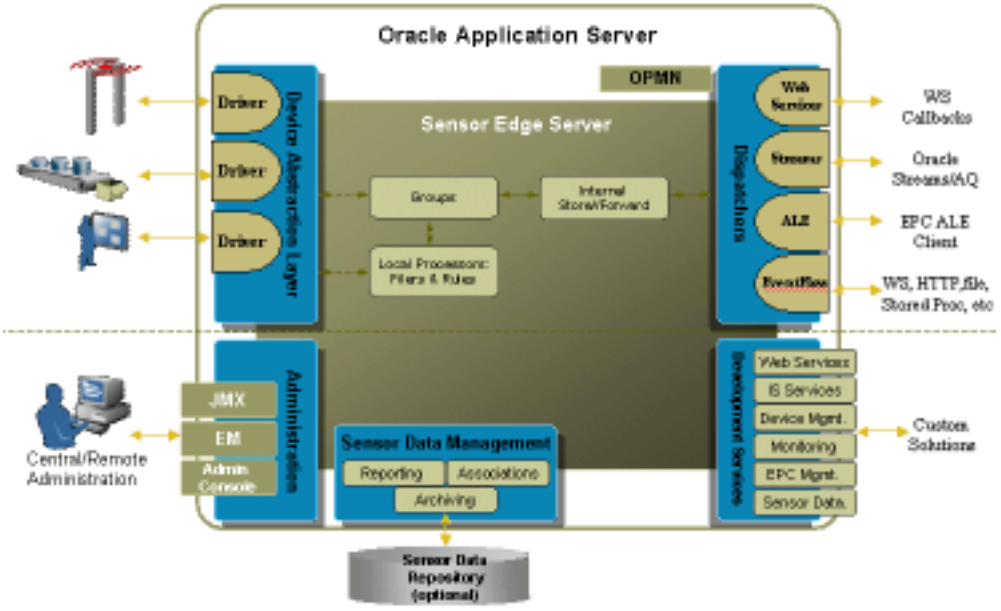
SES Console

You perform Oracle Sensor Edge Server administration, configuration, and monitoring using the SES console. For more information, see "Oracle Sensor Edge Server Console".

Oracle Sensor Edge Server Architecture

This section describes the highlights of Oracle Sensor Edge Server architecture. Figure 2–2 provides an overview of Oracle Sensor Edge Server architecture.

Figure 2–2 Oracle Sensor Edge Server Architecture



Device Drivers

Device Drivers communicate with sensors, and normalize incoming data into standard format. Device Drivers can be customized or built to suit your needs.

Device Groups

Device Groups are used by administrators to group devices logically, and manage them more efficiently. An Oracle Sensor Edge Server can have one or many Device Groups instantiated. Each Device Group is responsible for the Device Drivers that it manages.

Local Processing

Local processing (filters and rules) removes unwanted or low-level events. Filters and rules can be used to sort incoming data. They can be applied to individual Devices, or to Device Groups.

Event Processor

Event Processor is the central processing engine for event delivery. It loads the rest of the components, which include the Driver Manager and (on start up) the Event Dispatcher. Internally, it reads the configuration to find out which Event Dispatcher to load. The process is started on startup.

Driver Manager

Driver Manager loads and manages the life cycle of the Device Groups and Device Drivers. There is only one instance of a Driver Manager for each Oracle Sensor Edge Server.

Driver Manager provides an accessible context to Device Drivers so they can retrieve configuration information. It then loads the Device Groups, and binds them to the Device Drivers. The Driver Manager does not hold any thread internally, but an instance of it is held by the Event Processor instance as long as the server is up and running.

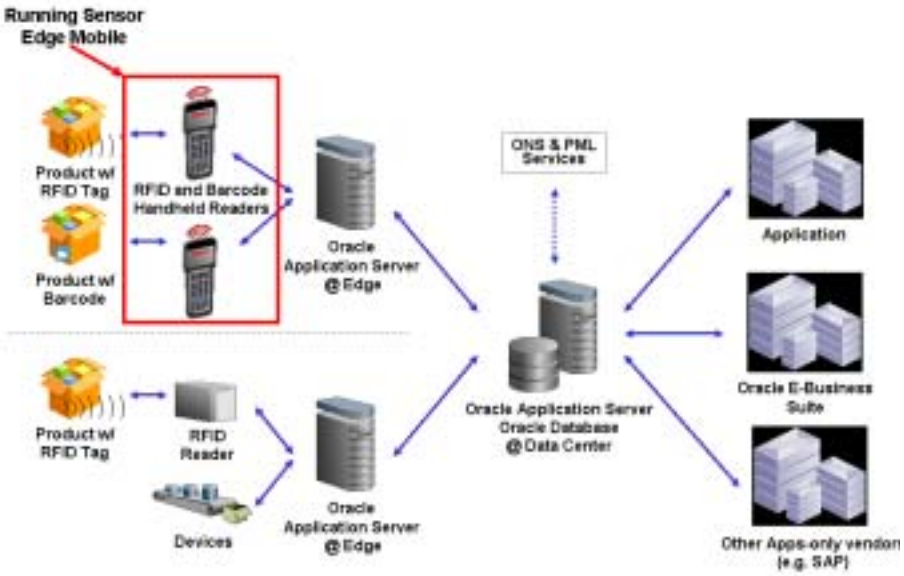
There is one Event Processor and one Driver Manager in each Oracle Sensor Edge Server. Each Driver Manager loads a number of Device Groups. In turn, each Device Group can have any number of Device Drivers. However, only one Device Driver can belong to one Device Group.

Oracle Sensor Edge Mobile Architecture

Oracle Sensor Edge Mobile is a platform that manages mobile sensor data collection on the device and communicates with the application running on the device or remotely through the device's browser. The communication is bi-directional, with events passing from devices, through the platform, to the application, and instructions passing from the application, through the platform, to the hardware Device Driver.

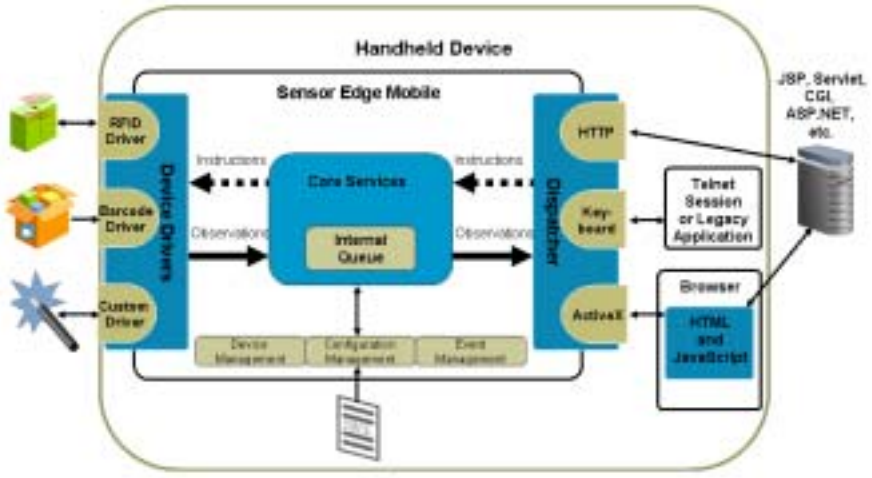
Figure 2–3 provides an architectural overview of Oracle Sensor Edge Mobile. Sensor Edge Mobile runs entirely on a handheld device, and communicates with other applications or services external to it, or operates entirely offline, collecting data that can be synchronized with an application at a later time.

Figure 2–3 Overview of Sensor Edge Mobile Architecture



Taking a closer look at the Oracle Sensor Edge Mobile platform reveals that observations collected by Device Drivers are put in an internal queue until the Dispatcher can process them. Figure 2–4 illustrates Sensor Edge Mobile components.

Figure 2–4 Sensor Edge Mobile Components



The Dispatcher communicates with the application using a local ActiveX control, by sending characters to the keyboard buffer, or through almost any other communication method. Only one Dispatcher can be active at a time, but that Dispatcher may be communicating with multiple client devices and corresponding drivers.

The Sensor Edge Mobile core components include the Driver Manager, Event Manager, and the Configuration Manager. Once these components are started, the main service calls on the Configuration Manager to read the configuration file. The service then starts the Dispatcher that has been configured, passing in any configuration parameters that have been specified.

The Driver Manager is responsible for loading and managing the lifecycle of the drivers. The Driver Manager calls on the Configuration Manager to determine which drivers need to be loaded, and what parameters to make available to them on instantiation, and then loads and initializes them.

There is only one Event Manager and one Driver Manager in Oracle Sensor Edge Mobile. The Driver Manager may load a number of drivers.

Device Driver Support

Oracle Sensor Edge Mobile supports Symbol 9000-G devices for the following operations:

- RFID Read
- RFID Write
- RFID Kill
- Barcode Read

Oracle Sensor Edge Mobile supports Intermec IP3 with Color 700 PocketPC attached for the following operations:

- RFID Read
- Barcode Read

Administering Oracle Sensor Edge Mobile

Oracle Sensor Edge Mobile can be administered through the Oracle Sensor Edge Server Web interface. You can perform the following administrative tasks:

- monitor overall service status
- monitor administrative logs
- review driver statistics
- shutdown the service

Sample Code and Demo Applications

Oracle Sensor Edge Mobile includes sample HTML and JavaScript code that shows examples of the following:

- reading and writing RFID tags
- reading Barcodes
- system status page
- service administration page

The sample code illustrates how to use the APIs and also demonstrates how Oracle Sensor Edge Mobile can be run immediately after installation and configuration.

Deployment Considerations

The Oracle Sensor Services architecture addresses the challenges faced in industrial supply chain environments as well as in specialized multi-sensor environments for commercial and government applications. To meet these diverse needs, Oracle offers dual Edge deployment configurations, each with it's own unique footprint and the built-in flexibility to support both centralized and federated site deployment scenarios.

The following considerations can help you determine which configuration to choose for your deployment or pilot.

Review Network Characteristics

In planning a deployment, first identify network bandwidth within the facility as well as intranet connectivity to other distribution centers and the Data Center. Typical considerations include the following:

- In-warehouse connectivity: typically either provided by direct Ethernet or Wireless 802.xx. Connectivity outages are to be expected.
- Warehouse to Warehouse or Data Center connectivity: In most cases, this connection is no more than a single ISDN line.

Identify Data Center Environment

Typical data center considerations include the following:

- Environment: In-warehouse hardware is typically ruggedized (bar code readers, system controllers, etc.) and, in some cases, environmental control is limited by operational requirements (for example, open dock doors).
- Support staff: Is IT support staff available on site or remotely, or not available?

Review Reader and Sensor Locations

For example, in locating RFID devices, there is typically a trade-off between customer constraints (pick and pack locations, conveyor lines, etc.) and where the RF survey indicates that the RFID hardware would work most efficiently and with minimal interference.

You should also identify the connectivity requirements for the reader and sensor devices (for example, Serial, Wireless, direct Ethernet, etc.).

Choose Edge Server Locations

Once the previous items are identified, design and document the location of Oracle Sensor Edge Servers for each data center. The number and location of Edge servers is typically determined by the following:

- The number of devices in each location and their interface requirements.
- Access to power and network connectivity for the Edge server.
- Environmental considerations (for example, exposure to weather and industrial equipment).

Oracle Sensor Edge Server and Sensor Data Repository Considerations

If the warehouse or distribution center has a server room and support staff, consider making the Sensor Data Repository (SDR) infrastructure available locally. Having the SDR infrastructure available locally can help minimize response times if complex business decisions need to be made locally and acted upon in real time. Oracle Sensor Edge Server offers a small footprint in warehouses or distribution centers while providing full data capture and filtering functionality. For more information on the SDR, see Chapter 4, "Using the Sensor Data Repository".

Having the SDR infrastructure available locally may also be necessary if network connectivity from the warehouse back to the central Data Center is minimal and latency would prevent actionable alerts or decisions on the scanned data.

Replication is typically implemented at the database level to synchronize transactions with other Data Centers or the central Data Center.

Most customers host their database and application servers at a central data center. This scenario uses minimal or no support staff at the warehouses or distribution centers. Only the Sensor Edge Server resides at the warehouse with connectivity back to the central data center for database and application server features.

Network bandwidth is critical in a centralized configuration and must be weighed against the processing to be performed on the filtered data. Data filtering, exceptions, and basic triggers can be performed locally. Complex rule-sets, product correlation, and cross-type queries can all be done against the remote instance and decisions returned to the local warehouse system.

If Control System integration is required, local processing filters or the SDR should be hosted locally to provide the features and performance required for automated warehouse operations.

3

Managing Oracle Sensor Edge Server

This chapter, through the following sections, describes how to manage and monitor an Oracle Sensor Edge Server instance using the Sensor Edge Server Console (the SES Console).

Overview of Oracle Sensor Edge Services Management

The Oracle Sensor Edge Server enables enterprises to incorporate information from sensors into their I.T. infrastructure and business applications by receiving event data from sensor devices or applications and then normalizing this data by putting it in a common data format and then stripping it of extraneous information using filters. The event data, which is now a normalized event message, is then sent to edge clients using a dispatcher. Depending on the configuration of the Oracle Sensor Edge Server's dispatcher, an Oracle Sensor Edge Server client receives event messages through Web Services, HTTP, EPC PML, ALE Web Services, or database streams. The payload of the message is always an event.

The SES Console enables you to manage and monitor the Oracle Sensor Edge Server instance. Table 3–1 describes the tabs of the SES Console and the tasks that they enable.

Table 3–1 Tasks Enabled by the SES Console

Tab	Tasks
Main	Tasks include: ■ Setting the Dispatcher for the Oracle Sensor Edge Server Instance ■ Setting the Devices and Filters Used by the Oracle Sensor Edge Server ■ Starting and Stopping an Individual Device ■ Restarting an Oracle Sensor Edge Server Instance Using the OracleAS Enterprise Manager ■ Accessing Other Oracle Sensor Edge Server Instances
Monitor Events	Viewing Unprocessed Event Data
View Log	Viewing Log Information
Event Reports	Tasks include: ■ Searching for Events by Device Name ■ Searching for Events by Tag ID ■ Creating Advanced Searches

Managing the Oracle Sensor Edge Server Instance

After you log into the SES Console using the OC4J administrator name and password, The console defaults to the *Configuration* tab, which displays the *Main* page. This page provides an overall view of the current Oracle Sensor Edge Server instance usage, its basic configuration and its current dispatcher. You can edit the basic configuration, such as the *Server* and *Site Name* parameters from this page, as well as select another dispatcher method. For more information on the basic configuration for an Oracle Edge Sensor Server Instance, see "Setting the General Information for the Oracle Sensor Edge Server Instance".

Note: You must first start OC4J. See "Starting and Stopping the Oracle Sensor Edge Server Instance".

Figure 3–1 The Main Page (Partial View)

See "Setting the General Information for the Oracle Sensor Edge Server Instance" for more information on the basic configuration of the Oracle Sensor Edge Server instance. For more information on dispatchers, see "Configuring Devices, Filter Instances, and Dispatchers".

The navigation tree (Figure 3–2), which displays on each page of the SES Console, enables you to both configure the current Oracle Sensor Edge Server instance and the extensions (filters, drivers, and dispatchers) in the repository. The tree organizes the configuration pages of the current Oracle Sensor Edge Server as a hierarchy, with the name of the current Oracle Sensor Edge Server as the top node. Clicking the plus (+) sign or the minus sign (-) next to a node enables you to display or hide items. You can modify the underlined items of the tree (such as *Groups* in Figure 3–2). Clicking an underlined item enables you to access a properties page, which enables you to modify and manage the selected item. Other items (that is, those that are not underlined), are titles and cannot be modified.

Figure 3–2 The Navigation Tree

The pages accessed through the tree enable you to perform the following tasks:

- Monitoring the Performance of the Oracle Sensor Edge Server Instance
- Setting the General Information for the Oracle Sensor Edge Server Instance
- Viewing Dispatchers, Drivers, and Filters
- Setting the Devices and Filters Used by the Oracle Sensor Edge Server
- Starting and Stopping an Individual Device

Accessing Other Oracle Sensor Edge Server Instances

The SES Console enables you to access other Oracle Sensor Edge Server instances that are connected to the same Sensor Data Repository using the *Edge Server Instance List* page, which you access using the *other servers* icon (Figure 3–4).

The page's *Edge Server Instance List* table lists the entries for the other Oracle Sensor Edge Servers that are connected to the same Sensor Data Repository. These entries enable you to identify and access the other server instances. Using this table, you can both access another Oracle Sensor Server instance, and view the running status of other Oracle Sensor Server instances. In addition you can edit and delete the entries for other Oracle Sensor Server instances.

To access another Oracle Sensor Server instance, click the name of the instance in the *Server Name* column of the *Edge Server Instance List* table and then enter the OC4J administrator name and password in the login page that appears.

Clicking the table's notepad and trash can icons enable you to edit or delete an entry for a selected Oracle Sensor Edge Server instance, respectively. The fields in the page's *New Server Instance* section enable you create an entry for an Oracle Sensor Edge Server instance, which is comprised of the instance's name and its location as described in "Creating an Entry for an Oracle Sensor Edge Server Instance".

Note: Because live Oracle Sensor Edge Server instances create their own entries once they are started, you are not required to create an entry for an Oracle Sensor Edge Server instance.

Figure 3–3 The Other Servers Icon

Creating an Entry for an Oracle Sensor Edge Server Instance

To create an entry for an Oracle Sensor Edge Server instance:

1. Click the *other servers* icon (Figure 3–4). The *Edge Server Instance List* page appears (Figure 3–3), which includes a table listing the Oracle Sensor Edge Server instances that are connected to the same Sensor Data Repository.
2. Enter a name for the Oracle Sensor Edge Server instance.
3. Enter the URL that points to the Oracle Sensor Edge Server instance. Enter the URL in the following format:

   ```
   http://<hostname/ip address>:<port>/edge
   ```

The <port> value is the port on which Oracle Application Server Containers for J2EE (OC4J) listens, which is located on the machine running the Oracle Sensor Edge Server instance. For a 10.1.3 OC4J instance, find the port value by navigating to oracle_home/opmn/bin and then running the following command:

```
opmnctl status - l
```

For a 10.1.2 OC4J instance, the OC4J listener port is generally 8888. For more information on OC4J 10.1.2, see *Oracle Application Server Containers for J2EE User's Guide* and "Installing OC4J".

4. Click **Create Entry**. The new Oracle Sensor Edge Server instance appears in the *Edge Server Instance List* table.

Editing an Entry for an Oracle Sensor Edge Server Instance

To edit an entry, first click the notepad icon for the selected Oracle Sensor Edge Server entry. The editing page then appears, which with its *Server Name* and *URL* fields populated with the information set for the selected entry. Perform either (or both) of the following:

- Enter a new name for the entry.
- Change the URL for the entry.

Click **Update Entry** to commit the changes, or click **Cancel** to set the entry back to its original state.

Monitoring the Performance of the Oracle Sensor Edge Server Instance

The *Usage Statistics* section of the *Sensor Edge Server Instance* page (Figure 3–1) lists the following performance metrics for the Oracle Sensor Edge Server Instance:

- Events Received -- The number of inbound instruction events received by the Oracle Sensor Edge Server instance. For more information on inbound events, see "Monitoring the Event Data".
- Events Generated -- The number of outbound events sent from the Oracle Sensor Edge Server instance. For more information on outbound events, see "Monitoring the Event Data".
- Events Sent -- The total number of events dispatched by the Oracle Sensor Edge Server instance.
- Queued Events -- The number of events that are currently waiting in the queue.

Clearing the Queue of the Event Data

Clicking **Clear Queue** enables you to remove all of the queued events in the system (and sets the number displayed for *Queued Events* to 0). Use this function to purge old, backed-up event messages before setting up and starting a dispatcher or before restarting the Oracle Sensor Edge Server instance.

WARNING: Once you click Clear Queue, you cannot recover purged event data.

Setting the General Information for the Oracle Sensor Edge Server Instance

The *General Settings* section of the *Main* page enables you to edit the basic information for the Oracle Sensor Edge Server instance (described in Table 3–2). Click **Save Changes** to commit the updates to the Oracle Sensor Edge Server.

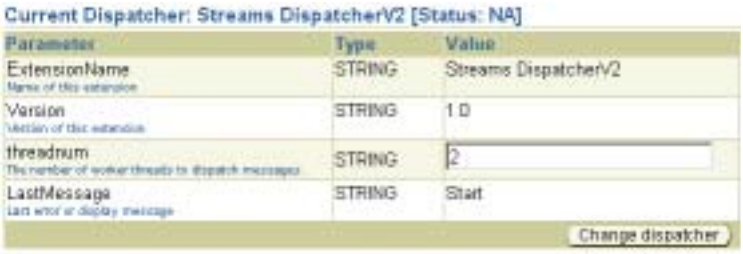
Table 3–2 The General Settings

Parameter	Description
Server Name	A name for the Oracle Sensor Edge Server instance.
Site Name	The site name for the Oracle Sensor Edge Server. This parameter is a grouping mechanism to logically distinguish between Oracle Sensor Edge Server instances.
Internal Queue	Event messages sent from readers (or any device) are enqueued into the internal queue. A dispatcher takes events from this queue and then dispatches them. For example, the Streams Dispatcher dispatches the event messages from this queue to a database. The following options enable you to protect data by controlling how event messages are held before they are dispatched and safeguard event data. . Options include: ■ Persist -- Selecting the *Persist* mode stores event messages to a disk where they are then gathered by a dispatcher. Storing these event messages to a disk prevents them from becoming lost if the Oracle Sensor Edge Server instance crashes. ■ Memory -- Select this option if either the database or the connection is slow to ensure that event messages do not back up when the devices generate them faster than they can be dispatched. When you select *Memory*, the Oracle Sensor Edge Server instance holds these messages in memory which creates less overhead because it does not write the message to a disk; however, the event messages are lost if the Oracle Sensor Edge Server instance crashes.
Log Level	A list of the following error logging options that set the level of severity for the messages written to the log file: ■ Error ■ Warning ■ Notify ■ Monitor ■ Debug The error messages written to the log file reflect not only the log level that you select, but also the log levels that are of greater severity than the log level that you select. The level you select affects the data that displays in the *View Log* tab. See also "Viewing Log Information".
Use Archive	Selecting this option enables the events to be saved to the Sensor Data Repository.
Shutdown Timeout	Enter the time (in milliseconds) that the Oracle Sensor Edge Server waits before shutting down threads that are not functioning properly.

Setting the Dispatcher for the Oracle Sensor Edge Server Instance

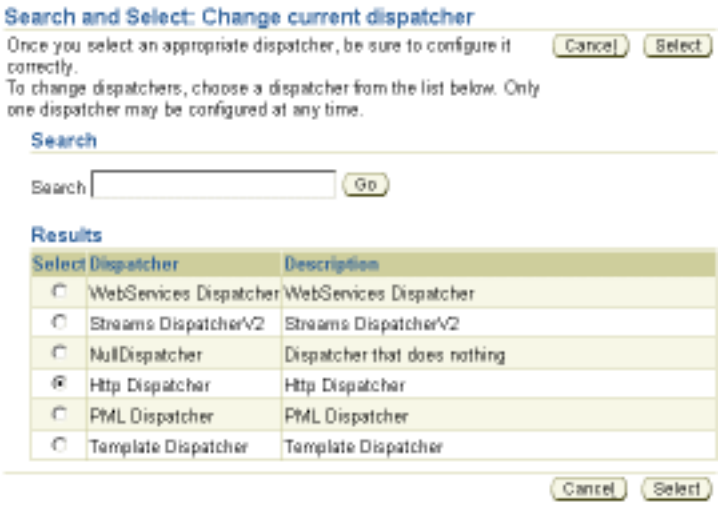
Clicking the **Change Dispatcher** in the *Main* page (Figure 3–5), enables you to change the dispatcher used by the Oracle Sensor Edge Server. See also "Managing Dispatchers for an Oracle Sensor Edge Server Instance".

Figure 3–4 Changing Dispatchers



From the *Search and Select* page (Figure 3–6) that appears, you can select an edge dispatcher to send event messages using such means as remote Web Services, Oracle Streams, or to a client application using HTTP.

Figure 3–5 Selecting a New Dispatcher



An Oracle Sensor Edge Server instance uses only one dispatcher at a time. After you assign a dispatcher as current, the Oracle Sensor Edge Server instance must be restarted. See "Starting and Stopping the Oracle Sensor Edge Server Instance".

Viewing Dispatchers, Drivers, and Filters

Expanding the *Available Extensions* folder in the tree (Figure 3–7) displays the dispatchers, filters, and drivers that are available to the Oracle Sensor Edge Server instance. Dispatchers forward events sent to the Oracle Sensor Edge Server instance to either a dispatching layer or directly to an application. Drivers enable communication between a device (such as reader) and the Oracle Sensor Edge Server instance and the filters, which generally either remove unwanted events (such as duplicates) or both remove and translate one or more events into a high-level event.

Figure 3–6 The Available Extensions Folder

Clicking an extension, such as a driver, displays the default properties of the selected driver. The values set for the parameters are read-only and do not represent the current configuration of these objects. You cannot configure them because the Oracle Sensor Edge Server instance does not use live instances of the extensions. Instead, the Oracle Sensor Edge Server instance reads and cleanses event data using instances of the of the filters and drivers listed in the *Extensions* folder. To create these filter instances and driver instances (known as devices), you must create a device group. See "Configuring Devices, Filter Instances, and Dispatchers" for more information on creating device groups. See "Setting the Dispatcher for the Oracle Sensor Edge Server Instance" for more information on setting the dispatcher for the Sensor Edge Server.

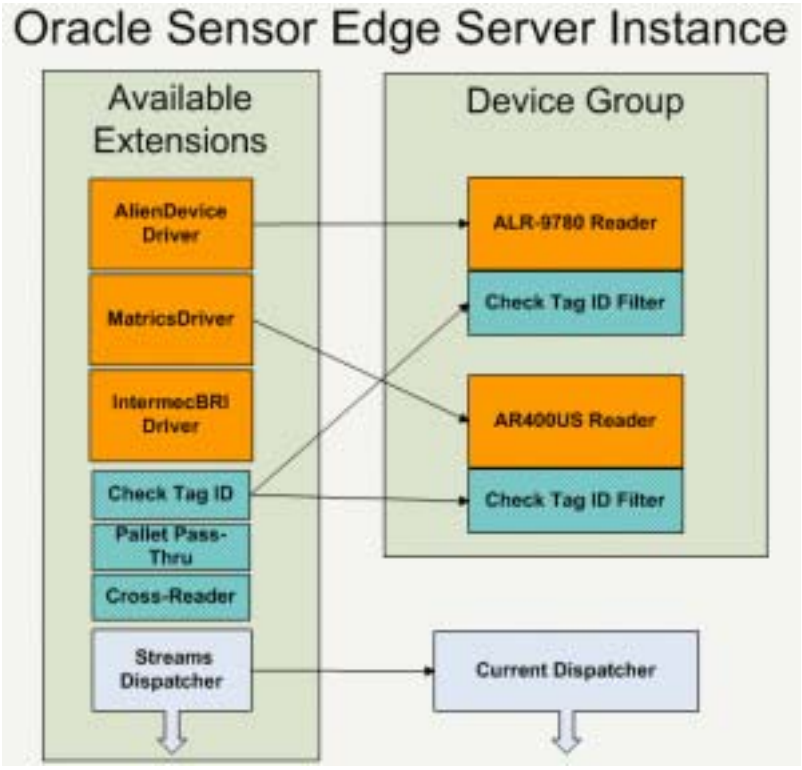
Setting the Devices and Filters Used by the Oracle Sensor Edge Server

To enable the Oracle Edge Sensor Server instance to receive, filter and dispatch event data, you must first create a device group, a logical grouping of devices (the instances of drivers) and the filters associated with these devices. An Oracle Sensor Edge Server instance can have one or many device groups instantiated. Each device group is responsible for all of the devices (and their associated filters) included within it.

Device groups make one or more devices into a logical device or "group." You use device groups to associate devices for processing as a single streams of events. For example, If you create device group called *Warehouse Exits* consisting of all of the devices placed at all of the exits of a warehouse (and if needed, add a filter instance to this group), then all of the generated events are viewed as originating from one logical device rather than from several devices.

You can group devices in terms of management if you want to treat them as a logical unit to manage (such as in the case of the aforementioned *Warehouse Exits* device group), or you can group them by the type of filtering they perform (as illustrated in Figure 3–8). For example, if you group devices by cross-reader filtering, then you create a group of related devices and then attach filters to that group. For more information, see "Managing the Filter Instances for a Device or Device Group"

Figure 3–7 Reader Devices Grouped by Filter



Viewing the Device Groups of the Oracle Sensor Edge Server

Clicking the *Groups* folder invokes the *Group Management* page (Figure 3–9), which lists the device groups and their respective filters. In addition, the page lists the running status of the devices for each group. This page also enables you to create a new device group (as described in "Creating a Device Group"). To view the configuration of a specific device group, expand the *Groups* folder and then select the appropriate node to invoke the *Configure Group* page for the selected device group. The default device group is called *Unassigned* and is a special reserved group. For more information on *Configure Group* page, see "Editing a Device Group".

Figure 3–8 Configuring a Device Group

Creating a Device Group

Creating a device group is the first step to connecting the Oracle Sensor Edge Server instance to devices and filters. Once you create a device group, you populate it with devices (the instances of the available drivers) and then attach filter instances to the individual devices (or to the entire device group). To create a new device group:

1. In the *Group Management* page, enter a name for the device group in the *Group Name* field and then click **Create New Group**. The *Configure Group* page appears for the new device group.

Figure 3–9 The Configure Group page

2. Create a device (a driver instance) for the device group by clicking **Add new device**. The *Search and Select* page appears, listing the drivers in the repository.

Figure 3–10 Selecting a Driver for the Device

select_device.gif illustrates the Search and Select page for devices.

****

Tip: Locate a driver by entering the name of the driver (or part of the driver name) in the *Search* field and then by clicking **Go**.

a. Select a driver for the device. Table 3–3 describes the drivers supported by Oracle Sensor Edge Server. For descriptions of the these drivers, the configuration parameters required to create devices from them, and the models supported by these drivers, see Chapter 6, "Configuring Devices, Filter Instances, and Dispatchers".

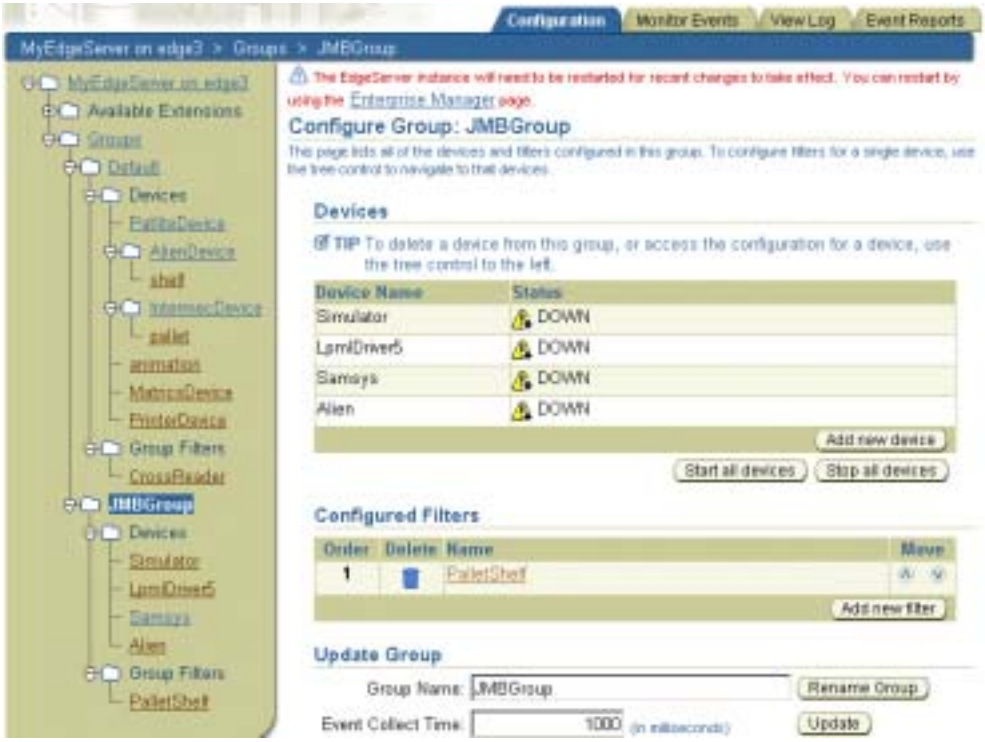
Table 3–3 Supported Drivers

Readers	Indicators, Notification and Display	Printers	Other
The drivers that support readers include: Alien Reader Driver (See "Configuring Alien Reader Driver-Based Devices".) BarcodeDriver (See "Configuring BarcodeDriver-Based Devices".) Intermec BRI Driver (See "Configuring Intermec BRI Driver-Supported Devices".) Intermec Reader Driver (See "Configuring Intermec Reader Driver-Based Devices".) Matrics Driver (See "Configuring Matrics Driver-Based Devices".) Samsys Driver (See "Configuring Samsys Driver-Based Device".) Tyco Reader Driver (See "Configuring Tyco Reader Driver-Based Devices".)	The drivers that support notification and display include: AnimationDriver (See "Configuring an Instance of the AnimationDriver".) ConsoleDriver (See "Configuring an Instance of the ConsoleDriver".) Edge Simulator Driver (See "Configuring Edge Simulator Driver-Based Devices".) PatliteDriver (See "Configuring PatliteDriver-Based Devices".) Prolite Driver (See "Configuring Prolite Driver-Based Devices".)	LpmlDriver (See "Configuring LpmlDriver-Based Devices".)	Edge Echo Driver (See "Configuring Edge Echo Driver-Based Instances".)

b. If needed, enter a name for the device in the *New Device Name* field. If you do not assign a name to the device, then Oracle Sensor Edge Server assigns a default name.

c. Click **Select**. The *Configure Group* page reappears, listing the device in the *Devices* section. The navigation tree also displays the device.

Figure 3–11 Devices Added to a Device Group

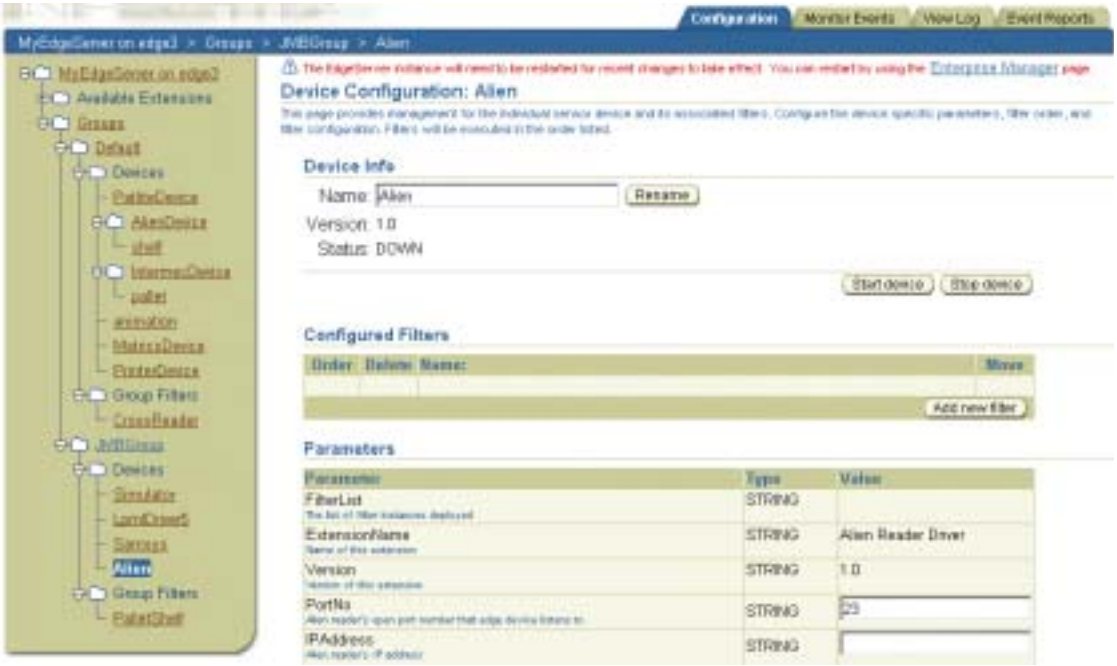


3. Configure the device by first selecting it from the tree. The *Device Configuration* page appears (Figure 3–13), displaying the parameters specific to the driver.

   a. Define the parameters. For more information on the driver parameters, see "Configuring Devices".

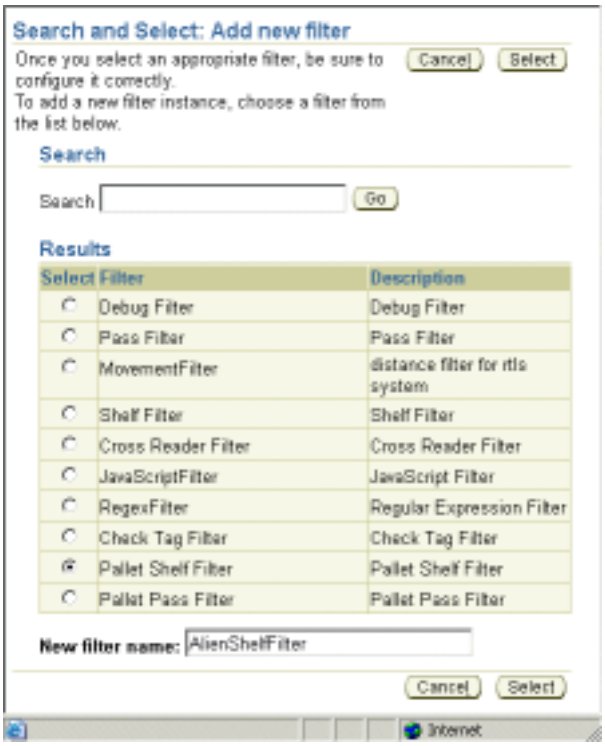
   b. Click **Save Changes**.

Figure 3–12 Configuring a Device for a Device Group



4. Add filters (filter instances) to the device by clicking **Add new filter**. The *Search and Select* page appears (Figure 3–14), listing the filters in the repository. Table 3–5 lists the Oracle Edge Sensor Server's seeded device- and group-level filters.

Figure 3–13 Selecting a Filter



a. Select a filter instance for the device.

Tip: Locate a filter by entering the name of the filter (or part of the filter name) in the *Search* field and then by clicking **Go**.

b. If needed, enter a name for the device in the *New Filter Name* field. If you do not assign a name to the filter instance, then Oracle Sensor Edge Server assigns a default name.

c. Click **Select**. The selected filter appears in the *Configured Filters* section of the device's configuration page (Figure 3–15) and in the tree under the device.

Figure 3–14 Adding Filter Instances to a Device

5. Configure the filter instance by first selecting the filter instance from the table in the *Configured Filters* section of the *Device Configuration* page, or from the tree. The *Filter Configuration* page appears (Figure 3–16), displaying the parameters of the selected filter.

a. If needed, rename the filter instance.

b. Define the filter parameters. For more information on filter parameters, see "Configuring Filter Instances".

c. Click **Save Changes**.

Note: You must stop and then restart the Oracle Sensor Edge Server after you create, edit or delete a device group. For more information, see "Starting and Stopping the Oracle Sensor Edge Server Instance".

Figure 3–15 Creating a Filter Instance for a Device

Adding a Filter to a Device Group

If needed, assign a filter to the device group as follows:

1. Select the device group from the tree. The *Configure Group* page appears.
2. Click **Add new filter**. The *Search and Select* page appears (Figure 3–14), listing the filters in the repository. Table 3–5 lists the device- and group-level filters of the Oracle Edge Sensor Server.
3. Select a filter instance for the device group.

   Tip: Locate a filter by entering the name of the filter (or part of the filter name) in the *Search* field and then by clicking **Go**.

4. If needed, enter a name for the device in the *New Filter Name* field. If you do not assign a name to the filter instance, then Oracle Sensor Edge Server assigns a default name.
5. Click **Select**. The filter appears in the *Configured Filters* section of the *Configure Group* page and in the tree under *Group Filters* for the selected device group.
6. Configure the filter instance by first selecting the filter instance from the table in the *Configured Filters* section of the *Configure Group* page, or from the tree. The *Filter Configuration* page appears (Figure 3–16), displaying the parameters of the selected filter.
7. If needed, rename the filter instance.
8. Define the filter parameters. For more information on filter parameters, see "Configuring Filter Instances".
9. Click **Save Changes**.

Editing a Device Group

The *Configure Group* page (Figure 3–9) enables you to edit the properties of a device group. This page, which appears when you click a device group in the navigation tree (Figure 3–2), enables you to do the following tasks:

- Renaming a Device Group
- Updating the Devices and Filters Used by a Device Group
- Starting and Stopping the Devices Assigned to a Device Group
- Deleting a Device Group

Renaming a Device Group

To rename a device group:

1. Enter a new name for the device group in the *Group Name* field.
2. Click **Rename Group**. The new group name appears in the tree.

Updating the Devices and Filters Used by a Device Group

You can add devices and filter instances to a device group using the **Add new device** and **Add new filter buttons** described in "Creating a Device Group". Once you complete the device and filter instance assignments, click **Update**.

Starting and Stopping the Devices Assigned to a Device Group

You must stop and then start device groups whenever you perform such tasks as adding new devices or filter instances. You start and stop the devices belonging to a device group using the **Start all devices** and **Stop all devices** buttons.

Deleting a Device Group

The **Delete Group** button enables you to remove a device group from the Oracle Sensor Edge Server. When you delete a device group, you also remove all of the devices and filter instances that have been configured for the group. Once you delete a device group, you must re-start entire Oracle Sensor Edge Server for the change to take effect.

Starting and Stopping the Oracle Sensor Edge Server Instance

Once you have modified an Oracle Sensor Edge Server instance, you must restart the Oracle Sensor Edge Server to commit the changes. You can stop and restart the Oracle Sensor Edge Server using either of the following methods:

- Stopping and Starting an Oracle Sensor Edge Server Instance Using opmnctl
- Restarting an Oracle Sensor Edge Server Instance Using the OracleAS Enterprise Manager

Stopping and Starting an Oracle Sensor Edge Server Instance Using opmnctl

After you change the Oracle Sensor Edge Server's general settings, its dispatcher, or edit any device group, you can stop and restart the Oracle Sensor Edge Server using the `opmnctl shutdown` and `opmnctl startall` commands. For more information, see *Oracle Process Manager and Notification Server Administrator's Guide*.

Restarting an Oracle Sensor Edge Server Instance Using the OracleAS Enterprise Manager

When you change anything in an Oracle Sensor Edge Server instance, the SES Console displays a message (Figure 3–17) notifying you to restart the Oracle Sensor Edge Server instance using OracleAS Enterprise Manager. This message includes a link to OracleAS Enterprise Manager.

Figure 3–16 The Restart Oracle Enterprise Manager Message

To restart an Oracle Sensor Edge Server instance:

1. Click the *Enterprise Manager* link in the message (Figure 3–17). The login page for OracleAS Enterprise Manager appears.
2. After you enter the OC4J user name and password, the cluster topology page appears.
3. Navigate to the *Home* page (Figure 3–18).
4. Click *Applications*.
5. Select **edge**.
6. Click **Restart**.

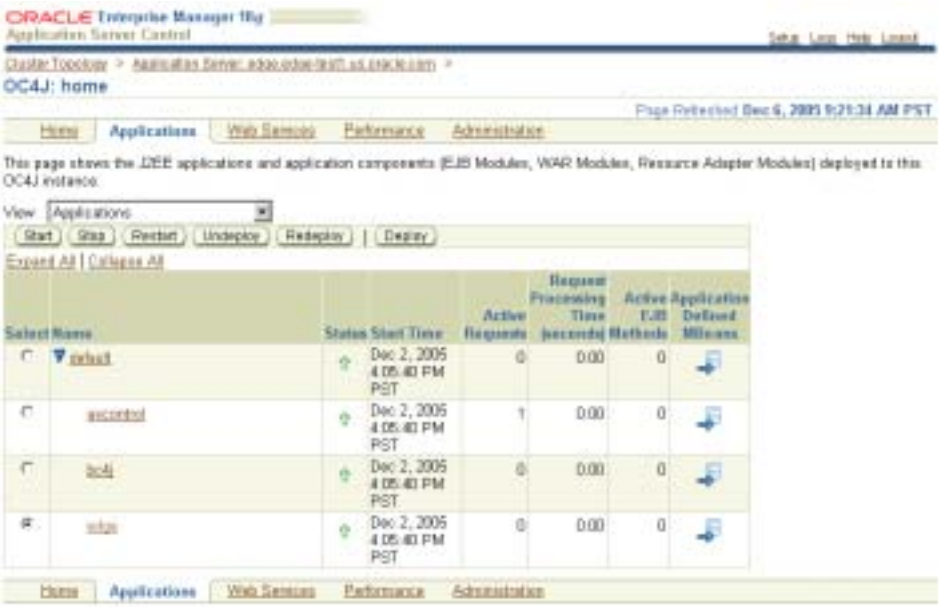
Caution: Error messages regarding invalid jms.xml typically occur because of an abnormal termination of OC4J, an OC4J crash, or the IP address of the server running OC4J changes.

If you encounter OC4J JMS Server startup problems after an abnormal shutdown, first check that no other OC4J JMS Server is running and using the same persistence files. Remove any .lock files from the ORACLE_HOME/j2ee/instance_name/persistence directory and try restarting again.

If problems persist, confirm that the jms.xml file is valid.

If problems still persist, remove the jms.state file from the persistence directory and try again. Removing this file may result in the loss of transaction information. See also the section entitled "Abnormal Termination", located in the "Resource Providers" in "Chapter 3: Oracle Enterprise Messaging Service (OEMS)" of *Oracle Containers for J2EE Services Guide*.

Figure 3–17 The Home Page of OracleAS Enterprise Manager



Note: To restart OC4J 10.1.2, stop OC4J through the SES Console and then start OC4J again using `Java_HOME/bin/java -jar oc4j.jar`.

Starting and Stopping an Individual Device

You must first stop a device before you update it. You can stop or start a device from the *Device Configuration* page (Figure 3–13) using the **Start Device** and **Stop Device** buttons. The page also displays the status of the device. The status messages (described in Table 3–4) describe the current state of the driver, or its state before it crashed.

Table 3–4 Device Status Messages

Status	Definition
Instantiated	The device successfully initialized itself (that is, it passed the `doInit()` entry point.
Initialization Failed	The initialization failed; the device is down (that is, the `doInit()` method failed).
Start	The device is started (it just entered the `doStart()`method).
Start Failed	The start failed; the `start()` method failed.
Stop	The device has stopped.
Stop Failed	The device failed to stop or clean up resources.
Finished	The device has completed processing. This is used by a device that has a fixed set of tasks and usually stops by itself when it completes these tasks, such as the Simulator when it finishes with the simulator file.
Connection	The device is attempting to connect.
Connection Failed	The device failed to connect.

Table 3–4 (Cont.) Device Status Messages

Status	Definition
Setup	The device is setting up resources and connections.
Setup Failed	The device failed during setup.
Run	The device is running.
Run Failed	The device failed while running.
Disabled	The device has been disabled.

Managing Filters

Filters can be attached to either a specific device or to a device group. Some filters, such as the Cross-Reader Redundant filter, are written as group-level filters and can only be attached to a device group. A device group filter provides filtering of events before they are delivered to an edge device. While some filters are written only for device groups, others are written only for device-level filtering and only function if they are attached to a specific device.

The *Configured Filters* tables of the *Device Configuration* (Figure 3–12) and *Configure Group* (Figure 3–15) pages enable you to add, delete, or reorder the filter instances. The **add new filter** button enables you to add additional filter instances to a device or to a device group. For more information, see "Creating a Device Group" and "Adding a Filter to a Device Group". Clicking the table's trash can icon enables you to remove a filter instance from a device or device group. The table's arrows enable you to prioritize the filter instances.

Prioritizing Filter Instances

The order of filter instances affects the efficacy of the data filtering. For example, if a device or device group is assigned a group filter, which groups IDs into an array of events and treats them as one item and a tag filter the filters out the events for a specific tag, *TagXYZ*, then applying the group filter before the tag filter results in the Oracle Sensor Edge Server receiving events grouped into chunks based on when they were detected, but only after the tag filter has strained out the events for *TagXYZ*. Reversing the order of the filters (that is, putting the group filter before the tag filter) would prevent the tag filter from filtering out anything, because it would see only the group events and not those for *TagXYZ*.

Using the arrows in the *Configured Filters* table (Figure 3–15), you can arrange the filter instances by selecting a filter instance and then moving it up or down using the arrows.

Managing the Filter Instances for a Device or Device Group

A filter instance is an instantiated object of a filter. Whenever a filter is applied to a device (or to a device group), a filter instance is created, enabling the device or device group to use the filter.

Although you can develop your own filters and then upload them (see "Adding Extensions to the Oracle Sensor Edge Server Instance"), Oracle Sensor Edge Server ships with several filters (described in Table 3–5).

Table 3–5 The Pre-Seeded Filters of the Oracle Sensor Edge Server

Filter Name	Function	Applied to Device Group? (Supports Group-Level Filtering)	Applied to Devices? (Supports Device-Level Filtering)
Check Tag ID Filter	A diagnostic tool that checks if a device is reading tags during a specified interval. See also "Configuring the Check Tag ID Filter".	No	Yes
Cross-Reader Redundant Filter	Blocks redundant events that are sent from the devices belonging to a device group. See also "Using the Cross-Reader Redundant Filter".	Yes	No
Debug Filter	Tracks events passing through the system and then writes these events to a log file. See also "Using the Debug Filter".	Yes	Yes
JavaScript Filter	Enables you to write filter logic in a scripting language. Changes made to the source scripts are loaded dynamically, thus eliminating the need to restart the server or any components. This filter relies on an external scripting engine that executes the script, such as Mozilla Rhino (`http://www.mozilla.org/rhino/`). See also "Configuring the JavaScript Filter".	Yes	Yes
Movement Filter	Smooths out movement tracking using **Real Time Location System (RTLS)** observations by averaging out spikes or sudden motion changes from errors or interference. See also "Configuring the Movement Filter".	No	Yes
Pallet Pass-Thru Filter	Enables you to see all of the tag IDs for items held in a container or on a pallet. See also "Configuring the Pallet Pass Thru Filter".	No	Yes
Pallet Shelf Filter	Sends events that signal new containers or pallets entering or exiting the field or gateway of a device reader. See also "Configuring the Pallet Shelf Filter".	No	Yes
Pass Filter	Notifies applications that a tag has passed through a device's gateway or range or transmission. This filter also blocks events, so that only one event, rather than duplicate events, are generated when a tag is detected by a device. See also "Configuring the Pass Filter".	No	Yes

Table 3–5 (Cont.) The Pre-Seeded Filters of the Oracle Sensor Edge Server

Filter Name	Function	Applied to Device Group? (Supports Group-Level Filtering)	Applied to Devices? (Supports Device-Level Filtering)
Polygon Filter	Filters out all movement observations reported by **Real Time Location System (RTLS)** devices and generates events only if the tag moves in or out of any predefined polygons. The polygons are defined using a set of *x*, *y* coordinates that define the vertices and parenthesis. For example: ((x,y), (x,y), ...), (....), ... See also "Configuring the Polygon Filter".	No	Yes
Regex Filter	Performs a regular expression search that looks for tags to either remove or to allow to pass through the streams. This filter enables you to define a set of patterns for the filter to search for in any of the event's fields. When the filter finds matches to the search criteria, it allows the event to pass through the system; if it finds no matches, it filters out the event. See also "Configuring the Regex Filter".	Yes	Yes
Shelf Filter	Signals that an item has entered, or exited the field or gateway of a device reader. See also "Configuring the Shelf Filter".	No	Yes

Monitoring the Event Data

Devices and filter instances communicate using events, messages that describe what has occurred. For example, a device informs other components that it has started by sending such a message. These event messages are specific to each type of device or filter instance. Within the Oracle Sensor Edge Server, any device can send or receive an event directly. In general, the event flow between the components follows two directions:

- Inbound -- The devices and filter instances send events to the current dispatcher.
- Outbound -- Applications send events to the devices and filter instances. These events are sometimes known as instruction events, since they often send commands to a device.

Viewing Event Data

The SES Console enables you to view the health of the Oracle Sensor Edge Server by viewing the event data that displays in the *Monitor Events* and *Events Reports* tab. The *Monitor Events* tab (Figure 3–21) enables you to view the data currently in the queue, while the *Events Reports* tab enables you to view the data that has been stored in the Sensor Data Repository.

Table 3–6 describes the inbound and outbound fields that display in the *Monitor Events* and *Events Reports* tabs.

Table 3–6 inbound and Outbound Event Data

Topic	Description
Type	A text representation of the Event Type. Types include *RFID Observation*, *RTLS* Observation, and *Temperature*.
Description	A text representation of the Event Subtype.
Device Name	The name of the device that generated the event
Data	The payload of the event.
Time	The time that the event was generated.

Viewing an Individual Event

The SES console enables you to view further information about an individual inbound, or outbound event, such as its *Event Type*, *Subtype* and *Correlation ID* fields. To access this information, click the *Details* icon.

Figure 3–18 The Details Icon

The *Event Details* page appears, which displays the Metadata and Payload information for a selected event.

Figure 3–19 The Event Details Page

Metadata

The *Metadata* section of the *Event Details* screen lists the following routing information and information related to the origin of the event:

- Event ID

  The meaning of Event ID differs in terms of unprocessed event data and event data that has been stored in the Sensor Data Repository. In terms of unprocessed data (that is, the data viewed from a detail page from the *Monitor Reports* tab which is depicted in Figure 3–21), Event ID refers to the tag ID that triggered the event. When you view the event data that has been stored in the Sensor Data

Repository which displays in the *Event Report* pages (Figure 3–23, Figure 3–24, and Figure 3–25), the *Event ID* field uniquely identifies an event.

- Device Name

  The name of the extension (device or filter instance) or application that generates the event.

- Source Name

  This field identifies the originator of the event. This is an optional field, one set by the client.

- Site Name

  The site that originally generated the message.

- Type

  The number value that corresponds to the type of event generated. Event types are grouped as follows:

  – 0 - 99: System Messages

  – 100 - 199: Generic Instructions to Devices

  – 200 - 299: Observations from Devices

  – 500 - 599 Custom messages (not registered)

  Table 3–7 describes the registered event types that display in the *Type* field.

- Subtype

  The number value that corresponds to the subtype of event. Table 3–7 describes the values that display in the *Subtype* field.

- Description

  A text description of the Event Type and its Event Subtype. For example, if the event subtype is 200 (*RFID Observation*) and its subtype is 2 (*tag exits field*), then the *Description* field displays *RFID Outfield*. If the Event Type is 200 and the Event Subtype is 9 (that is, no explicit message), then the *Description* field displays the event type 200 message (*RFID Observation*) followed by the display of the Event Subtype as *RFID Observation (subtype 9)*. Table 3–7 notes the text representations that display in the *Description* field.

- Correlation ID

  A unique ID that identifies this thread of events. The correlation ID is used for message responses to a particular client (such as checking if a device functions). Any message response sent back by the client has the same ID. This ID, which is set by the client, correlates the sent event message to the received event message so that it cannot be used as a parameter in the device. This is an optional field.

Table 3–7 Registered Event Types and Related Subtypes

Event Type	Description	Type Description (As displayed in the Event reports)	Subtypes
0	Unknown; a value of 0 represents a bad event or a system internal event.	System event	N / A
1	Instructions or commands; message events.	Instruc return code	Subtypes include: ■ 1 and 2 -- Error message ■ 3 -- Function not supported ■ 4 -- Oracle Sensor Edge Server startup message ■ 5 -- Oracle Sensor Edge Server shutdown message. ■ 10 -- Notification message
100	General instructions.	Instruction	Subtypes include: ■ 0 -- Get device status. ■ 1 -- Start a device. ■ 2 -- Stop a device.
101	RFID instructions.	Instruc write	Subtypes include: ■ 0 -- Write to a tag. ■ 1 -- Destroy a tag. ■ 2 -- Get field strength. ■ 3 -- Read tag payload.
102	Printer instructions.	Instruc print	Subtypes include: ■ 1 -- Print using LPML. This subtype displays as *Instr print label* in the *Event Details* page. ■ 2 -- Print raw (direct payload). This subtype displays as *Instr print raw* in the *Events Details* page.
103	Lightstack instructions.	Indicator	Subtypes include: ■ 1 -- Lightstack XML command. This subtype displays as *Indicator play* in the *Event Details* page. ■ 2 -- Displays as *Indicator response* in the *Event Details* page. ■ 3 -- Displays as *Indicator clear* in the *Event Details* page.
104	Audio	Audio	1 -- Displays as *Audio instruction* in the *Event Details* page.

Table 3–7 (Cont.) Registered Event Types and Related Subtypes

Event Type	Description	Type Description (As displayed in the Event reports)	Subtypes
200	RFID	RFID observation	Subtypes include: ■ 0 -- Normal. ■ 1 -- Tag enters field. Displays as *RFID observation* in the *Event Details* page. ■ 2 -- Tag exits field.Displays as *RFID outfield* in the *Event Details* page. ■ 3 -- Tag pass through field. Displays as *RFID pass* in the *Event Details* page. ■ 4 -- Tag group enters field. Displays as *RFID Pallet in field* in the *Event Details* page. ■ 5 -- Tag group exits field. Displays as *RFID pallet out field* in the *Event Details* page. ■ 6 -- Tag group pass through field. Displays as *RFID pallet pass* in the *Event Details* page. ■ 7 -- Container event. Displays as *RFID Container* in the *Event Details* page.
201	RTLS	RTLS	Subtypes include: ■ 1 -- Displays as *RTLS observation* in the *Event Details* page. ■ 2 -- Displays as *RTLS in polygon* in the *Event Details* page. ■ 3 -- Displays as *RTLS out polygon* in the *Event Details* page. ■ 4 -- Displays as *RTLS moved* in the *Event Details* page.
202	Physical contact	Physical contact	1 -- Displays as *physical disconnect* in the *Event Details* page.
203	Temperature	Temperature	Subtypes include: ■ 1 -- Displays as *Temperature reading* in the *Event Details* page. ■ 2 -- Displays as *Temperature change* in the *Event Details* page.
204	Humidity	Humidity	Subtypes include: ■ 1 -- Displays as *Humidity reading* in the *Event Details* page. ■ 2 -- Displays as *Humidity change* in the *Event Details* page.
205	Weight	Weight	1 -- Displays as *Weight reading* in the *Event Details* page.
206	Tampering	Tampering	1 -- Displays as *Tampered with* in the *Event Details* page.

Table 3–7 (Cont.) Registered Event Types and Related Subtypes

Event Type	Description	Type Description (As displayed in the Event reports)	Subtypes
208	Message Board	Message board	N / A
209	PLC	Automation	N / A
210	Printer Response	Printer Response	2 -- Print Successful. Displays as *Print Successful* in the *Event Details* page.
211	Hazardous Sensors	Hazardous	N / A
212	Barcode	Barcode	1 -- Displays a *Barcode read* in the *Event Details* page.
213	Radiation	Radiation	N / A

Payload

The *Payload* section of the *Event Details* page displays the following fields:

- Tag ID

  The identity of the item described in this event. The text value of this field identifies a tag (that is, a read or target) to an event instruction.

- Data

  The tag data (or payload of the event). This is an optional field.

- Timestamp

  The date and time when the event was generated.

Viewing Unprocessed Event Data

The SES Console enables you to view this event data in real-time from its *Monitor Events* tab. Clicking the *Monitor Events* tab displays the *Monitor Events* page (Figure 3–21).

Figure 3–20 The Monitor Events Page

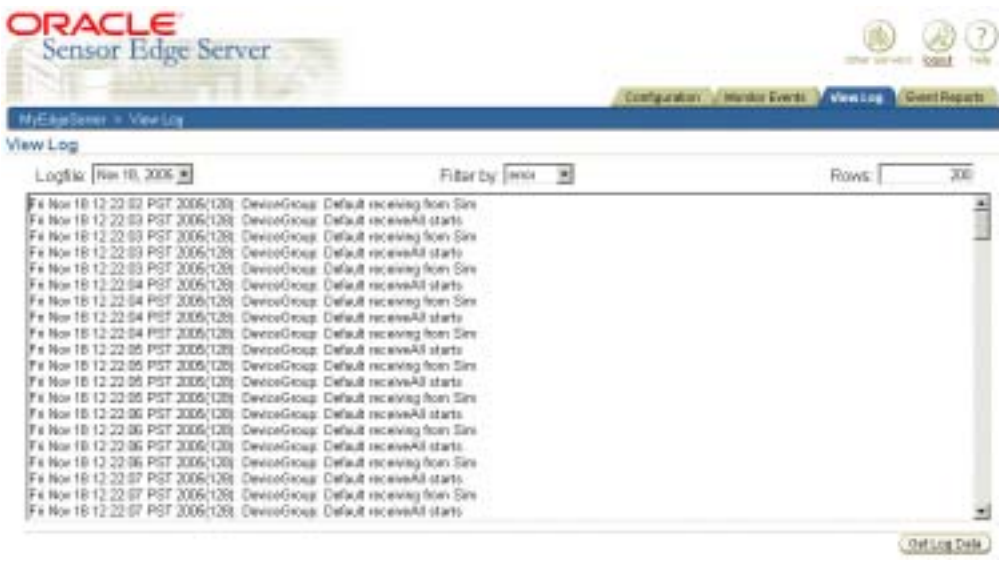
The *Monitor Event* page enables you to view all of the inbound event data and the outbound event data that passes through the queue. The event data displayed on this page has not been processed; it has not been sent to the Sensor Data Repository.

> **Tip:** The event data displayed in the *Monitor Events* page is dynamic and often-changing; if data remains static, then devices may not be sending events or receiving instruction events.

Viewing Log Information

The *View Log* page (Figure 3–22), which you access from the *View Log* tab, displays data written to the log according to the severity level selected from the *Log Level* list located on the *Main* page (Figure 3–1). For example, if you selected *Error* from the *Main* page, then the *View Log* page only displays error-level messages; if you selected *All* from the *Main* page, then the *View Log* page displays all of the logging data.

Figure 3–21 The View Log Page



Using the *View Log* page, you can select the log file by date and set the number of rows displayed in this page.

To view log data:

1. Select the date for the log file from the *Logfile* list.
2. If needed, select a log level from the *Filter by* list.
3. If needed, enter the number of rows for display in the *Rows* field.
4. Click **Get Log Data**.

Viewing Processed Event Data

The SES Console enables you to track the events that have been processed and stored in the Sensor Data Repository using the pages accessed through the *Event Reports* tab. These pages enable you to retrieve data based on tag ID, device name, or on the period during which the event passed through the queue. You can refine tag ID and device

name searches by building queries that include such search criteria as Event Type and Event Subtype.

When you click the Events Reports tab, the *View Tags* page (Figure 3–23) appears by default.

Figure 3–22 The View Tags Page

You can select other types of event searches using the navigation tree in the *View Tags* page. The tree appears in all of the pages accessed through the Events Reports tab.

Searching for Events by Tag ID

To retrieve events by tag ID:

1. Enter a portion of the tag ID. (This is a `like` pattern.)
2. Enter a starting date, or select a starting date using date-time editor. This is an optional condition; leaving this field blank excludes this condition from the search.
3. Enter a starting time (as *hh:mm:ss*). This is an optional condition; leaving this field blank excludes this condition from the search.
4. Enter an ending date, or select an ending date using the date-time editor. This is an optional condition; leaving this field blank excludes this condition from the search.
5. Enter an ending time (as *hh:mm:ss*). This is an optional condition; leaving this field blank excludes this condition from the search.

   Tip: Clicking within the *End Time* field populates the *End Date* field with the same value as that entered in the *Start Date* field.

6. Click **Fetch Results**. The search results appear in the *Results* table. Clicking the *Details* icon enables you view a specific event. For more information, see "Refining Tag ID and Device Name Searches". To add conditions to the search, click **Advanced Search**.

Searching for Events by Device Name

To retrieve events by device name:

1. Enter the device name (or a portion of the device name). This is a `like` pattern.
2. Enter a starting date, or select a starting date using the date-time editor. This is an optional condition; leaving this field blank excludes this condition from the search.
3. Enter a starting time (as *hh:mm:ss*). This is an optional condition; leaving this field blank excludes this condition from the search.
4. Enter an ending date, or select an ending date using the date-time editor. This is an optional condition; leaving this field blank excludes this condition from the search.
5. Enter an ending time (as *hh:mm:ss*). This is an optional condition; leaving this field blank excludes this condition from the search.

   Tip: Clicking within the *End Time* field populates the *End Date* field with the same value as that entered in the *Start Date* field.

6. Click **Fetch Results**. The search results appear in the *Results* table. Clicking the *Details* icon enables you view a specific event. For more information, see "Refining Tag ID and Device Name Searches". To add conditions to the search, click **Advanced Search**.

Refining Tag ID and Device Name Searches

The **Advanced Search** button on the *View Tags* and *View Device* pages enables you to narrow search results by building query statements.

To add a query statement to a device name or tag ID search:

1. Enter the tag ID or device name search criteria. If needed, add the time and date constraints for the search.
2. Click **Advanced Search**. The *Advanced Search* page appears, populated with the search criteria entered for either the tag ID or device name.

Figure 3–23 The Advanced Search Page (For a Tag Search)

3. Build the query statements as follows:

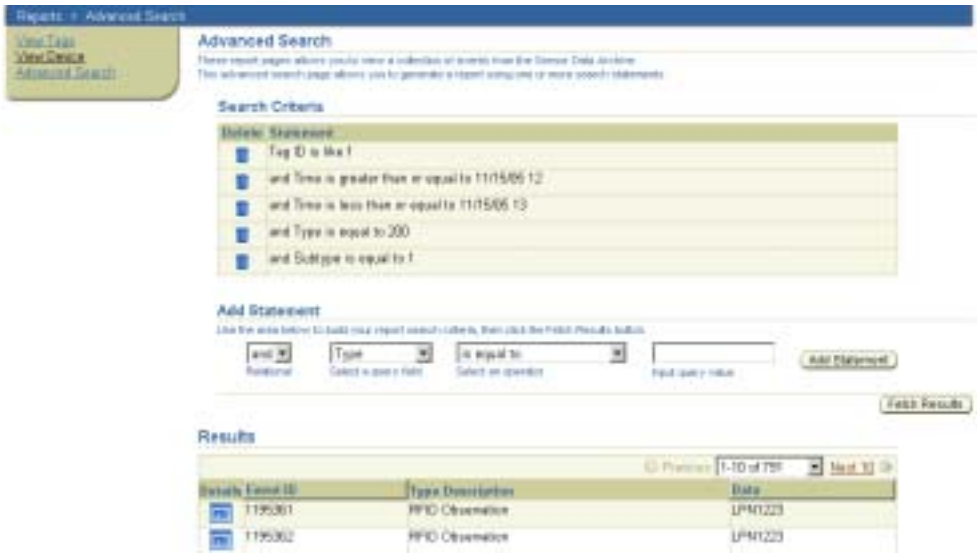
- Using the *Select a query field* list, select the event Metadata (such as *Event ID*, *Device Name*, *Source Name*, *Site Name*, *Type*, *Subtype* and *Correlation ID*) and the Payload (*Tag ID*, *Data*, and *Time*)
- Select an operator to compare the search value selected from the *Select a query field* list to the value entered in the *Input query value* field. Options include:
  - is equal to
  - is not equal to
  - is less than
  - is less than or equal to
  - is greater
  - is greater than or equal to
  - is like
- Enter a value in the *input query value* field that is relevant to the option selected in the *Select a query* field. For example, select *Type* from the *Select a query* field and then enter 200 in the *input query value* field.

4. Click **Add Statement**. The query statement appears in the *Search Criteria* statement section.
5. Click **Fetch Results**. The Sensor Data Repository is queried using the search statements. The requested data displays in the *Results* section of the page. You can sort the data in ascending and descending order by *Event ID* and by *Data*. To view an individual event, click the *Details* icon. For more information, see "Viewing an Individual Event".

Creating Advanced Searches

The *Advanced Search* page enables you retrieve specific event data by constructing a query comprised of statements to retrieve event data using the event data's Metadata and Payload information.

Figure 3–24 The Advanced Search Page



To create specific search criteria for event data:

1. Using the *Select a query field* list, select the event Metadata (such as *Event ID*, *Device Name*, *Source Name*, *Site Name*, *Type*, *Subtype* and *Correlation ID*) and the Payload (*Tag ID*, *Data*, and *Time*).
2. Select an operator to compare the search value selected from the *Select a query field* list to the value entered in the *Input query value* field. Options include:
   - is equal to
   - is not equal to
   - is less than
   - is less than or equal to
   - is greater
   - is greater than or equal to
   - is like
3. Enter a value in the *input query value* field that is relevant to the option selected in the *Select a query* field. For example, select *Type* from the *Select a query* field and then enter 200 in the *input query value* field.

   Tip: The *Relational* list appears once you complete a query and click **Add Statement**. This list enables you to bind the query statements using compound search conditions (*and*, *or*, *not*). Use the trash can icons to remove a query statement.

4. Click **Add Statement**. The search statement appears in the *Search Criteria* section.
5. If needed, add other statements.
6. Click **Fetch Results**. The Sensor Data Repository is queried using the selected search statements. The results display in the *Results* section of the page. You can sort the data in ascending and descending order by *Event ID* and by *Data*. To view an individual event, click the *Details* icon. For more information, see "Viewing an Individual Event"

Adding Extensions to the Oracle Sensor Edge Server Instance

An extension is a custom-built driver, dispatcher or filter which you upload to the Oracle Sensor Edge Server by packaging the component in an Extension Archive file. The Extension Archive file is a JAR file containing all of the class files and native binaries for the driver, filter, or dispatcher, as well as properties files or static data files. In addition, the Extension Archive includes the Extension Archive Descriptor file, an XML file containing instructions for the Oracle Sensor Edge Server on loading and managing the extension.

Note: Setting the element content of `<IsExtensionMonitorEnabled>` to `true` enables an extension to be dynamically uploaded to the Oracle Sensor Edge Server. You do not have to restart the Oracle Sensor Edge Server. However, for the Oracle Edge Sensor Server to use the instances created from an extension, you must restart the Oracle Edge Sensor Server as described in.

Extension Archive Files

Before you can upload a custom extension, such as a driver, dispatcher, or filter, you must package the extension files into an Extension Archive. An Extension Archive contains all of the extension's binaries, startup data, and configuration information. Each Extension Archive contains only one extension implementation, which is loaded at runtime. The Extension Archive contains the following directories:

- Meta-INF
- classes
- lib

Meta-INF

This directory contains any meta information about the archive. This directory must include the Extension Archive Descriptor file. The Extension Archive Descriptor file is an XML file located in the `META-INF` directory that contains the information needed by the Oracle Sensor Edge Server to load and manage the extension.

The Extension Archive Descriptor file is called `ext.xml`. Example 3–1 illustrates an Extension Archive Descriptor file (`ext.xml`) for a filter extension called Loop Back Filter.

Example 3–1 The Extension Archive Descriptor File for a Filter Extension

```
<?xml version="1.0"?>
<Extension>
<name>Loop Back Filter</name>
<version>1.0</version>
<className>oracle.edge.impl.filter.LoopBackFilter</className>
<type>Filter</type>
<Parameters>
        <Parameter name="TagID" defaultValue="" description="The Invalid Tag ID">
            <valueType type="string"/>
        </Parameter>
        <Parameter name="LightStackName" defaultValue="stack1" description="The
Light Stack Instance Name">
            <valueType type="string"/>
        </Parameter>
    </Parameters>
```

Example 3–2 describes a simplified version of the DTD for ext.xml; Table 3–8 describes this DTD's elements.

Example 3–2 The DTD for the Extension Archive Descriptor File

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT Extension (name, version, className, type, Parameters)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT version (#PCDATA)>
<!ELEMENT className (#PCDATA)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT Parameters (Parameter+)>
<!ELEMENT Parameter (valueType)>
<!ATTLIST Parameter
  name CDATA #REQUIRED
  displayName CDATA #IMPLIED
  defaultValue CDATA #IMPLIED
  description CDATA #IMPLIED
```

```
  encrypted (true|false)  #IMPLIED
  isClearText (true|false) #IMPLIED
  required  (true|false)  #IMPLIED>
<!ELEMENT valueType EMPTY>
<!ATTLIST valueType
 type (int | string | double | boolean) #REQUIRED>
```

Table 3–8 Elements and Attributes of the DTD for the Extension Archive Descriptor File

Element	Attribute or Text	Description
`Extension`		Defines the properties of an extension.
`name`	`#text`	The name of the extension.
`type`	`#text`	The type of the extension, such as a driver, filter, or dispatcher. Although the match is not case-sensitive, there must be no extra spaces or special characters in the text. The reserved values are: Device, Filter, Dispatcher.
`version`	`#text`	A text representation of the version number of the extension.
`className`	`#text`	The name of the class to load and instantiate the driver. This is the entry class that implements one of the standard extension interfaces. You must include a package name to form a fully qualified class name.
`Parameters`	(Parent of the <Parameter> element.)	The parameters that users can edit after an extension has been uploaded.
`Parameter`	Attributes include: ■ `name` ■ `displayName` ■ `defaultValue` ■ `encrypted` ■ `isClearText` ■ `required`	■ `name` -- The name of the parameter. ■ `displayName` -- The display name of the parameter. ■ `defaultValue` -- The default value for the parameter. ■ `encrypted` -- Indicates whether the value for the parameter should be encrypted so that the value does not have to be stored in clear-text format. ■ `isClearText` -- Enables the default value (and the value for the parameter instance) to be reset to clear-text format. If the encrypted parameter is set to `true`, then the clear-text format is read and then set to encrypted format the next time the Oracle Sensor Edge Server starts. ■ `required` -- Indicates whether the parameter value is required.
`valueType`	`type`	The type of the parameter (which can be one of the following): ■ `int` -- if the parameter is a 32-bit signed integer. ■ `string` -- for a string of variable length. ■ `double` -- for a double precision number. ■ `boolean` -- for a boolean value (true, false).

classes

This directory includes all of the classes files, native binaries, files, or static data. The classes files packaged into JAR files must be expanded on top of this directory. This release does not support loading JAR libraries.

lib

The Extension Archive file also includes the `lib` directory, where you specify third-party libraries. Example 3–3 illustrates an Extension Archive file for an Alien device driver, where the `lib` directory includes the library specific to the Alien device driver, `Gateway.jar`.

Example 3–3 Extension Archive File for an Alien Device Driver

```
meta-inf/ext.xml
meta-inf/Manifest.mf
classes/oracle/edge/impl/driver/AlienReader.class
lib/Gateway.jar
```

Packaging an Extension Archive File

To package an Extension Archive file:

1. Build a sandbox directory. Use this directory as the JAR source directory.
2. At the top of this directory, create the `META-INF` and `classes` directories.
3. Copy all class files and properties files (if any) to the classes directory. In the META-INF directory, create `ext.xml`, the Extension Archive Descriptor file.
4. Archive the files. You can use the JAR tool included in the JDK, or any other standard compression utility. Run the JAR tool from top-level directory of the sandbox. For example, executing `jar cvMf test.jar` archives the files in the sandbox directory into `test.jar`. You can then upload test.jar to the Oracle Sensor Edge Server. Do not archive the `META-INF` and `classes` directories as part of the sandbox directory. For example, the command `c:/work> jar tvf test.jar` displays the files in `test.jar` have been properly archived as follows:

   ```
   0 Thu Apr 08 14:36:56 PDT 2004 META-INF/
   71 Thu Apr 08 14:36:56 PDT 2004 META-INF/ext.xml
   0 Thu Apr 08 13:42:52 PDT 2004 classes/
   0 Thu Apr 08 13:42:52 PDT 2004 classes/my/
   0 Thu Apr 08 13:42:58 PDT 2004 classes/my/test.class
   ```

   Note: No slashes or other directory indicators appear before the `META-INF` and `classes` directories. Including the entire path in the JAR prevents the Oracle Sensor Edge Server from locating the Extension Archive Descriptor file or the classes. As a result, the extension cannot be deployed.

Uploading Extensions

To upload an extension:

1. Package the driver, filter, or dispatcher in an Extension Archive File as described in "Adding Extensions to the Oracle Sensor Edge Server Instance".
2. Copy this JAR file to:

   `ORACLE_HOME/j2ee/applications/edge/edge/extensions`

3. Restart the Oracle Sensor Edge Server by restarting the OC4J Instance. The extensions display in the tree as available drivers, filters, or dispatchers and can be configured as the Oracle Sensor Edge Server's devices, filter instances, or current dispatcher.

   Tip: If the `<IsExtensionMonitorEnabled>` element has been set to `true` in `edgeserver.xml`, then you only need to copy the JAR file to `ORACLE_HOME/edge/extensions`. The running Oracle Sensor Edge Server then picks up the extension automatically and does not need to be stopped and then restarted. Because this method of adding an extension slows performance, it is recommended only for development instances.

   In Oracle Application Server 10*g* (10.1.3), `edgeserver.xml` is deprecated and maintained only for backward compatibility.

Extension Class Hierarchy

All of the extensions of the Oracle Sensor Edge Server are arranged as:

`EdgeObject`—The basic root class, which contains a unique identifier.

`AbstractEdgeExtensionImpl`—Implements the EdgeExtension interface.

- `AbstractDispatcherV2`—The base class for all dispatchers.
  - `AbstractDispatcher`—Implements `AbstractDeviceV2` class to provide basic dispatcher behavior.
    - `SimpleDispatcher`—Enables you to build custom dispatchers.

`AbstractFilter`—The base class for all filters. It implements the `Filter` interface and defines the monitoring API for filters required to generate events that monitor filter performance.

- `SimpleFilter`—Enables you to write both device-level and device group-level filters.

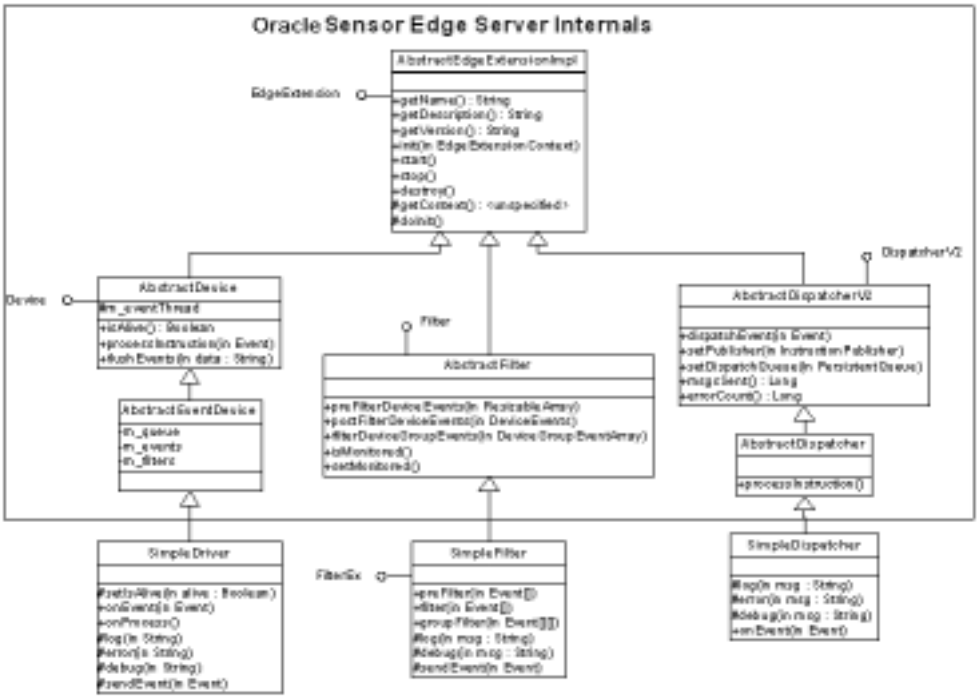
`AbstractDevice`—The base class for all devices. It provides its own thread for reading data from the device and processing the read data.

- `AbstractEventDevice`—Extended from `AbstractDevice` class; provides integration with the device-level filters and implements the required methods to propagate events to the event processor.
  - `SimpleDriver`—Provides the common functionality required by most custom drivers. Many of the drivers that ship with Oracle Sensor Edge Server are extended from `SimpleDriver`.

Note: Extend from the `SimpleDriver`, `SimpleFilter`, or `SimpleDispatcher` classes to create an extension.

Figure 3–26 illustrates the extension class hierarchy.

Figure 3–25 The Edge Extension Hierarchy



Implementing Extensions

The `doInit()` method implements extensions, as this call initializes the instance of an extension at runtime.

Extension Context

When the instance of an extension is created at runtime, the corresponding `Context` is created that enables the extension to:

- Set (or retrieve) the runtime `Context` data.
- Locate and communicate with other extensions of the Oracle Edge Sensor Server.
- Access the system facilities of the Oracle Edge Sensor Server.
- Retrieve information about the instance itself.

Retrieving Information About the Instance The base class, `EdgeExtension`, provides utility functions for an instance to retrieve information about itself. These methods include:

- `getContext()`

  Returns the runtime context.

- `getName()`

  Returns the name of the extension.

- `getDescription()`

  Returns the description of the extension.

- `getVersion()`

  Returns the version string of the extension.

Accessing the Runtime Context of an Instance To retrieve the instance's `Context` object, use

```
EdgeExtensionContext context = super.getContext();
```

The method call, `getContext()`, returns the `Context` object of the current instance.

Managing the Parameters of an Instance

An instance of an extension does not hold its own persistent data or configuration; configuration data is passed in at runtime when the instance is initialized. The configuration data is defined as parameters, which are composed of name / value pairs. Each parameter has a unique name and an optional value.

Note: This release of the Oracle Sensor Edge Server does not directly support trees or arrays of values. You are responsible for un-marshalling the data when forming non-scalar type data.

Exposing Custom Parameters

Extensions often have specific configurations. For example, a driver might include such configuration parameters as serial port name, baud rate, IP address, port number, login and password. These parameters must be defined to enable the driver to communicate with the device.

To expose parameters for a driver implementation, you must modify the Extension Archive Descriptor file. Example 3–4 illustrates a device that has two parameters that can be configured: serial port name and baud rate, defined within the `<Parameter>` extract tags.

Example 3–4 An Extension Archive Descriptor File with Exposed Parameters

```
<Extension>
         <name>My Driver</name>
         <type>Device</type>
         <className>my.testdriver</className>
         <Parameters>
                   <Parameter name="port" displayName="Serial Port">
                          <valueType type="string"/>
                   </Parameter>
                   <Parameter name="baud" displayName="Baud Rate">
                         <valueType type="int"/>
                   </Parameter>
         </Parameters>
</Extension>
```

Retrieving Parameter Values

Once you have defined the Extension Archive Descriptor file's `<Parameter>` tags, you can fetch the values for the parameters using the `EdgeExtensionConfigInfo` object. The values defined within the `<Parameter>` tags are retrieved using the `Context` object (illustrated in Example 3–5).

Example 3–5 Retrieving Parameter Values Using the Context Object

```
EdgeExtensionContext context = super.getContext();
ConfigParameter filenameParam = ct.getParameter( fileName );
```

The `getParameter()` method returns a `ConfigParameter` object. The `getParameter()` method returns the value for a parameter. (In Example 3–5, the `ConfigParameter` object is called `filenameParam` and the `getParameter()` method returns the value for a parameter called `fileName`.)The name of the target parameter must be passed to the `ConfigParameter` object. Further, the name of this parameter must match the name given to the name attribute of the `<Parameter>` element of the Extension Archive Descriptor file. Once you obtain the `ConfigParameter` object, you can get the value of the parameter (illustrated in Example 3–6).

Example 3–6 Retrieving the Value of a Parameter

```
m_fileName = filenameParam.getStringValue();
```

Note: The `getStringValue()` method returns the `string` value of the parameter. If the value for the parameter is an `int`, call the `getIntegerValue()` method, which returns an integer object.

4

Using the Sensor Data Repository

This chapter describes the Sensor Data Repository through the following sections:

- "Overview of the Sensor Data Repository"
- "Schema Reference"

Overview of the Sensor Data Repository

The Sensor Data Repository is a collection of database tables, views, and PL/SQL packages for storing and querying sensor event data. See also "Oracle Sensor Edge Server and Sensor Data Repository Considerations".

Relational Tables

The Sensor Data Repository's relational tables store the actual event observations and metadata. The relational views are based on these tables. Applications use these relational views and the programming interface.

Table 4–1 describes the relational tables in the Sensor Data Repository.

Table 4–1 Relational tables in the Sensor Data Repository

Table Name	Purpose
EDG_EVENT_TAB Table	Table storing the events from the middle ware and applications.
EDG_TAG_TAB Table	Cached copy of the tags that have been observed so far.
EDG_CAP_TAB table	Mapping table to define the kind of events that a device can send or receive.
EDG_CTXT_TAB Table	The context/containment table for defining relationship between contexts and containments.
EDG_CTXT_REL_TAB table	The table where the relationship between the contexts are defined.
EDG_EVENT_INFO_TAB Table	Table for storing information related to event such as Event Type and Subtypes.
EDG_DEVICE_TAB Table	Devices table.
EDG_DIAG_TAB Table	Table to store diagnostic information.
EDG_LOG Table	Log table for warning and internal error.

Relational Views in the Sensor Data Repository

The Sensor Data Repository's views (described in Table 4–2) are used for querying the stored data.

Table 4–2 Relational Views in the Sensor Data Repository

View Name	Description
EDG_CAP	View of the device's capabilities
EDG_CTXT	Read-only view of the context; the view can be changed using PL/SQL
EDG_CTXT_REL	Read-only view of the context relationship
EDG_CTXT_REL_NAME_VW	View of the context relationship with context names
EDG_DEVICE	Read-only view of the device table
EDG_DEV_CAP_VW	Device capability view
EDG_DEV_DIAG_VW	View of the device's diagnostic information
EDG_DEV_EVENT_VW	View showing the event captured by a device
EDG_DEV_LAST_DIAG_VW	View showing the last diagnostics information
EDG_DEV_LAST_OBSV_VW	View of the latest observation made by a device
EDG_DIAG	Read-only view of the diagnostics table
EDG_EVENT	Read-only view of the event table
EDG_EVENT_INFO	View of the currently valid event metadata
EDG_EVENT_VW	View of the event with type and subtype in place
EDG_TAG	Read-only view of the tags seen
EDG_TAG_LAST_DEV_VW	View of the last device that detected the tag
EDG_TAG_PATH_VW	View of the path taken by the tag in terms of devices that have detected it

Sensor Data Repository PL/SQL Package

The Sensor Data Repository defines a PL/SQL package which enables you to manipulate the contextual containment relationship and manually insert an event into the queue. Table 4–3 describes the procedures specified in the PL/SQL package. See also "PL/SQL Programming Interface".

Table 4–3 Procedures specified in the package

Procedure	Description
on_event	Procedure to handle the incoming event and sort out how to disassemble the parts and put them into various tables.
create_ctxt	Procedure to add a new context
update_ctxt_rel	Procedure to update the context hierarchy
update_ctxt	Procedure to update the context
remove_ctxt	Procedure to remove a context

Operations and Queries on the Repository

This section describes the following operations and queries of the Sensor Data Repository.

- Creating and Deleting Repositories
- Saving Observations to the Repository
- Querying the Archive

Creating and Deleting Repositories

The creation and deletion of a Sensor Data Repository is essentially the same as creating a Sensor Data Repository schema. The creation of a repository should be automatic if you opt to install it as part of the installation process. Otherwise, you can manually invoke a SQL script to install the repository. On the server side, there is a flag, called `isArchived`, that should be turned on to enable the server to start archiving data.

Saving Observations to the Repository

The Oracle Sensor Edge Server, if set to archiving mode, automatically sends events to Sensor Data Repository so that all of the events going to the Oracle Sensor Edge Server instance are archived. The application has the option of manually enqueuing events to the repository through the use of PL/SQL procedure in the `edg_sda` package (see "PL/SQL Programming Interface").

Querying the Archive

Use the views and relational tables described in Table 4–1 and Table 4–2 to query the Sensor Data Repository. See also "Schema Reference".

Schema Reference

This section lists the tables and views and PL/SQL programming interface of the Sensor Data Repository.

Tables

The Sensor Data Repository includes the following tables:

- EDG_CAP_TAB table
- EDG_CTXT_REL_TAB table
- EDG_CTXT_TAB Table
- EDG_DEVICE_TAB Table
- EDG_DIAG_TAB Table
- EDG_EVENT_INFO_TAB Table
- EDG_EVENT_TAB Table
- EDG_LOG Table
- EDG_TAG_TAB Table

EDG_CAP_TAB table

A mapping table that defies the types of events that a device can send or receive.

Table 4–4 EDG_CAP_TAB table

Name	Data Type	Nulls?	Default Value	Description
OBJECT_ID	NUMBER(10)	N	na	Primary key for this mapping entry
REF_DEVICE	NUMBER(10)	Y	na	Reference to the device interested
REF_EVENT_INFO	NUMBER(10)	Y	na	Reference to the event information interested
SEND_OR_RECV	VARCHAR2(16)	Y	'SEND'$	Flag to indicate whether the device can send or receive the event
CREATED_BY	VARCHAR2(256)	Y	USER$	The user who created this entry
CREATED_TIME	TIMESTAMP(6)	Y	SYSTIMESTAMP$	The time when the entry was created

EDG_CTXT_REL_TAB table

The table in which relationships between the contexts are defined.

Table 4–5 EDG_CTXT_REL_TAB table

Name	Data Type	Nulls?	Default Value	Description
OBJECT_ID	NUMBER(10)	N	na	Primary key for the relationship entry between the contexts
CID	NUMBER(10)	Y	na	The child context's id$
PID	NUMBER(10)	Y	na	The parent context's id$
CREATED_BY	VARCHAR2(256)	Y	USER	The user who created this entry$
CREATED_TIME	TIMESTAMP(6)	Y	TIMESTAMP	The time when the entry was created
RETIRED_BY	VARCHAR2(256)	Y	na	The user who voided this entry
RETIRED_TIME	TIMESTAMP(6)	Y	na	The time when the entry was voided
IS_CURRENT	VARCHAR2(1)	Y	na	Indicator as to whether the entry is current
REF_NEXT_VER	NUMBER(10)	Y	na	Reference to the next version of the relationship, added to help reconstruct the containment history for a context.

EDG_CTXT_TAB Table

The context/containment table for defining the relationship between contexts and containments.

Table 4–6 EDG_CTXT_TAB table

Name	Data Type	Nulls?	Default Value	Description
OBJECT_ID	NUMBER(10)	N	na	Primary key for the context entry
NAME	VARCHAR2(1024)	Y	na	The name for the context
DESCRIPTION	NUMBER(10)	Y	na	The description for the context
IS_DEFAULT	VARCHAR2(256)	Y	'F'$	Context is the universe flag
CREATED_BY	TIMESTAMP(6)	Y	USERS$	User who created the entry
CREATED_TIME	VARCHAR2(256)	Y	SYSTIMESTAMP$	Time entry was created
RETIRED_BY	TIMESTAMP(6)	Y	na	User who voided the entry
RETIRED_TIME	VARCHAR2(1)	Y	na	Time entry was voided
IS_CURRENT	NUMBER(10)	Y	'T'$	Current entry flag

EDG_DEVICE_TAB Table

The devices table.

Table 4–7 EDG_DEVICE_TAB table

Number	Name	Data Type	Nulls?	Default Value
1 Primary key for devices	OBJECT_ID	NUMBER(10)	N	na
2 Tag ID (EPC code) representing this device	TAG_ID	VARCHAR2(256)	Y	na
3 Name for this device object	NAME	VARCHAR2(256)	N	na
4 Description for this device	DESCRIPTION	VARCHAR21024)	Y	na
5 Site name where this device is located	SITE_NAME	VARCHAR2(256)	N	na
6 User who created this device	CREATED_BY	VARCHAR2(256)	Y	USER$
7 Time when this device entry was created	CREATED_TIME	TIMESTAMP(6)	Y	SYSTIMESTAMP$
8 Reference to help locate the last diagnostic status of the device	LAST_STATUS	NUMBER(10)	Y	na

EDG_DIAG_TAB Table

Table used to store diagnostic information.

Table 4–8 EDG_DIAG_TAB table

Number	Name	Data Type	Nulls?	Default Value
1 Primary key for diagnostic entry	OBJECT_ID	NUMBER(10)	N	na

Table 4–8 (Cont.) EDG_DIAG_TAB table

Number	Name	Data Type	Nulls?	Default Value
2 Reference to the device for this diagnostic entry	REF_DEVICE	NUMBER(10)	Y	na
3 Status for the device, server, or component	STATUS	VARCHAR2(64)	N	na
4 Message associated with an error or warning	MESSAGE	VARCHAR21024)	Y	na
5 Time when the erroneous event was encountered	TIME	TIMESTAMP(6)	Y	na
6 User who created this diagnostic entry	CREATED_BY	VARCHAR2(256)	Y	USER$
7 Time when this diagnostic entry was created	CREATED_TIME	TIMESTAMP(6)	Y	SYSTIMESTAMP$

EDG_EVENT_INFO_TAB Table

Table used to store event information (such as Type and Subtype).

Table 4–9 EDG_EVENT_INFO_TAB table

Number	Name	Data Type	Nulls?	Default Value
1 Primary key for event entry	OBJECT_ID	NUMBER(10)	N	na
2 Name for event entry	NAME	VARCHAR2(256)	N	na
3 Type for this event information entry	TYPE	NUMBER(5)	N	na
4 Subtype for this event information entry	SUBTYPE	NUMBER(5)	Y	-1$
5 Flag indicating if event is custom-defined or provided by, and registered with, Oracle	REGISTERED	VARCHAR2(1)	Y	'F'$
6 Description of the event	DESCRIPTION	VARCHAR2(1024)	Y	na
7 Usage pattern for the id field in the event; must be in sync with the driver	ID_USAGE	VARCHAR2(1024)	Y	na
8 Usage pattern for the data field in the event; must be in sync with the driver	DATA_USAGE	VARCHAR2(1024)	Y	na
User who created this entry	CREATED_BY	VARCHAR2(256)	Y	USER$
Time entry was created	CREATED_TIME	TIMESTAMP(6)	Y	SYSTIMESTAMP$
User who voided this entry	RETIRED_BY	VARCHAR2(256)	Y	na
Time this entry was voided	RETIRED_TIME	TIMESTAMP(6)	Y	na
Denotes if entry is valid	IS_CURRENT	VARCHAR2(1)	Y	'T'$

EDG_EVENT_TAB Table

Table used to store event information from middle ware and applications.

Table 4–10 EDG_EVENT_TAB table

Number	Name	Data Type	Nulls?	Default Value
1 Primary key for events	OBJECT_ID	NUMBER(10)	N	na
2 Tag ID (EPC code) for the tag	TAG_ID	VARCHAR2(256)	Y	na
3 Device from which the event originated	REF_DEVICE	NUMBER(10)	Y	na
4 Context relevant to the event; set by applications depending on context	REF_CTXT	NUMBER(10)	Y	-1$
5 Source name of the event	SOURCE_NAME	VARCHAR2(64)	Y	'F'$
6 Reference to event metadata	REF_EVENT_INFO	NUMBER(10)	Y	na
7 Reference used to correlate aggregated events	CORRELATION_ID	VARCHAR2(64)	Y	na
8 Data field for the event; depends on type of device. See ID_USAGE field in EDG_EVENTINFO for more information	DATA	VARCHAR2(1024)	Y	na
Time event occurred	TIME	TIMESTAMP(6)	Y	na

EDG_LOG Table

Table used to store warnings and internal errors.

Table 4–11 EDG_LOG table

Number	Name	Data Type	Nulls?	Default Value
1 Primary key for log entry	OBJECT_ID	NUMBER(10)	N	na
2 Log level, allowed values are: "N" for notify, "W" for warning, and "E" for error.	LOG_LEVEL	VARCHAR2(1)	Y	na
3 Message to be logged for the error / warning	MESSAGE	VARCHAR2(4000)	Y	na
4 Database user who created the log entry	CREATED_BY	VARCHAR2(64)	Y	USER$
5 Time entry was created inside the database	CREATED_TIME	TIMESTAMP(6)	Y	SYSTIMESTAMP$

EDG_TAG_TAB Table

Cached copy of the tags observed so far.

Table 4–12 EDG_TAG_TAB table

Number	Name	Data Type	Nulls?	Default Value
1 Primary key for tag entry	OBJECT_ID	NUMBER(10)	N	na

Table 4–12 (Cont.) EDG_TAG_TAB table

Number	Name	Data Type	Nulls?	Default Value
2 Tag ID (EPC code)	TAG_ID	VARCHAR2(256)	Y	na
3 Reference to the last observation of the tag	LAST_EVENT	NUMBER(10)	Y	na
4 User who created the entry	CREATED_BY	VARCHAR2(256)	Y	USER$
5 Time entry was created	CREATED_TIME	TIMESTAMP(6)	Y	SYSTIMESTAMP$

Views

The Sensor Data Repository includes the following views:

- EDG_CAP
- EDG_CTXT
- EDG_CTXT_REL
- EDG_CTXT_REL_NAME_VW
- EDG_DEVICE
- EDG_DEV_CAP_VW
- EDG_DEV_DIAG_VW
- EDG_DEV_EVENT_VW
- EDG_DEV_LAST_DIAG_VW
- EDG_DEV_LAST_OBSV_VW
- EDG_DIAG
- EDG_EVENT
- EDG_EVENT_INFO
- EDG_EVENT_VW
- EDG_TAG
- EDG_TAG_LAST_DEV_VW
- EDG_TAG_PATH_VW

EDG_CAP

View of the device's capabilities

Table 4–13 EDG_CAP view

View	Data Type	Nulls?
1 OBJECT_ID Primary key for device-event capability mappings	NUMBER(10)	N
2 REF_DEVICE Reference to the device of interest	NUMBER(10)	Y
3 REF_EVENT_INFO Reference to the event information of interest	NUMBER(10)	Y
4 SEND_OR_RECV Flag indicating is device can send / receive events	VARCHAR2(16)	Y

```
SELECT object_id, ref_device, ref_event_info, send_or_recv
FROM edg_cap_tab
```

EDG_CTXT

Read-only view of the context, the view can be changed using PL/SQL procedures.

Table 4–14 EDG_CTXT view

View	Data Type	Nulls?
1 OBJECT_ID Primary key for the context entry	NUMBER(10)	N
2 NAME Name for the context	VARCHAR2(1024)	Y
3 DESCRIPTION Description of the context	VARCHAR2(1024)	Y
4 SEND_OR_RECV Whether or not the context is the universe	VARCHAR2(16)	Y

```
SELECT object_id, name, description, is_default
FROM edg_ctxt_tab
WHERE is_current = 'T'
WITH READ ONLY
```

EDG_CTXT_REL

Read-only view of the context relationship.

Table 4–15 EDG_CTXT_REL view

View	Data Type	Nulls?
1 OBJECT_ID Primary key for the relationship entry between the contexts	NUMBER(10)	N
2 CID The Child context's ID	NUMBER(10)	Y
3 PID The Parent context's ID	NUMBER(10)	Y

```
SELECT object_id, cid, pid
FROM edg_ctxt_rel_tab
WHERE is_current = 'T'
WITH READ ONLY
```

EDG_CTXT_REL_NAME_VW

Read-only view of the context relationship.

Table 4–16 EDG_CTXT_REL_NAME_VW view

View	Data Type	Nulls?
1 CHILD_ID The Child context's ID	NUMBER(10)	Y
2 PARENT_ID The Parent context's ID	NUMBER(10)	Y
3 PARENT_NAME Name of parent context	VARCHAR2(1024)	Y

Table 4–16 (Cont.) EDG_CTXT_REL_NAME_VW view

View	Data Type	Nulls?
4 CHILD_NAME Name of child context	VARCHAR2(1024)	Y

```
SELECT
rel.cid child_id,
rel.pid parent_id,
cP.name parent_name,
cC.name child_name
FROM edg_ctxt cP, edg_ctxt cC, edg_ctxt_rel rel
WHERE cP.object_id = rel.pid
 AND cC.object_id = rel.cid
```

EDG_DEVICE

Read-only view of the device table.

Table 4–17 EDG_DEVICE view

Number	Data Type	Nulls?
1 OBJECT_ID Primary key for the device	NUMBER(10)	N
2 TAG_ID Tag ID (EPC code) representing the device	VARCHAR2(256)	Y
3 NAME Name for the device object	VARCHAR2(256)	N
4 DESCRIPTION Device description	VARCHAR2(1024)	Y
5 SITE_NAME Name of site where device is located	VARCHAR2(256)	N
6 CREATED_BY User who created device entry	VARCHAR2(256)	Y
7 CREATED_TIME Time when device entry was created	TIMESTAMP(6)	Y
8 LAST_STATUS Reference to help locate the last diagnostic status of the device	NUMBER(10)	Y

```
SELECT
"OBJECT_ID", "TAG_ID", "NAME", "DESCRIPTION", "SITE_NAME", "CREATED_BY", "CREATED
_TIME", "LAST_STATUS"
FROM edg_device_tab
WITH READ ONLY
```

EDG_DEV_CAP_VW

Device capability view.

Table 4–18 EDG_DEV_CAP_VW view

Number	Data Type	Nulls?
1 DEVICE_TAG_ID Tag ID (EPC code) representing the device	VARCHAR2(256)	Y

Table 4–18 (Cont.) EDG_DEV_CAP_VW view

Number	Data Type	Nulls?
2 DEVICE_NAME Name for the device object	VARCHAR2(256)	N
3 DEVICE_DESC Description of the device	VARCHAR2(64)	Y
4 SITE_NAME Name of site where device is located	VARCHAR2(1024)	N
5 EVENT_TYPE Event information entry type	VARCHAR2(256)	N
6 EVENT_SUBTYPE Event information subtype	NUMBER(5)	Y
7 EVENT_DESC Description for this event type	NUMBER(5)	Y
8 SEND_OR_RECV Flag indicating if device can send / receive	VARCHAR2(1024)	Y

```
SELECT
dev.tag_id device_tag_id,
dev.name device_name,
dev.description device_desc,
dev.site_name site_name,
ei.type event_type,
ei.subtype event_subtype,ei.description event_desc,
cap.send_or_recv send_or_recv
FROM edg_cap cap, edg_device dev. edg_event_info ei
WHERE cap.ref_device=dev.object_id
 AND cap.ref_event_info=ei.object_id
```

EDG_DEV_DIAG_VW

View showing device's diagnostic information.

Table 4–19 EDG_DEV_DIAG_VW view

View	Data Type	Nulls?
1 DEVICE_TAG_ID Tag ID (EPC code) representing the devices	VARCHAR2(256)	Y
2 STATUS Status of the device, server or component	VARCHAR2(64)	Y
3 MESSAGE Message related to error or warnings	VARCHAR2(1024)	Y
4 TIME Time when erroneous condition was encountered	TIMESTAMP(6)	Y
5 DEVICE_NAME Name for the device object	VARCHAR2(256)	N
6 DEVICE_DESC Description of the device	VARCHAR2(1024)	Y
7 SITE_NAME Site name where device resides	VARCHAR2(256)	N

```
SELECT
dev.tag_id device_tag_id,
```

```
diag.status status,
diag.message message,
diag.time time,
dev.name
device_name,
dev.description device_desc,
dev.site_name site_name
FROM edg_diag diag, edg_device dev
WHERE dev.object_id=diag.ref_device
```

EDG_DEV_EVENT_VW

View showing events captured by a device.

Table 4–20 EDG_DEV_EVENT_VW view

View	Data Type	Nulls?
1 DEVICE_NAME Name for the device object	VARCHAR2(256)	N
2 DEVICE_TAG_ID Tag ID (EPC code) representing the device	VARCHAR2(256)	Y
3 DEVICE_DESC Description for the device	VARCHAR2(1024)	Y
4 SITE_NAME Name of site where device is located	VARCHAR2(256)	N
5 EVENT_ID Reference to the event object	NUMBER(10)	N
6 EVENT_TAG_ID Tag ID (EPC code) representing the tag and device for this event	VARCHAR2(256)	Y
7 DEVICE_ID Reference to the device object	NUMBER(10)	Y
8. EVENT_DATA Data field for the event. Varies according to type of device. See id_usage field in edge_event_info for more information	VARCHAR2(1024)	Y
9 EVENT_TIME Time when event occurred	TIMESTAMP(6)	Y
10 EVENT_CORRELATION_ID Reference used to correlate aggregated events	VARCHAR2(64)	Y
11 EVENT_NAME Name of the kind of event	VARCHAR2(256)	N
12 EVENT_TYPE Type for the event information entry, such as 200 for RFID observations	NUMBER(5)	N
13 EVENT_SUBTYPE Subtype for the event information entry, such as 1 for IN_FIELD RFID observation	NUMBER(5)	Y
14 EVENT_DESC Description for kind of event	VARCHAR2(1024)	Y

```
SELECT
dev.name device_name,
dev.tag_id device_tag_id,
dev.description device_desc,
dev.site_name site_name
```

```
diag.status status,
ev.*
FROM edg_device dev, edg_event_vw ev
WHERE ev.device_id=dev_object_id
```

EDG_DEV_LAST_DIAG_VW

View showing the last diagnostics information.

Table 4–21 EDG_DEV_LAST_DIAG_VW view

View	Data Type	Nulls?
1 DEVICE_TAG_ID Tag ID (EPC code) representing the devices	VARCHAR2(256)	Y
2 STATUS Status of the device, server or component	VARCHAR2(64)	Y
3 MESSAGE Message related to error or warnings	VARCHAR2(1024)	Y
4 TIME Time when erroneous condition was encountered	TIMESTAMP(6)	Y
5 DEVICE_NAME Name for the device object	VARCHAR2(256)	N
6 DEVICE_DESC Description of the device	VARCHAR2(1024)	Y
7 SITE_NAME Site name where device resides	VARCHAR2(256)	N

```
SELECT
dev.tag_id device_tag_id,
diag.status status,
diag.message message,
diag.time time,
dev.name device_name,
dev.description device_desc,
dev.site_name site_name
FROM edg_diag diag, edg_device dev
WHERE dev.last_status = diag.object_id
```

EDG_DEV_LAST_OBSV_VW

View showing the last observation made by a device.

Table 4–22 EDG_DEV_LAST_OBSV_VW view

View	Data Type	Nulls?
1 DEVICE_NAME Name for the device object	VARCHAR2(256)	N
2 DEVICE_TAG_ID Tag ID (EPC code) representing the device	VARCHAR2(256)	Y
3 DEVICE_DESC Description of the device	VARCHAR2(1024)	Y
4 SITE_NAME Site name where device resides	VARCHAR2(256)	Y
5 EVENT_ID Reference to the event object	NUMBER(10)	N

Table 4–22 (Cont.) EDG_DEV_LAST_OBSV_VW view

View	Data Type	Nulls?
6 DEVICE_ID Tag ID (EPC code) representing the devices	VARCHAR2(256)	Y
7 EVENT_TAG_ID Reference to the device object	NUMBER(10)	Y
8 EVENT_DATA Data field for the event. Varies according to device. See id_ usage field in edg_event_info	VARCHAR2(1024)	Y
9 EVENT_TIME Time when event occurred	VARCHAR2(1024)	Y
10 EVENT_CORRELATION_ID Reference used to correlate aggregated events	VARCHAR2(64)	Y
11 EVENT_NAME Name of the kind of event	VARCHAR2(256)	Y
12 EVENT_TYPE Type for the event information entry, such as 200 for RFID observations	NUMBER(5)	Y
13 EVENT_SUBTYPE Subtype for the event information entry, such as 1 for IN_FIELD RFID observations	NUMBER(5)	Y
14 EVENT_DESC Description for the kind of event	VARCHAR2(1024)	Y

```
SELECT
device_name,
device_tag_id,
device_desc,
site_name,
event_id,
event_tag_id,
device_id,
event_data,
event_time,
event_correlation_id,
event_name,
event_type,
event_subtype,
event_desc
FROM (
```

EDG_DIAG

Read-only view of the diagnostics table

Table 4–23 EDG_DIAG view

View	Data Type	Nulls?
1 OBJECT_ID Primary key for the diagnostics entry	NUMBER(10)	N
2 REF_DEVICE Reference to the related device.	NUMBER(10)	Y
3 STATUS The status of the device, server or component	VARCHAR2(64)	Y

Table 4–23 (Cont.) EDG_DIAG view

View	Data Type	Nulls?
4 MESSAGE Message related to error or warnings	VARCHAR2(1024)	Y
5 TIME Time when erroneous condition was encountered	TIMESTAMP(6)	Y
6 CREATED_BY The user who created the diagnostic entry	VARCHAR2(256)	Y
7 CREATED_TIME Time when the diagnostic entry was created	TIMESTAMP(6)	Y

```
SELECT
"CONNECT_ID", "REF_DEVICE", "STATUS", "MESSAGE", "TIME", "CREATED_BY", "CREATED_
TIME"
FROM edg_diag_tab
WITH READ ONLY
```

EDG_EVENT

Read-only view of the events table.

Table 4–24 EDG_EVENT view

View	Data Type	Nulls?
1 OBJECT_ID Primary key for the event	NUMBER(10)	N
2 TAG_ID Tag ID (EPC code) representing the tag and device for which the event was created	VARCHAR2(256)	Y
3 REF_DEVICE The device from which the event originated	NUMBER(10)	Y
4 REF_CTXT Context relevant to the event; set by applications depending on the contextual situation	NUMBER(10)	Y
5 SOURCE_NAME Source name of the event	VARCHAR2(64)	Y
6 REF_EVENT_INFO Reference to the event metadata	NUMBER(10)	Y
7 CORRELATION_ID Reference used to correlate aggregated events	VARCHAR2(64)	Y
8 DATA Data field for the event; varies depending on device see id_ usage in edg_event_info for more information	VARCHAR2(1024)	Y
9 TIME Time when the event occurred	TIMESTAMP(6)	Y

```
SELECT
"OBJECT_ID", "TAG_ID", "REF_DEVICE", "REF_CTXT", "SOURCE_NAME", "REF_EVENT_INFO",
"CORRELATION_ID" "DATA", "TIME"
FROM edg_event_tab
WITH READ ONLY
```

EDG_EVENT_INFO

View of the currently valid event metadata.

Table 4–25 EDG_EVENT_INFO view

View	Data Type	Nulls?
1 OBJECT_ID Primary key for the event information entry	NUMBER(10)	N
2 NAME Name for the event information entry	VARCHAR2(256)	N
3 TYPE Type of event information entry	NUMBER(5)	N
4 SUBTYPE Subtype of event information entry	NUMBER(5)	N
5 REGISTERED Flag indicating if event information is custom-defined or provided by, and registered at, Oracle	VARCHAR2(1)	Y
6 DESCRIPTION Description for the kind of event	VARCHAR2(1024)	Y
7 ID_USAGE Usage pattern for the ID field in the event (from the middle ware side); must be in sync with the driver implementation	VARCHAR2(1024)	Y
8 DATA_USAGE Usage pattern for the data field in the event (from the middle ware side); must be in sync with the driver implementation	VARCHAR2(1024)	Y

```
SELECT object_id, name, type, subtype, registered, description, id_usage, data_
usage, data_usage
FROM edg_event_tab
WHERE is_current='T'
```

EDG_EVENT_VW

View of the event with Type and Subtype in place.

Table 4–26 EDG_EVENT_VW view

View	Data Type	Nulls?
1 EVENT_ID Primary key for the event	NUMBER(10)	N
2 EVENT_TAG_ID Tag ID (EPC code) representing the tag and device for which the event was created	VARCHAR2(256)	Y
3 DEVICE_ID Device from which the event originated	NUMBER(10)	Y
4 EVENT_DATA Data field for the event. Varies according to kind of device. See id_usage field in edg_event_info for more information	VARCHAR2(1024)	Y
5 EVENT_TIME Time when the event occurred	TIMESTAMP(6)	Y
6 CORRELATION_ID Reference used to correlate aggregated events	VARCHAR2(64)	Y

Table 4–26 (Cont.) EDG_EVENT_VW view

View	Data Type	Nulls?
7 EVENT_NAME Name of the kind of event	VARCHAR2(256)	N
8 EVENT_TYPE Type for the event information entry, such as 200 for RFID observations	NUMBER(5)	N
9 EVENT_SUBTYPE Subtype for the event information entry, such as 1 for IN_FIELD RFID observation event	NUMBER(5)	Y
10 EVENT_DESC Description for the kind of event	VARCHAR2(1024)	Y

```
SELECT
ev.object_id event_id,
ev.tag_id event_tag_id,
ev.ref_device device_id,
ev.data event_data,
ev.time event_time,
ev.correlation_id event correlation_id,
md.name event_name,
md.type event_type,
md.subtype event_subtype,
md.description event_desc
FROM edg_event_tab ev, edg_event_info md
WHERE ev.ref_event_info=md.object_id
```

EDG_TAG

Read-only view of the tags screen.

Table 4–27 EDG_TAG view

View	Data Type	Nulls?
1 OBJECT_ID Primary key for the tag entry	NUMBER(10)	N
2 TAG_ID Tag ID (EPC code).	VARCHAR2(256)	Y
3 LAST_EVENT Reference to the last observation of the tag	NUMBER(10)	Y

```
SELECT
object_id tag_id, last_event
FROM edg_tag_tab
WITH READ ONLY
```

EDG_TAG_LAST_DEV_VW

View of the last device that detected the tag.

Table 4–28 EDG_TAG_LAST_DEV_VW view

Number	Data Type	Nulls?
1 DEVICE_NAME Name for the device object	VARCHAR2(256)	N

Table 4–28 (Cont.) EDG_TAG_LAST_DEV_VW view

Number	Data Type	Nulls?
2 DEVICE_TAG_ID Tag ID (EPC code) representing the device	VARCHAR2(256)	Y
3 DEVICE_DESC Description for the device	VARCHAR2(1024)	Y
4 SITE_NAME Name of site where device is located	VARCHAR2(256)	N
5 EVENT_ID Reference to the event object	NUMBER(10)	N
6 EVENT_TAG_ID Tag ID (EPC code) representing the tag and device for this event	VARCHAR2(256)	Y
7 DEVICE_ID Reference to the device object	NUMBER(10)	Y
8. EVENT_DATA Data field for the event. Varies according to type of device. See id_usage field in edge_event_info for more information	VARCHAR2(1024)	Y
9 EVENT_TIME Time when event occurred	TIMESTAMP(6)	Y
10 EVENT_CORRELATION_ID Reference used to correlate aggregated events	VARCHAR2(64)	Y
11 EVENT_NAME Name of the kind of event	VARCHAR2(256)	N
12 EVENT_TYPE Type for the event information entry, such as 200 for RFID observations	NUMBER(5)	N
13 EVENT_SUBTYPE Subtype for the event information entry, such as 1 for IN_FIELD RFID observation	NUMBER(5)	Y
14 EVENT_DESC Description for kind of event	VARCHAR2(1024)	Y

```
SELECT
devEVENT.*
FROM edg_tag, edg_dev_event_vw devEVENT
WHERE tag.last_event-devEVENT.event_id
```

EDG_TAG_PATH_VW

View of the path taken by the tag in terms of the devices that have detected it.

Table 4–29 EDG_TAG_PATH_VW view

View	Data Type	Nulls?
1 DEVICE_NAME Name for the device object	VARCHAR2(256)	N
2 DEVICE_TAG_ID Tag ID (EPC code) representing the device	VARCHAR2(256)	Y
3 DEVICE_DESC Description for the device	VARCHAR2(1024)	Y
4 SITE_NAME Name of site where device is located	VARCHAR2(256)	N

Table 4–29 (Cont.) EDG_TAG_PATH_VW view

View	Data Type	Nulls?
5 EVENT_ID Reference to the event object	NUMBER(10)	N
6 EVENT_TAG_ID Tag ID (EPC code) representing the tag and device for this event	VARCHAR2(256)	Y
7 DEVICE_ID Reference to the device object	NUMBER(10)	Y
8. EVENT_DATA Data field for the event. Varies according to type of device. See id_usage field in edge_event_info for more information	VARCHAR2(1024)	Y
9 EVENT_TIME Time when event occurred	TIMESTAMP(6)	Y
10 EVENT_CORRELATION_ID Reference used to correlate aggregated events	VARCHAR2(64)	Y
11 EVENT_NAME Name of the kind of event	VARCHAR2(256)	Y
12 EVENT_TYPE Type for the event information entry, such as 200 for RFID observations	NUMBER(5)	Y
13 EVENT_SUBTYPE Subtype for the event information entry, such as 1 for IN_FIELD RFID observation	NUMBER(5)	Y
14 EVENT_DESC Description for kind of event	VARCHAR2(1024)	Y
15 NEXT_DEVICE_ID Reference to the next device object	NUMBER	Y
16 NEXT_DEVICE_NAME Name of the next device object	VARCHAR2(256)	Y
17 NEXT_DEVICE_TAG_ID Tag ID (EPC code) representing the next time	VARCHAR2(256)	Y
18 TIME_DIFF Time taken to travel from one device to the next	INTERVAL DAY() TO SECOND()	Y

```
SELECT
device_name
.device_tag_id
.device_desc
.site_name
.event_id
.event_tag_id
.device_id
.event_data
.event_time
.event_correlation_id
.event_name
.event_type
.event_subtype
.event_desc
.next_device_id
.next_device_name
.next_device_tag_id
.(event_end_time-event_time) time_diff
```

```
FROM (
 SELECT edv.*,
  DECODE( LEAD(edv.device_id) OVER (PARTITION BE edv.event_tag_id ORDER BY
edv.event_time), edv.device_id,
0, 1) hop,
 LEAD(edv.device_id) OVER (PARTITION BY edv.event_tag_id ORDER BY edv.event_time)
next_device_id,
LEAD(edv.device_name) OVER (PARTITION BY edv.event_tag_id ORDER BY edv.event_time)
next_device_id,
LEAD(edv.device_TAG_ID) OVER (PARTITION BY edv.event_tag_id ORDER BY edv.event_
time) next_device_tag_id,
LEAD(edv.event_time) OVER (PARTITION BY edv.event_tag_id ORDER BY edv.event_time)
event_end_time
FROM edg_dev_event_vw edv
 ) ev_path
WHERE hop=1
```

PL/SQL Programming Interface

The Sensor Data Repository includes the following PL/SQL package.

EDG_SDA Package

```
1: PACKAGE EDG_SDA 1S
2:
3: --- procedure to handle the incoming event
4: --- and sort out how to disassemble the parts
5: --- and put them into various tables.
6:
7:  PROCEDURE on_event
8: (p_correlation_id IN edg_event_tab.correlation_id%TYPE
9: .p_source_name IN edg_event_tab.source_name%TYPE
10. .p_site_name IN edg_device_tab.site_name%TYPE
11. .p_device_name IN edg_device_tab.name%TYPE
12: .p_type IN edg_event_info_tab.type%TYPE
13: .p_subtype IN edg_event_info_tab.type%TYPE
14: .p_time IN edg_event_tab.time%TYPE
15: .p_id IN edg_event_tab_tag.id%TYPE
16: .p_data IN edg_event_tab.data%TYPE
17: );
18:
19: --- procedure to add a new context
20: PROCEDURE create_ctxt
21: (p_object_id) OUT edg_ctxt_tab.object_id%TYPE
22: .p_name IN edg_ctxt_tab.name%TYPE
23: .p_description IN edg_ctxt_tab.description%TYPE
24: .p_parent_ctxt_tab.object_id%TYPE
25: );
26:
27: ---procedure to add a new context
28: PROCEDURE create_ctxt
29: (p_name IN edg_ctxt_tab.name%TYPE
30: .p_description IN edg_ctxt_tab.description%TYPE
31: .p_parent_ctxt_id IN edg_ctxt_tab.object_id%TYPE
32: );
33.
34: --- procedure to update the context hierarchy
35: PROCEDURE update_ctxt_rel
36: (p_cid IN edg_ctxt_rel_tab.cid%TYPE
37: .p_cid IN edg_ctxt_rel_tab.pid%TYPE
```

```
38: );
39:
40: --- procedure to update the context
41: PROCEDURE update_ctxt
42: (p_object_id IN edg_ctxt_tab.object_is%TYPE
43: .p_name IN edg_ctxt_tab.name%TYPE
44: .p_description IN edg_ctxt_tab.description%TYPE
45: );
46:
47: --- procedure to remove a context
48: PROCEDURE remove_ctxt
49: (p_object_id IN edg_ctxt_tab.object_id%TYPE
50:
51: END EDG_SDA;
```

5

Oracle Sensor Edge Mobile

This chapter, through the following sections, describes the Oracle Sensor Edge Mobile.

Overview of Oracle Sensor Edge Mobile

The mobile Oracle Sensor Edge Server (Sensor Edge Mobile) is a client-side (PocketPC only) application (Figure 5–1) configured with drivers that read barcode and read and write RFID tags.

Figure 5–1 Oracle Sensor Edge Mobile Administration Page on a Handheld Device

The architecture is similar to that of the Oracle Sensor Edge Server, in that it includes dispatchers that can communicate directly with applications. For example, Sensor Edge Mobile includes a dispatcher which talks with an ActiveX control that enables management pages written in HTML with JavaScript to control the drivers and display or submit the returned data.

Sensor Edge Mobile sits between the sensor device and the application. On one side, Sensor Edge Mobile interfaces with different types of sensors and devices and on the other end it feeds filtered data events to applications. The communication is bi-directional, with events passing from devices, through the platform, to the application, and instructions passing from the application, through the platform, to the hardware device driver.

The Sensor Edge Mobile service runs entirely on the handheld device, and can communicate with other applications or services that are external to it, or operate entirely offline, collecting data for later synchronization with an outside application. The flow of events goes from the collection device to the driver. The queue collects events until the dispatcher can process them. The dispatcher can communicate with the application through an ActiveX control by sending characters to the keyboard buffer, or through almost any other communication method. Only one dispatcher can be active at a time, but that one dispatcher may be communicating with multiple client devices and any number of drivers.

The Sensor Edge Mobile code components include the Driver Manager, Event Manager, and the Configuration Manager. Once these components are started, the main service calls on the Configuration Manager to read the configuration file. It then starts the configured dispatcher and passes in any configuration parameters that have been specified.

The Driver Manager is responsible for loading and managing the life cycle of the drivers. The Driver Manager calls on the Configuration Manager to determine which drivers need to be loaded, and what parameters to make available to them on

instantiation, and then loads and initializes them. The Driver Manager does not hold any thread internally, but an instance of it is held by the core instance as long as the platform is running.

There is only one Event Manager and one Driver Manager in the Sensor Edge Mobile. The Driver Manager may load any number of Drivers.

Connecting Sensor Edge Mobile to Applications

The Sensor Edge Mobile is designed to allow application programmers to have easy access to events coming from the devices, and to submit instructions to the device. The application may either communicate directly with a dispatcher, or it may communicate through another layer (such as an ActiveX control) to the dispatcher.

Applications interface with Sensor Edge Mobile through such means as:

- An ActiveX control, which then communicates with the Sensor Edge Mobile running as a service process in the background.
- A keyboard dispatcher, which brings focus to a specified application's window and then pushes characters into the keyboard buffer, just as if the user had typed them. For more information, see "Configuring the Keyboard Dispatcher".
- A custom dispatcher. Dispatchers are loaded dynamically, and interfaces are published so that any third party can develop a specialized dispatcher.

Configuring the Dispatchers and Drivers

Drivers and the current dispatcher retrieve their configuration information at the startup of the Oracle Sensor Edge Mobile service. This configuration is represented as parameters comprised of a name / value pairs.

The drivers and current dispatchers are configured in `EdgeMobileCofig.xml` (Example 5–1), which is located in the same directory as the Sensor Edge Mobile application. The file describes the single dispatcher used by the service in the `<CurrentDispatcher>` parameter as well as the driver (or drivers) that are loaded within the `<DeviceList>` element. These elements pass different parameter values. In Example 5–1, the Keyboard Dispatcher is set as the current dispatcher with the Intemec IP3 driver (named `IP3Driver` as the sole `<device>` element defined in this file) passing parameters as required.

Example 5–1 EdgeMobileConfig.xml

```
<EdgeMobileConfig>
   <CurrentDispatcher>
   <Name>Keyboard Dispatcher</Name>
   <Version>1.0</Version>
   <Description>Keyboard Dispatcher</Description>
   <Library>KeyboardDispatcher.dll</Library>
      <Parameters>
      <Param name="DestinationApplication" value="Telnet"/>
      <Param name="RFIDReadMacro" value="TelnetMultipleRFID"/>
      <Param name="BarcodeReadMacro" value="TelnetMultipleBarcode"/>
      <Param name="TelnetSingle" type="KeySequenceMacro">
         <tab/><tab/>RFID<tab/>
         <data name="TagID"/>
         <tab/>
         <data name="TagData"/>
         <cr/>
      </Param>
```

```
      <Param name="TelnetMultipleRFID" type="KeySequenceMacro">
         <tab/><tab/>RFID<tab/>
         <Repeat>
            <data name="tag_id"/>
            <tab/>
            <data name="data"/>
            <Separator>
               <tab/><tab/>
            </Separator>
         </Repeat>
         <Cr/>
      </Param>
      <Param name="TelnetMultipleBarcode" type="KeySequenceMacro">
         <tab/><tab/>Barcode<tab/>
      <Repeat>
         <data name="data"/>
         <Separator>
             <tab/><tab/>
         </Separator>
      </Repeat>
         <Cr/>
      </Param>
      </Parameters>
   </CurrentDispatcher>
   <DeviceList>
      <Device>
         <Name>IP3Driver</Name>
         <Version>1.1</Version>
         <Description>Intermec RFID IP3 driver</Description>
         <Library>IP3Driver.dll</Library>
         <Parameters>
         </Parameters>
      </Device>
   </DeviceList>
</EdgeMobileConfig>
```

Configuring the Keyboard Dispatcher

The Keyboard Dispatcher collects data and puts the data into the keyboard buffer and the keyboard codes into a macro. This macro simulates the user typing while running an application such as a Telnet session or Web browser. The Keyboard Dispatcher performs the following operations (in the following order):

1. Captures the event data from the device driver.
2. Brings focus to the window of a specified application.
3. Sends the data and control characters specified in the `KeySequenceMacro` language as characters typed into the application window.

By modifying `EdgeMobileConfig.xml`, you specify which dispatchers and drivers are loaded and pass configuration parameters to these extensions. For Sensor Edge Mobile to use the Keyboard Dispatcher, you must define the Keyboard Dispatcher as the configuration file's `<CurrentDispatcher>` element as described in Example 5–2.

Example 5–2 Configuring the Keyboard Dispatcher as the Current Dispatcher

```
<CurrentDispatcher>
        <Name>Keyboard Dispatcher</Name>
        <Version>1.0</Version>
        <Description>Keyboard Dispatcher</Description>
```

```
        <Library>KeyboardDispatcher.dll</Library>
        <Parameters>
               <Param name="DestinationApplication" value="Telnet"/>
               <Param name="RFIDReadMacro" value="TelnetMultipleRFID"/>
               <Param name="BarcodeReadMacro" value="TelnetMultipleBarcode"/>
               <Param name="TelnetSingle" type="KeySequenceMacro">
                      <tab/><tab/>RFID<tab/>
                      <data name="tag_id"/>
                      <tab/>
                      <data name="data"/>
                      <enter/>
               </Param>
               <Param name="TelnetMultipleRFID" type="KeySequenceMacro">
                      <tab/><tab/>RFID<tab/>
                      <Repeat>
                             <data name="tag_id"/>
                             <tab/>
                             <data name="data"/>
                             <Separator>
                                    <tab/><tab/>
                             </Separator>
                      </Repeat>
                       <enter/>
                </Param>
                <Param name="TelnetMultipleBarcode" type="KeySequenceMacro">
                       <tab/><tab/>Barcode<tab/>
                       <Repeat>
                              <data name="data"/>
                              <Separator>
                                    <tab/><tab/>
                              </Separator>
                       </Repeat>
                       <enter/>
                </Param>
        </Parameters>
</CurrentDispatcher>
```

Defining DestinationApplication Parameter

The `DestinationApplication` parameter defines the name of the application that should be given focus before the keyboard buffer is sent character data from the Sensor Edge Mobile service. The syntax of this parameter is as follows:

```
<Param name="DestinationApplication" value="Telnet"/>
```

The application name itself is specified (`"Telnet"` in Example 5–1), not the path to the application. On the top of the application window of most applications, this name can be seen after the document name and a dash (for example, "Doc1.doc - Microsoft Word" for the Microsoft Word application).

If more than one of these application windows are open, then the application window closest to the front (the first window in the "z-order") is brought to the front and acted upon.

Defining the RFIDReadMacro Parameter

The `RFIDReadMacro` parameter defines which of the key sequence macros in the configuration file should be used for observation data from RFID reads. This parameter allows the definition of many key sequence macros in the configuration file,

and one of those to be selected by putting the desired macro's name in this parameter. The syntax of this parameter is as follows:

```
<Param name = "RFIDReadMacro" value = "TelnetMultipleRFID"/>
```

Defining the BarcodeReadMacro Parameter

The `BarcodeReadMacro` parameter defines which of the key sequence macros in the configuration file should be used for observation data from barcode reads. This parameter allows the definition of many key sequence macros in the configuration file, and one of those to be selected by putting the desired macro's name in this parameter. The syntax of this parameter is as follows:

```
<Param name="BarcodeReadMacro" value="TelnetMultipleBarcode"/>
```

Defining the Key Sequence Macro Parameters

The Keyboard Dispatcher sends data collected by Sensor Edge Mobile to the receiving application as if it were typed by the user. To accomplish this for an existing application, you first position the focus of the browser or application window on a specific input control that is to receive the data. For instance, sending the *tab* character a certain number of times to position the focus on a particular input field. You may also then want to generate an *Enter* key press to cause the data to be submitted or otherwise acted upon. To specify the specific sequence of keystrokes that the application requires, you must define a key sequence macro (illustrated in Example 5–3).

Example 5–3 Defining a Key Sequence Macro

```
<Param name="TelnetSessionRFID" type="KeySequenceMacro">
        <tab/><tab/>RFID<sp/>Data<tab/>
        <data name="tag_id"/>
        <tab/>
        <data name="data"/>
        <enter/>
</Param>
```

In Example 5–2, the parameter is given a unique user-defined name (`"TelnetSessionRFID"`), and its attribute `type` must be specified as `"KeySequenceMacro"`. The body of the macro contains a sequence of control characters, a space specified (`<sp/>`, and one or more `<data>` tags to indicate where the data values should be sent, as if the user typed them at the keyboard.

For example, if the values coming from the Sensor Edge Mobile for the data value "tag id" is "123456789" and the value for `"data"` is "0A0B0C0D0E0F" from an RFID tag then the sequence sent to the application is:

```
\t\tRFID Data\t123456789\t0A0B0C0D0E0F\r
```

Where `\t` indicates a tab character in the sequence, and `\r` indicates a carriage-return (Enter) character. This should tab across the application window to the first input field, type `"RFID Data"` (the space character code <sp/> is required), then tab to the *TagID* field and enter it, and then perform the same routine with the *TagData* field. Finally, it will send a carriage-return (Enter), which may be used to click a button and submit or save the data.

Line breaks, tabs, and spaces in the XML are not sent to the application; you can tab and pretty-print the contents of the macro in the configuration file without getting

unwanted characters sent to the application. For instance, in Example 5–4, "Macro1" and "Macro2" send the same character sequence:

Example 5–4 Sending a Character Sequence

```
<Param name="Macro1" type="KeySequenceMacro">
        <data name="tag_id"/>
        <tab/>
        <data name="data"/>
        <enter/>
</Param>
<Param name="Macro2" type="KeySequenceMacro"><data name="tag_id"/><tab/><data
name="data"/><enter/></Param>
```

Creating a Key Sequence Macro

To build a key sequence macro, record the keystrokes required to get to the input fields of the destination application, and where the data fields are to be typed. Then write this sequence as a key sequence macro as illustrated in Example 5–4. Finally, set the "RFIDReadMacro" or "BarcodeReadMacro" parameter to the name of the new key sequence macro ("MyMacro" in Example 5–5).

Example 5–5 Setting the RFIDReadMacro to the Name of the New Key Sequence Macro

```
<Param name="RFIDReadMacro" value="MyMacro"/>
        <Param name="MyMacro" type="KeySequenceMacro">
        <data name="tag_id"/>
        <tab/>
        <data name="data"/>
        <enter/>
</Param>
```

Now events from an RFID read will generate the key sequence illustrated in Example 5–5, inserting the values indicated by the <data> elements.

Enabling the Key Sequence Macro to Handle Repeating Elements

The examples illustrated thus far only handle one set of "tag_id" and "data" data elements from an RFID tag. To make this macro work for a series of these data value pairs, you must define the concept of repetition. Specify a portion of the key sequence macro that is to be repeated, as well as what character sequence should be sent between repetitions to get the focus to the next row or field of the application using the <Repeat> and <Separator> macro elements.

Example 5–6 Defining <Repeat> and <Separator> Elements

```
<Param name="TelnetMultipleRFID" type="KeySequenceMacro">
        <tab/><tab/>RFID<tab/>
        <Repeat>
                <data name="tag_id"/>
                <tab/>
                <data name="data"/>
                <Separator>
                        <tab/><tab/>
                </Separator>
        </Repeat>
        <enter/>
</Param>
```

Example Example 5–6 illustrates the <Repeat> and <Separator> macro elements. These specify the section of the macro that is to repeat for each data set, and what sequence should be sent between data sets.

For three pairs of data observations, with"Tag_id" and "data" value pairs ("Tag1", "Data1"), ("Tag2", "Data2"), and ("Tag3", "Data3"). The sequence generated from the macro illustrated in Example 5–6 would be:\t\tRFID\tTag1\tData1\t\tTag2\tData2\t\tTag3\tData3\r

Breaking this sequence down, the portion before the <Repeat> sent once:

```
\t\tRFID\t
```

And the repeated portions <data name="tag_id"/><tab/><data name="data"/>:

```
Tag1\tData1  Tag2\tData2  Tag3\tData3
```

each separated by the <Separator> sequence:

```
\t\t        \t\t
```

And finally the carriage-return (Enter) sent once:

```
\r
```

While the <Separator> sequence separates each repeated sequence, it never follows the last repeated sequence. In this example, the "\t\t" separator sequence is not be sent after the last pair of data values (in this case "Tag3\tData3").

Key Macro Element Keys for Special Control Sequences To simulate the pressing of any of the keyboard keys that may be required by an application, special control sequences (described in Table 5–1) are provided for use in the keyboard sequence macros.

Table 5–1 Control Sequences for Keyboard Keys

Element	Key Sequence Sent
<space/>	Spacebar
<sp/>	Spacebar
<enter/>	Enter key
<cr/>	Enter key
<backspace/>	Backspace key
<bs/>	Backspace key
<tab/>	Tab key
<backtab/>	Shift-Tab key
<clear/>	Clear key
<esc/>	Escape key
<pageup/>	Page Up key
<pagedown/>	Page Down key
<end/>	End key
<home/>	Home key
<leftarrow/>	Left Arrow key

Table 5–1 (Cont.) Control Sequences for Keyboard Keys

Element	Key Sequence Sent
<uparrow/>	Up Arrow key
<rightarrow/>	Right Arrow key
<downarrow/>	Down Arrow key
<ins/>	Insert key
<del/>	Del key
<cancel/>	Control-break
<f1/>	F1 key
<f2/>	F2 key
<f3/>	F3 key
<f4/>	F4 key
<f5/>	F5 key
<f6/>	F6 key
<f7/>	F7 key
<f8/>	F8 key
<f9/>	F9 key
<f10/>	F10 key
<f11/>	F11 key
<f12/>	F12 key

As illustrated in Example 5–6, to send a sequence of literal characters, any spaces or tabs must be explicitly entered, as in <sp/> and <tab/>. Any other tabs and spaces in the macro will be ignored as pretty-print formatting of the XML. The other reason this is done is that the XML standard for parsing tabs and spaces does not preserve all of these "whitespace" characters between elements, and this modification would alter the macro processing in ways that would be difficult to predict.

Keyboard Macro Elements for Control Keys or Data Positioning The <control char="x"/> and <data name="tag_id"/> macro elements (described in Table 5–2) specify generated control keys or positions where data should be typed.

Table 5–2 Keyboard Macro Elements

Element	Description
<control char="x"/>	Control-char
<data name="tag_id"/>	Named data value

The special <control> element generates a control sequence just as if the user held the *Control* key and then typed the specified character. For instance, to generate Control-B, specify <control char="b"/>.

The <data> element fetches the current value of the data in the observation event, and inserts it at the current position. The specific names and their meaning (such as "tag_id" representing the Tag-ID of an RFID tag) are made available to the application as name and value pairs. Table 5–3 describes the pre-defined values for the <data> element.

Table 5–3 Pre-Defined Values for the <data> Element

Name	Description
`device_name`	The name of the device generating the observation event.
`device_desc`	A description of the device generating the observation event.
`timestamp`	The timestamp of the event creation in a textual presentation format.
`block_index`	The index of this event in a block (group) of events.
`block_count`	The number of events in this event block.
`tag_id`	RFID: The TagID of an RFID tag as a hexadecimal string. The data payload is in "data".
`data`	RFID: The data payload as a hexadecimal string. Barcode: The barcode read data as a hexadecimal string.
`type`	Barcode: The type of barcode ("1d" or "2d").
`checksum`	Barcode: The checksum value as a hexadecimal string.
`encoding_format`	Barcode: The encoding format.

The `<data>` element also acts with the `<Repeat>` element (described in "Enabling the Key Sequence Macro to Handle Repeating Elements" to iterate through a sequence of data values, if multiple events are returned by a read. See "Handling Observation Events" for more information on the “block_index” and “block_count” data values.

If no tags can be read in an RFID read, an event is returned with the `"tag_id"` and `"data"` values set to an empty string (`""`).

Checking Device Status

The Device Manager can "ping" devices to determine if they are alive and responsive. devices which do not respond are restarted without user intervention.

The Device Manager records the last time an event has been returned from the device and, if it has been a long time since the last event, issues a ping OS event to the driver. The driver interface contract requires that the device responds with a special response-from-ping event in a reasonable period of time. If the device does not respond in a configurable time, the Device Manager issues a shutdown command to the device. If after a period of time the device has not responded with a shutdown acknowledgement event, then Device Manager will unload the driver `.dll` and load and initialize a new device.

The ActiveX Application Interface

This revised ActiveX API has a number of major new features and differences from the previous beta release:

- The control has no visible rendering of its own, so the designer has the flexibility to process or present the observation data in any way.
- The data is returned to the JavaScript by registering a function to receive observation data events. The application can then set input fields with this data, or do whatever it pleases with it.

- The new API allows simultaneous access to any of the device classes supported by the currently configured drivers. For example, RFID and Barcode data could be read from one HTML page.

Object Declaration

Example 5–7 describes the object declaration of the Oracle Sensor Edge Mobile client control. In this example, the "event_handler" parameter registers the name of the JavaScript function that is to be called when an observation event is received. If no parameter is specified, a call to the default event handler function, "handle_event()" will be attempted. No error is returned if the function does not exist, and processing will proceed without an event handler.

Example 5–7 The Object Declaration of the Sensor Edge Mobile Client Control

```
<object id="SEMobileControl" name="SEMobileControl"
        classid="CLSID:049BE519-EE78-4AA5-8FC8-C5AE084CB26C"
        width="0"
        height="0">
        <param name="event_handler" value="handle_event"/>
<table><tr><td bgcolor="#FF0000">
<hr/>Unable to load Sensor Edge Mobile control.<hr/>
</td></tr></table>
</object>
```

The Activex API includes the following methods:

- rfid_read()
- rfid_write()
- rfid_kill()
- barcode_read()
- set_trigger_rfid_read()
- set_trigger_barcode_read()
- process_instruction()
- is_supported()

rfid_read()

The rfid_read() method will issue a read for all tags in range. The method signature is:

```
var retVal = SEMobileControl.rfid_read( );
```

The tags read will be returned by zero or more callbacks to the registered event handler method. If the read succeeds, the return value will be zero. If it fails for any reason, a non-zero value will be returned.

rfid_write()

The rfid_write() method issues a write for all of the tags in range. The method signature is:

```
var retVal = SEMobileControl.rfid_write( srcForm.srcData.value,
                                         srcForm.srcPasscode.value );
```

In this example, the parameters are hidden variables in the form `srcForm`. The `srcData` variable represents the data to write, represented as a hexadecimal string, and the `srcPasscode` contains the pass code for a write. If the write succeeds, the return value will be zero. If it fails for any reason, a non-zero value will be returned.

rfid_kill()

The `rfid_kill()` method issues a kill for all tags in range. The method signature is:

```
var retVal = SEMobileControl.rfid_kill( srcForm.srcTagId.value,
                                        srcForm.srcPasscode.value );
```

The `srcTagId` variable represents the TagID of the tag that is to be killed (represented as a hexadecimal string) and the `srcPasscode` contains the passcode for a kill. If the kill succeeds, the return value will be zero. If it fails for any reason, a non-zero value will be returned.

barcode_read()

The `barcode_read()` method will issue a read of the barcode pointed to by the device. The method signature is:

```
var retVal = SEMobileControl.barcode_read( );
```

The barcode read observation event will be returned by a callback to the registered event handler method. If the read succeeds, the return value will be zero. If it fails for any reason, a non-zero value will be returned.

set_trigger_rfid_read()

The `set_trigger_rfid_read()` method will tell Sensor Edge Mobile that when the user pulls the trigger on the device, a read of RFID data should be done. The method signature is:

```
var retVal = SEMobileControl.set_trigger_rfid_read();
```

The call is required to register the RFID read operation to the trigger pull. If the call succeeds, the return value will be zero. If it fails, there is no RFID device driver loaded, and a non-zero value will be returned.

set_trigger_barcode_read()

The `set_trigger_barcode_read()` method tells Sensor Edge Mobile that when the user pulls the trigger on the device, a read of barcode data should be done. The method signature is:

```
var retVal = SEMobileControl.set_trigger_barcode_read();
```

The call is required to register the barcode read operation to the trigger pull. If the call succeeds, the return value will be zero. If it fails, there is no barcode device driver loaded, and a non-zero value will be returned.

process_instruction()

The process_instruction method is a lower-level interface for the processing of instructions by drivers. It is not necessary for the currently defined RFID and barcode operations, but is made available for easy expansion of the instruction repertoire when a custom driver is used. The method signature is

```
var retVal=SEMobileControl.process_instruction(instruction,
data);
```

The "instruction" is an integer value (expressed as a string) from a list of defined constants. In EdgeMobilehtml_samples is a file named SEMobileCore.js. This file's values can be included in an HTML document by using a script element as follows:

```
<script type="text/javascript" src="SEMMobileCore.js"></script>
```

Then the instruction to start an RFID read using these constants would look like:

```
var retVal=SEMobileControl.rfid_read();
```

This instruction performs the same operation as the following:

```
var retVal=SEMobileControl.rfid_read();
```

Possible values for the commands are:

- Symbol Description

  RFID_READ START an RFID read. The argument value is ignored. RFID_WRITE DESCRIPTION string for device generating event. RFID_KILL Timestamp of event creation in textual presentation format. BARCODE_READ Index of this event in a block (group) of events. REGISTER_TRIGGER0 Number of events in this event block. TRIGGER_PULL0 RFID: TagID of RFID tag as a hexadecimal string. Data payload is in"data".

  The instruction to start a barcode read would look like:

  ```
  var retVal = SEMobileControl.process_instruction(BARCODE_
  DEAD. "");
  ```

  The equivalent method call is:

  ```
  var retval = SEMobileControl.barcode_read();
  ```

  The instruction to do an RFID write using these constants would look like:

  ```
  var retVal = SEMobileControl.process_instruction(RFID_WRITE,
  "0102030405060708,123");
  ```

  The hexidecimal value to write is separated from the passcode by a comma. The instruction is equivalent to the following:

  ```
  var retVal = SEMobileControl.rfid_write("0102030405060708",
  "123");
  ```

  An RFID kill would look like:

  ```
  var retval = SEMobileControl.process_instruction(RFID_KILL,
  "0102030405060708,123");
  ```

  As with the RFID write, the hexidecimal value to write is separated from the passcode by a comma. The instruction is equivalent to the following:

  ```
  var retval = SEMobileControl.rfid_kill("0102030405060708",
  "123);
  ```

  Finally, the instruction to register the trigger pull to a given read operation would look like:

  ```
  var retval = SEMobileControl.process_instruction( REGISTER_
  TRIGGER0, RFID_READ);
  ```

  The instruction performs exactly the same operation as the following:

  ```
  var retval = SEMobileControl.set_trigger_rfid_read();
  ```

is_supported()

The `is_supported` method has a single argument of an instruction code such as RFID_WRITE and returns true or false depending on whether or not there is a configured driver that can process this instruction. The method signature is:

if (SEMobileControl.is_supported(RFID_KILL) { / /RFID_KILL instruction supported by a configured driver...}

The "instruction" is an integer value (expressed as a string) from a list of defined constants in the `SEMobileCore.js` file.

Handling Observation Events

The registered JavaScript event handler function is called by the ActiveX control when an observation event is received. In the event handler method, the event object is referenced by the name of the ActiveX control instance. In Example 5–8 and Example 5–9 `"SEMobileControl"` is used as the control name.

The event data is available from the ActiveX object until the JavaScript returns from the event handler function. The data that is present in all events is available as properties, and the other data that is specific to a given event type is available by lookup. Table 5–4 lists the event data.

Table 5–4 Available Event Data

Name	Description
`device_name`	The name of the device generating the observation event.
`device_desc`	A description of the device generating the observation event.
`is_rfid_read`	Returns "true" id the vent is an RFID read event.
`is_barcode_read`	Returns "true" if the event is a barcode read event.
`timestamp`	The timestamp of event creation.
`block_index`	The index of this event in a block (group) of events.
`block_count`	The number of events in this event block.
`type`	The type code for the event object.
`subtype`	The subtype code for the event type.

Example 5–8 describes an event handler function that reports on the event by using properties on the event.

Example 5–8 Event Handler Function for Event Reporting

```
function handle_event()
{
    if ( SEMobileControl.is_barcode_read )  {
    alert ("Barcode data read from " + SEMobileControl.device_name );
    }
    if ( SEMobileControl.is_rfid_read )  {
      alert ("RFID data read from " + SEMobileControl.device_name );

}
```

All of these properties are only available until the event handler function has returned. As soon as it returns, the Sensor Edge Management control can again call the handler with another event, and any data associated with the last event is no longer available.

When RFID tags are read, the events returned are marked as part of a block (with a read of one tag being represented as a block of one event). To reflect this, all events have a block index and a block count associated with them. If the event is a singleton, the block count and index are set to "1", while if the block contains three events, the block index and count values for the three events in the block will look like (1, 3), (2, 3), and (3, 3). The "block_index" and "block_count" properties allow access to these values.

There may be other data specific to the class of device that is associated with the event. These are represented as name/value pairs, and can be looked up with the following `data()` method. The `data()` property looks up a data value specific to an event from a specific device class. Different data names are defined for each device class, and that if the data value is not present in the current event, an empty string ("") is returned. Table 5–5 describes the defined data names and their associated device class.

Table 5–5 Data Names

Name	Description
tag_id	RFID: The TagID of an RFID tag as a hexadecimal string. The data payload is in "data".
data	RFID: The data payload as a hexadecimal string. Barcode: The barcode read data as a hexadecimal string.
type	Barcode: The type of barcode ("1d" or "2d").
checksum	Barcode: The checksum value as a hexadecimal string.
encoding_format	Barcode: The encoding format.

Example 5–9 describes how the event handler first determines the device class (RFID or barcode) and then references `data()` to fetch the device class-specific data value.

Example 5–9 Fetching the Device Class-Specific Data Value

```
function handle_event()
{
   if ( SEMobileControl.is_barcode_read )  {
   alert ("Barcode data is " + SEMobileControl.data("data") );
   }
   if ( SEMobileControl.is_rfid_read )  {
       /* Note that "tag_id" is an RFID-specific event value */
       alert ("RFID tag ID is " + SEMobileControl.data("tag_id") );
     }
}
```

Deprecated Activex Application Interface

The current Sensor Edge Mobile ActiveX control can be accessed as follows:

- As a visible control on an HTML page containing the observation data collected by the driver.
- Through the control's API, where the control is not visible on the page, but provides an API for JavaScript.
- As a mixture of a visible control and the APIs in use.

This API only allows access to one device class (that is, either barcode or RFID) from a given HTML page.

Note: This API is deprecated in Oracle Sensor Edge Server Version 10.1.3.

Managing Sensor Edge Mobile

The Administration HTML page (Figure 5–1) enables you to manage Oracle Sensor Edge Mobile. Using this page, you can perform the following tasks:

- Extension Configuration and Management

  For drivers, the HTML page lists the configured drivers and their state (initializing, active, shutting down, or not responding). Release 10.1.3 of Oracle Sensor Edge Server Mobile 10.1.3 supports the Symbol 9000-G and Intermec IP3 with Color 700 PocketPC attached.

  For Symbol 9000-G, Sensor Edge Server Mobile supports the following operations:

  – rfid_read()
  – rfid_write()
  – rfid_kill()
  – barcode_read()

  For the Intermec IP3, Sensor Edge Mobile supports the following:

  – rfid_read()
  – barcode_read()

  For dispatchers, the page displays the configured driver and its current state (initializing, active, or shutting down).

- Monitoring Running Status

  The page provides the service status of Sensor Edge Mobile (initializing, active, or shutting down).

- Service Management

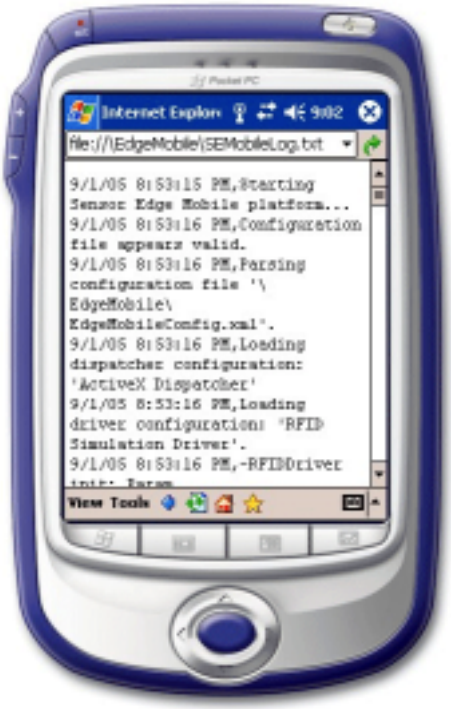
  The page provides a Shutdown Service button, which you use after you have configured a driver, changed a dispatcher, or edited the XML configuration file (`EdgeMobileConfig.xml`).

Figure 5–2 Shutting Down the Sensor Edge Mobile Service

- Logging

  The pages displays updates to the log entries in real time.

Figure 5–3 Viewing the System Logs



- Performance Monitoring

  For each loaded driver, the page displays counts of observation data events and the timestamp of the last event from each driver

- Configuration

  The page enables you to read the configuration XML file (`EdgeMobileConfig.xml`) and change the configuration of the file as needed.

Internationalization

For internationalization, all of the string literals in the Sensor Edge Mobile are in a `.dll` file which enables the shipping of international executables.

6

Configuring Devices, Filter Instances, and Dispatchers

This chapter includes the following sections:

Overview of Device, Filter Instance and Dispatcher Configuration

This chapter describes the extensions (drivers, filters, and dispatchers) that the Oracle Sensor Edge Server supports out of the box and how to configure their parameters when you create instances of these objects (or, in the case of the dispatcher, when you set the current dispatcher for an Oracle Sensor Edge Server instance). Because these extensions are static objects, you must create instances of them to enable an Oracle Sensor Edge Server instance to use them to process event data. Fore more information, see "Setting the Dispatcher for the Oracle Sensor Edge Server Instance" and "Setting the Devices and Filters Used by the Oracle Sensor Edge Server".

Setting the URI Parameters for Devices and Dispatchers

Some of the device and current dispatcher configuration requires you to define a *URI* parameter. This parameter, which requires a `String` value, tells the **Transport Library** which transport and parameters to use. Enter the value for the URI parameter in the following format:

esc://<transport>?<param1>=<value1>&<param2>=<value2>&...

Where:

- *<transport>* is the name of the transport to use (as in *transportstransport@name* in the **transports.xml** file).
- *<param1>* is a parameter name for the transport.
- *<value>* is the value for the parameter.

You can specify as many parameters as you need. For example, to connect using TCP/IP to a server called *test.oracle.com* at port 9999, enter the value for the *URI* parameter as:

esc://tcp&hostname=test.oracle.com&port=9999

To connect to a serial port *COM1* at 9600 baud, enter the value for the *URI* parameter as:

```
esc://com?port=COM1&baud=9600
```

To connect to TCP port 9999 at IP 1.2.3.4, enter the value for the *URI* parameter as:

esc:/ /tcp?hostName=1.2.3.4&port=9999

To write output to a file and read from another file (which is useful for testing and logging), enter the value for the *URI* parameter as:

```
esc://file?inputFileName=/tmp/myInFile.txt&outputFileName=/tmp/myOutFile.txt
```

The values defined for the URI parameter are specific to the transport type. Table 6–1 describes each supported transport type and its associated parameter values.

Table 6–1 Supported Transport Types

Name	Description	Parameter Values
com	Serial Communication Port	*port* is the name of the port (such as *COM1*, *COM2*, *tty1S*). *baud* is the baud rate (such as *9600*, *38400*, *115200*). Example: *esc://com?port=COM1&baud=9600*
tcp	TCP / IP Connection	*hostname* is the IP address or hostname to connect to (for example, 127.0.0.1). *port* is the port number. *timeout* is an optional parameter that sets the number (in seconds) in which to wait for a valid connection. Example: *esc://tcp?hostName=127.0.0.1&port=9999*
stdout	Use standard input and output streams.	N / A
file	Files used for input and output.	*outFileName* is an optional parameter naming the output file. If you do not enter a value, then the output is discarded. *inFileName* is an optional parameter naming the input file. If you do not enter a value, then there is no input. *purgeOnClose* is an optional parameter requiring a `boolean` value. Setting this parameter to *true* purges the input file when the connection is closed. Use this parameter for polling data. Example: *esc://file?inputFileName=/tmp/myInFile.txt&outputFileName=/tmp/myOutFile.txt*
http	Post to a URL through HTTP.	*URL* is the URL to post to. *proxyServer* is an optional parameter specifying which proxy server to use. *proxyPort* is an optional parameter specifying the port for the proxy.
ftp	Post and retrieve files through FTP.	*hostname* is the name of the FTP server *user* is the name of the user. *password* is the password for the user.

Tip: You can also define commonly used parameters in the **transports.xml** file. All of the values defined in the *URI* parameter override those defined in the `transports.xml` file.

Configuring Devices

The Oracle Sensor Edge Server provides drivers that support RFID readers (described in Table 6–3).

Table 6–2 RFID Readers

Driver Name	Supported Devices
Alien Reader Driver	All of the Alien Technology RFID readers, such as the Alien NannoScanner (Model 915MHz, the ePC Reader). See "Configuring Alien Reader Driver-Based Devices".
BarcodeDriver	Any barcode reader that works with standard ASCII mode through serial or network communication. See "Configuring BarcodeDriver-Based Devices".
Intermec BRI Driver	All of the Intermec RFID readers that support BRI (Basic Reader Interface), such as the IF5 Fixed RFID Reader. See "Configuring Intermec BRI Driver-Supported Devices".
Intermec Reader Driver	Devices include: ■ Intermec IDK (Model ITRF91501) ■ Intermec PCMCIA Reader (Model ITR100) See "Configuring Intermec Reader Driver-Based Devices".
Matrics Driver	Supported fixed Readers by Symbol Technologies include: ■ Matrics AR400 Reader ■ Matrics SR400 Reader ■ Matrics XR400 Reader See "Configuring Matrics Driver-Based Devices".
Samsys Driver	SAMSys EPC UHF Long-Range Reader (Model MP9320 2.7) See "Configuring Samsys Driver-Based Device".
Tyco Reader	Tyco Sensormatic Agile2 Reader (Powered by ThingMagic) See "Configuring Tyco Reader Driver-Based Devices"

In addition, Oracle Sensor Edge Server provides drivers that support printer devices and those for display and notification (described in Table 6–3).

Table 6–3 Printer Drivers and Display and Notification Drivers

Driver Name	Supported Devices
AnimationDriver	This is a software-only driver used for device simulation. See "Configuring an Instance of the AnimationDriver".
ConsoleDriver	This is a software-only driver used for device simulation. See "Configuring an Instance of the ConsoleDriver".
Edge Simulator Driver	This is a software-only driver used for device simulation. See "Configuring Edge Simulator Driver-Based Devices".

Table 6–3 (Cont.) Printer Drivers and Display and Notification Drivers

Driver Name	Supported Devices
LpmlDriver	Supported printer devices include: ■ Zebra Technologies R110Xi printer / encoder ■ Intermec EasyCoder PM4i Printer ■ Sato Barcode Printer CL408e (which supports LPML) ■ Loftware Print Server See "Configuring LpmlDriver-Based Devices".
PatliteDriver	PHE-3FB PC-Controlled Light by Patlite (USA) Corp. The Lightstack Device Controllers are available for download from Oracle Technology Network (http: / / www.oracle.com / technology /) See "Configuring PatliteDriver-Based Devices".
Prolite Driver	Pro-Lite TruColorII LED Display (Model PL-M2014RV6) See "Configuring Prolite Driver-Based Devices".

The Oracle Sensor Edge Server also ships with Edge Echo Driver. The Edge Echo Driver does not control any physical device; instead, this driver receives an instruction event, copies the event, and then sends it back out. This event has all of the same attributes as the original instruction event, depending on the configuration of the Edge Echo Driver instance. The Oracle Sensor Edge Server processes the event as any other event generated by a device: the event is filtered and then dispatched by the current dispatcher. See also "Configuring Edge Echo Driver-Based Instances".

Configuring Alien Reader Driver-Based Devices

The Alien Reader Driver supports all of the Alien Technology RFID readers.

To configure an Alien Reader Driver driver-based device, define the following parameters:

- *IPAddress* -- The hostname or IP address of the machine running the Device Controller. If it runs on the same machine as the Oracle Sensor Edge Server, enter *127.0.0.1*.
- *Port Number* -- The port number used to communicate with the device (23 is the default).
- *username* and *password*
- `AntennaSeqIDList` -- The list of identifiers for each antenna.
- `AntennaMappedDeviceNameList` -- The list of mapped device names associated with each antenna.

Observation Events Generated by the Alien Reader Driver

Observation events are events that are generated by the driver and are dispatched out to the system. Table 6–4 lists the events generated by the Alien Reader Driver.

Table 6–4 Observation Events Generated by the Alien Reader Driver

Type	Subtype	TagID	Data	Description
200	Various	Tag ID	The data field (payload) stored in a tag.	General tag observation event.

Table 6–4 (Cont.) Observation Events Generated by the Alien Reader Driver

Type	Subtype	TagID	Data	Description
1	0	Null	"alive or dead"	Each event using data property to indicate whether the device is alive or not.
1	0	Null	"dis"	An event indicating a successful device startup operation.
1	2	Null	"dif"	An event indicating a failed device startup operation.
1	0	Null	"dss"	An event indicating an successful device stop operation.
1	2	Null	"dsf"	An event indicating a failed device stop operation.
1	2	Null	"drf"	An event indicating a failed event-receiving operation.
1	0 or 2	Null	<callResults> <callResult name="write-tag"> <code>0/1</code> <message/> </callResults>	An event indicating a successful or failed tag-writing operation.

The Instruction Event Accepted by the Alien Reader Driver

Applications send devices instruction events to tell them to perform certain tasks. Table 6–5 lists the instruction event accepted by the Alien Reader Driver.

Table 6–5 Instruction Event Sent by the Alien Reader Driver

Type	Subtype	TagID	Data	Description
101	0	Null	<methodCalls> <methodCall name="write-tag""> <params> <param name='tagid'>%TAGID%</param> <param name='tagdata'>%TAGDATA%</param> </params> </methodCall> </methodCalls>	Write data to a tag identified by the specified tag ID.

Configuring an Instance of the AnimationDriver

The AnimatorDriver does not support a device; this driver supports interactive software simulations of devices. The configured instance of this driver accepts user input or events from the Oracle Sensor Edge Server and runs animation sequences defined in the configuration file (Example 6–1) on the screen.

To configure an AnimationDriver instance, define the *fileName* parameter as the full or relative path to the configuration file, an `.xml` file in the format described in

Example 6–1. Using this file, you can configure the AnimationDriver instance to accept and generate any type of event.

Example 6–1 Configuration File for the AnimationDriver

```
<animate>
  <window>... </window>
  <task>...</task>
</animate>
```

To configure this file, define its <window> and <task> elements.

Defining the <window> Element

The <window> element enables you to define a window. Each window appears as a top level window in the server's display. Within the <window> element, you can define any number of user interface controls in the window that trigger tasks.

The <window> element is comprised of three attributes which define the window and its dimensions: name, width, and height.

- name is a unique name for the window, which appears on the window's title bar.
- width is the width of the window, in pixels.
- height is the height of the window, in pixels.

Once you define the window, you can then define the controls within in it. Example 6–2 illustrates a window with a single push button control. The AnimationDriver supports button, a control type for a push button.

Example 6–2 Defining Windows in the Configuration File

```
<window name='myWin' width='200' height='100'>
  <control type='button' name='Pass Thru'/>
</window>
```

You define the tasks triggered when the users clicks the button using the <task> element.

Defining the <task> Element

A task is a set of actions triggered by either a control or an event. Example 6–3 illustrates the <task> element, with its <when> and <action> child elements. The <when> element tells the task when to run the <action> element. There can be multiple tasks that map to overlapping conditions.

Example 6–3 Defining Tasks in a Configuration File

```
<task name='TaskA'>
  <when> ... </when>
  <action> ... </action>
</task>
```

You define the <when> element using the following conditions:

- <onClick> matches a user clicking a control in a window. This condition takes one argument, the name of the control for the event. For example, <onClick name='Pass Thru'/> is matched if the user clicks the control button named *Pass Thru* (illustrated in Example 6–2).

- `<onEvent>` is fired if an event is received by the driver instance that matches the criteria defined for this event. You can define the following criteria for the event.
  - `type`, which matches the type of the event.
  - `subtype`, which matches the subtype of the event.
  - `id`, which looks for an exact match of the ID.
  - `ids`, which looks for any event with any ID that matches from the list. The value of the attribute should be a comma separated list of IDs it's looking for.
  - `data`, to look for a match in the data field of the event.
- `<onInterval>` is fired on a periodic basis. You define this element by specifying the `frequency` attribute. Set this attribute using a `String` value or using a number (`int`) to define the repeat interval to fire `<onInterval>` in seconds. For example, setting this attribute using a `String` value once sets `<onInterval>` to fire once upon startup.

Once you define when the task starts, you can then define what the task performs in the `<action>` element. Within the `<action>` element, you can define the following:

- `<audio>` which plays an audio file.
- `<img>` which shows a picture at a certain location.
- `<delay>` to delay between frames.
- `<send>` to send an event out to the system.

Configuring BarcodeDriver-Based Devices

To configure a BarcodeDriver -based device, define the *uri* parameter as the connection to the lightstack. See "Setting the URI Parameters for Devices and Dispatchers".

RFID Observation Event Returned by the BarcodeDriver

Table 6–6 describes the RFID observation event returned by the BarcodeDriver.

Table 6–6 RFID Observation Event Returned by the BarcodeDriver

Type	Subtype	Direction	Title	Description
200	1	From device (inbound)	ID Observed	A new barcode is read and decoded. The barcode is decoded and stored in the tag ID field of the event when a new barcode is observed.

Configuring an Instance of the ConsoleDriver

The ConsoleDriver is a simulation driver that displays all of the events it receives to a graphical window, which includes the following views:

- *Event List* -- Displays a list of all of the events received by a device since startup.
- *Details* -- If you double-click an event displayed in the *Event List* view, the details specific to that event display in this window.
- *Send* -- Select this view to send an event to the system. Before sending an event, ensure that the *TYPE*, *SUBTYPE*, and *ID* fields have data (that is, they are not empty). The *TIME* field displays current time stamp.

This driver is agnostic to event types and data. It only displays the events that it has received and sends any user input from the *Send* dialog. The user interface enables you

to input event data and send it out for dispatching as if it had been generated by an actual device. Use this driver to monitor and debug the event flow in the system.

Configuring Edge Echo Driver-Based Instances

The Edge Echo Driver does not control any physical device; instead, this driver receives an instruction event, copies the event, and then sends it back out. This event has the same attributes as the original instruction event except for the *Timestamp* attribute, which you can configure to display the current time. The Oracle Sensor Edge Server processes the event generated by an instance of the Edge Echo Driver the same as any other event generated by a device: the event is filtered and then dispatched by the current dispatcher. See also "Viewing Event Data".

To configure an instance of the Edge Echo Driver driver, define the following parameters:

- *stamptime* -- Enter *true* to set the *Timestamp* attribute of the event generated by the Edge Echo Driver instance to the current time.
- *logFileName* -- Enter the location of the log file. This is a `String` value.

Configuring Edge Simulator Driver-Based Devices

The Edge Simulator Driver generates events to simulate a real device. In general, you use the EdgeSimulator driver to test configurations and deployment designs; however, you can also use it for internal functional testing to see how events are processed throughout the system. The Edge Simulator Driver acts the same as any driver, except that instead of connecting to a physical device to read events, it takes parameters from an input file (such as Example 6–6) as instructions on when to generate fake events. This begins as soon as the device starts (which starts when the Oracle Sensor Edge Server starts).

Configuring an Edge Simulator Driver-based device requires that you define the device's *FILENAME* parameter by entering the name of this input file, which instructs the Edge Simulator Driver-based device how to generate fake events using the following instructions.

`<EventList>`

The `<EventList>` element defines a loop. This element is also the main block that groups all of the other instructions together. `<EventList>` has one attribute, repeat, which must be present to control looping. The value for repeat must be a decimal number from 0 to `LONG_MAX`. To generate events only once, set the repeat attribute to 1. Setting repeat to n results in all instructions looping n times. Setting repeat to 0 disables the block and causes the parser to skip it.

Example 6–4 illustrates the syntax for generating two events, pausing, generating two more events, and then looping 20 times:

Example 6–4 Defining a Loop

```
<EventList repeat='20'>
<Event> … </Event>
<Event> … </Event>
<EventInterval>…</ EventInterval>
<Event> … </Event>
<Event> … </Event>
</EventList>
```

You can include any number of instructions inside the <EventList> element. The order in which they are defined is the order in which they are executed.

<EventInterval>

The <EventInterval> element instructs the Simulator to pause for a certain period of time before proceeding. This is usually used to throttle the data rate. A decimal number defines the time period, in milliseconds, to wait for before executing the next instruction. Section 6–5 illustrates how to instruct the Simulator to wait for half a second between each event and three seconds between loops:

Example 6–5 The <EventInterval> Element

```
<EventList repeat='20'>
   <Event> … </Event>
   <EventInterval>500</ EventInterval>
   <Event> … </Event>
   <EventInterval>500</ EventInterval>
   <Event> … </Event>
   <EventInterval>3000</ EventInterval>
</EventList>
```

<Event>

The <Event> element tells the Simulator to send an event. The child elements (described in Table 6–7) control the event's fields.

Table 6–7 Event Elements for the Simulator

Event Field	Value
<type>	The number value that corresponds to the type of event.
<subtype>	The number value for the subtype. For example, the subtype value in Example 6–6 corresponds with a General Instruction Event, which is an event sent by application or a device to tell a specific device to perform an operation. In Example 6–6, the value of 1 turns on the device.
<id>	The text value of this field identifies a tag (that is, a read or target) to an event instruction. In Example 6–6, one of the <id> values for a tag is 03ffff045679.
<data>	The tag data. This is an optional field.
<deviceName>	The name of the device or application that generates the event. The <deviceName> enables the Simulator to appear as if it is another device when generating events.

Example 6–6 illustrates an input file which includes two groups of events: the first one runs only once and the second runs 20 times.

Example 6–6 Simulator Input File

```
<EdgeEventSimulation>
      <EventList repeat='1'>
         <Event>
            <type>100</type>
            <subtype>1</subtype>
            <id>03ffff045679</id>
            <data>No Data</data>
            <deviceName>My Device</deviceName>
```

```
        </Event>
   <EventInterval>500</ EventInterval>
      <Event>
         <type>100</type>
            <subtype>1</subtype>
            <id>03ffff045680</id>
            <data>No Data</data>
            <deviceName>My Device</deviceName>
     </Event>
          <EventInterval>3000</ EventInterval>
   </EventList>

   <EventList repeat='20'>
      <Event>
         <type>100</type>
         <subtype>1</subtype>
         <id>04ffff045679</id>
         <data>No Data</data>
         <deviceName>My Device</deviceName>
    </Event>
         <EventInterval>500</ EventInterval>
     <Event>
         <type>100</type>
          <subtype>1</subtype>
          <id>04ffff045680</id>
          <data>No Data</data>
          <deviceName>My Device</deviceName>
     </Event>
   </EventList>
</EdgeEventSimulation>
```

Although the format of the Event Type is fixed, you can extend the Event Type by mapping its fields to different meanings depending on the type of event.

Configuring an HtmlDriver Instance

The HtmlDriver is a display driver that enables you to display a window on either the Oracle Sensor Edge Server screen or to connect to remote servers to display HTML content. Use the HtmlDriver for applications that push data to display complex status or messages on graphical displays.

The HtmlDriver includes the Display Server (which is included in `htmlDriver.jar`), which enables display windows on remote servers. The Display Server runs as a Java process that waits for a network connection from the HtmlDriver instance. You can set up any number of Display Servers on a network. You must note the IP or hostname of the machines running the Display Server processes. More than one Display Server can run on the same machine provided that each Display Server is assigned to a different port.

Note: JDK 1.3 or higher must be installed on the machine and `java.exe` is in the path.

To install the display server on a machine:

1. Copy `htmlDriver.jar`.

2. Extract the files to a local directory. For example, create a directory called `displayServer` and run `jar xf htmlDriver.jar`.
3. Navigate to the `classes` directory (`cd classes`).
4. Run the DisplayServer application using

   ```
   java oracle.edge.impl.driver.HtmlDriver <port> [title] [x]
   [y] [width] [height] [showHistory]
   ```

   Where:

   - `<port>` specifies the TCP / IP port on which the Display Server listens. The port cannot already be in use. This is a required parameter.
   - `[title]` is an optional text string that names the window.
   - `[x]` is the display window's *x* position on the screen.
   - `[y]` is the display windows's *y* position on the screen.
   - `[width]` is the width of the window. If set to 0, then the window uses the screen width.
   - `[height]` is the height of the window. If set to 0, then the window uses the screen height.
   - `[showHistory] --` If set to *true*, then a small list displays at the top of the window showing all of the previously shown documents. Using this window enables users to manually scroll back to the last displayed document.

   For example, to run the Display Server on your desktop using port 8999 with a full-screen window, run:

   ```
   java oracle.edge.impl.driver.HtmlDriver 8999 "My Window" 0 0 0 0 false
   ```

   Note: This command contains quote strings that have spaces. Refer to the shell documentation.

To configure an HtmlDriver instance to enable display on remote servers, define the following parameters:

- Define the *serverlist* parameter by entering a comma-separated list of Display Servers that this driver instance can contact. The `String` value for this parameter is in the following format:

  <hostname or IP>:<port>,...

  To provide security, you must list each Display Server.
- Enter the TCP / IP port on which the Display Server listens. This port cannot already be in use on the machine.

If you do not define the *serverList* parameter, then the instance displays on a local window at the Oracle Sensor Edge Server's display. Running the instance on the local window any does not require Display Servers.

To display the instance on the local window:

- Define the display window by first entering a name for the window in the *title* field and then enter a `boolean` (true or false) in the *ShowHistory* field to display a window of viewed pages.
- Set the dimensions of the window by entering `int` values the following parameters:

- *x* -- The *x* position of the window on the screen, in pixels.
- *y* -- The *y* position of the window on the screen, in pixels.
- Enter the width and height of the window, in pixels. Entering 0 sets the window to the screen's height and width.

Events Supported by the HtmlDriver

Table 6–8 describes the events supported by the HtmlDriver.

Table 6–8 Events Supported by the HrmlDriver

Type	Subtype	Direction	Title	Description
210	1	From application (outbound)	Display	Display the HTML document embedded in the DATA field of the event. The ID is a number, starting from 0, that defines which server to send it to. This is the index of server as defined in the *serverList* parameter in the driver instance.
210	2	From application (outbound)	Print LPML Directly	Display the HTML document referenced by the URL provided in the DATA field of the event. The ID is a number, starting from 0, that defines which server to send it to. This is the index of server as defined in the *serverList* parameter in the driver.

Configuring Intermec BRI Driver-Supported Devices

The Intermec BRI Driver supports all of the RFID readers by Intermec that support BRI (Basic Reader Interface) and communicate with the Oracle Edge Sensor Server either though a serial or network (TCP/IP) connection. The Oracle Edge Sensor Server has been tested against the IF5 reader.

For more information, see http://www.intermec.com

Configuring an Intermec BRI Driver-based device, you must define the parameters described in Table 6–9.

Table 6–9 Parameter Values for an Intermec BRI Driver Based Device

Parameter Name	Value Type	Description
URI	String	The connection identifier that indicates the connection type and connection parameters as described in "Setting the URI Parameters for Devices and Dispatchers".
Status Query Response Timeout	Int	The timeout for waiting response from status query. The value set for the timeout must be lower than that set for the *Status Query Interval* parameter.
Status Query Interval	Int	The time, in milliseconds, between two consecutive status inquiries.
AntennaIds	String	The identifier(s) for connected antennae. Use a comma (,) to separate multiple identifiers for multiple antennae.

Table 6–9 (Cont.) Parameter Values for an Intermec BRI Driver Based Device

Parameter Name	Value Type	Description
DataFieldLength	Int	The size of the data field to be retrieved in terms of the number of bytes. 0 means no data fields need to be retrieved.
DataFieldValueType	String	Specify one of the three value types for the data field: ■ Int (integer) ■ HexString (hexadecimal string) ■ String (ASCII string) This parameter is effective only if the *DataFieldLength* parameter is larger than 0.
TagType	N / A	Specifies the types of tags to read: Options include: ■ ISO6B / G1 ■ ISO6B / G2 ■ ISO6C ■ EPC / Class / Gen1 ■ EPC / Class1 / Gen2 ■ Phillips / V1.19 ■ Mixed

Observation Events Generated by the Intermec BRI Driver

Observation events are events that are generated by the driver and are dispatched out to the system. Table 6–4 lists the events generated by the Intermec BRI Driver.

Table 6–10 Observation Events Generated by the Intermec BRI Driver

Type	Subtype	TagID	Data	Description
200	Various	Tag ID	The data field (payload) stored in a tag.	General tag observation event.
1	0	Null	"alive or dead"	An event using a data property to indicate whether the device is alive or not.
1	0	Null	"dis"	An event indicating a successful device startup operation.
1	2	Null	"dif"	An event indicating a failed device startup operation.
1	0	Null	"dss"	An event indicating an successful device stop operation.
1	2	Null	"dsf"	An event indicating a failed device stop operation.
1	2	Null	"drf"	An event indicating a failed event-receiving operation.

Table 6–10 (Cont.) Observation Events Generated by the Intermec BRI Driver

Type	Subtype	TagID	Data	Description
1	0 or 2	Null	<callResults> <callResult name="write-tag"> <code>0/1</code> <message/> </callResults>	An event indicating a successful or failed tag-writing operation.

The Instruction Event Accepted by the Intermec BRI Driver

Applications send devices instruction events to tell them to perform certain tasks. Table 6–5 describes the instruction event accepted by the Intermec BRI Driver.

Table 6–11 Instruction Event Sent by the Intermec BRI Driver

Type	Subtype	TagID	Data	Description
101	0	Null	<methodCalls> <methodCall name="write-tag""> <params> <param name='tagid'>%TAGID%</param> <param name='tagdata'>%TAGDATA%</param> Name='tagdataVauleType'> (String/HexString)</param> </params> </methodCall> </methodCalls>	Write data to a tag identified by the specified tag ID.

Configuring Intermec Reader Driver-Based Devices

For more information, see http://www.intermec.com

Requirements

Creating an Intermec Reader Driver-based device requires the following components, which are bundled and shipped with the Intermec Reader driver:

- IntelliTag IDK

  The IntelliTag IDK (the IDK) is a set of Intermec-supported software libraries and tools. This library, which is the only supported method of communicating with Intermec devices, is supported only on the Windows 32 platform (that is, Windows 2000 and Windows XP). The IntelligTag IDK is available at

  http://www.oracle.com/technology/products/iaswe/edge_server/extensions.html

- Intermec Reader Driver

Configuring an Intermec Reader Driver-based device requires that you define the following parameters:

- Set *IPAddress* to the hostname or IP address. If it runs on the same machine as the Edge Server, enter *127.0.0.1*.
- Set *PortNo* to the port number. The default is *6666*.

- Set the `AntennaSeqIDList` to the list of identifiers for each antenna.
- Set `AntennaMappedDeviceNameList` to the list of mapped device names associated with each antenna.

Configuring LpmlDriver-Based Devices

LPML (Label Printing Markup Language) is an XML-based printer language defined by Oracle Corp. which is supported by label printers and printer server vendors. The LpmlDriver supports any printer or print server systems that support the LPML language and binding.

To configure a device based on the LpmlDriver, you must define the parameters of the device as follows:

- *URI* -- The string used by the transport layer to connect to the printer. See "Setting the URI Parameters for Devices and Dispatchers".
- Selecting **Response Mode** sets the device to expect that the printer its communicating with supports LPML response messages. As a result, the device waits for responses from the printer for each print job.
- Selecting **Batch Mode** instructs the printer to send out as many jobs as possible in a single connection session before waiting for responses. This option enables the device to use a windowing algorithm instead of a half duplex communication mode with the printer.
- If **Batch Mode** is selected, set the *ReadTimeOut* parameter to the amount of time (in milliseconds) to wait before considering that the printer failed to respond.
- *maxRetryCount* -- The number of times to retry a printing job. This is an `int` value.
- *PerJobDelay* -- The interval (in seconds) to delay between each print job. This is an `int` value.

The Printer Response Observation Event Generated by the LpmlDriver

Table 6–12 lists the Printer Response observation event generated by the LpmlDriver, which are dispatched to the system.

Table 6–12 Printer Response Observation Event Generated by the LpmlDriver

Type	Subtype	Tag ID	Data	Description
210	2	Not used.	LPML response message	The response message from either the printer (if **Response Mode** is selected), or from the driver itself to tell the sender the result of the print job. The *correlationId* and *sourceName* fields are taken from the original print job request.

The Instruction Event Accepted by the LpmlDriver

Instructions are events sent to a driver by an application the command the driver to perform a certain task. Table 6–13 describes the instruction event accepted by the LpmlDriver.

Table 6–13 Instruction Event Accepted by the LpmlDriver

Type	Subtype	Tag ID	Data	Description
102	1	Not used.	LPML message	Print the LPML message specified in the *Data* field.

Configuring Matrics Driver-Based Devices

The Matrics Driver supports the Matrics Readers made by Symbol Technologies that communicate with the Oracle Sensor Edge Server using a serial or network (TCP/IP) connection.

Configuring a Matrics Driver-based device requires that you define the following parameters:

- *URI* -- The connection type and connection identifiers. See "Setting the URI Parameters for Devices and Dispatchers".
- *Status Query Interval* -- The time (in milliseconds) between to consecutive status queries. This is an `int` value.
- *Status Query Response Timeout* -- The time (in milliseconds) for the timeout for the waiting response from the status query. This `int` value must be less than the value set for the *Status Query Interval* parameter.
- *AntennaIds* -- The connected antennae. Use a comma (,) to separate entries.
- *Node Address* -- The node address for the reader connected under RS485. The address must match the value in the Matrics administration console.

Observation Events Generated by the Matrics Driver

Observation events are events that are generated by the driver and are dispatched out to the system. Table 6–4 lists the events generated by the Matrics Driver.

Table 6–14 Observation Events Generated by the Matrics Driver

Type	Subtype	TagID	Data	Description
200	Various	Tag ID	The data field (payload) stored in a tag.	General tag observation event.
1	0	Null	"alive or dead"	An event using a data property to indicate whether the device is alive or not.
1	0	Null	"dis"	An event indicating a successful device startup operation.
1	2	Null	"dif"	An event indicating a failed device startup operation.
1	0	Null	"dss"	An event indicating an successful device stop operation.
1	2	Null	"dsf"	An event indicating a failed device stop operation.
1	2	Null	"drf"	An event indicating a failed event-receiving operation.

Table 6–14 (Cont.) Observation Events Generated by the Matrics Driver

Type	Subtype	TagID	Data	Description
1	0 or 2	Null	<callResults> <callResult name="write-tag"> <code>0/1</code> <message/> </callResults>	An event indicating a successful or failed tag-writing operation.

Instruction Event Accepted by the Matrics Driver

Applications send devices instruction events to tell them to perform certain tasks. Table 6–5 lists the instruction events accepted by the Matrics Driver and their usage.

Table 6–15 Instruction Event Sent by the Matrics Driver

Type	Subtype	TagID	Data	Description
101	0	Null	<methodCalls> <methodCall name="write-tag""> <params> <param name='tagid'>%TAGID%</param> <param name='tagdata'>%TAGDATA%</param> </params> </methodCall> </methodCalls>	Write data to a tag identified by the specified tag ID.

Configuring PatliteDriver-Based Devices

The Patlite series of lightstacks supported by the PatliteDriver do not generate events, but instead act as indicator lights and signals. Sending events to Patlite lightstacks and trees turns on lights or causes them to blink for certain intervals.

To configure a device based on the PatliteDriver, define the *URI* parameter. See "Setting the URI Parameters for Devices and Dispatchers".

Configuring Prolite Driver-Based Devices

To configure a device based on the Prolite Driver, define the *URI* parameter by entering the connection URI. See "Setting the URI Parameters for Devices and Dispatchers".

Configuring Samsys Driver-Based Device

The Samsys Driver supports the SAMSys MP320 2.7 EPC reader that communicates with the Oracle Sensor Edge Server either through a serial or network (TCP/IP) connection. The Samsys MP9320 2.7 EPC reader, which supports CHUMP (Cookie-handling UDP Message Protocol), has been tested against the Oracle Edge Sensor Server.

Configure a Samsys Reader Driver-based device by defining the following parameters:

- *Connection Identifier* -- The connection type and connection identifiers. See "Setting the URI Parameters for Devices and Dispatchers".

- *Status Query Interval* -- Enter the time (in milliseconds) between to consecutive status queries. This is an `int` value.
- *Status Query Response Timeout* -- The amount of time of the timeout for the waiting response from the status query. This `int` value must be less than the value set for the *Status Query Interval* parameter.
- *Tag Scan Mode* -- Specify either autonomous or polling. Selecting autonomous mode sets the MP9320 2.7 EPC reader to notify the device whenever it scans tags. Selecting polling mode sets the MP9320 2.7 EPC reader to scan tags only if the driver sends out the `get-tag-list` command.
- *Tag Type* -- Specify the type of tag for the device to read, such as EPC1, EPC0, IS18gA, IS186B, STG.
- *AntennaIds* -- The connected antennae. Use a comma (,) to separate entries.

Observation Events Generated by the Samsys Driver

Observation events are events that are generated by the driver and are dispatched out to the system. Table 6–4 lists the events generated by the Samsys Driver.

Table 6–16 Observation Events Generated by the Samsys Driver

Type	Subtype	TagID	Data	Description
200	Various	Tag ID	The data field (payload) stored in a tag.	General tag observation event.
1	0	Null	"alive or dead"	Each event using data property to indicate whether the device is alive or not.
1	0	Null	"dis"	An event indicating a successful device startup operation.
1	2	Null	"dif"	An event indicating a failed device startup operation.
1	0	Null	"dss"	An event indicating an successful device stop operation.
1	2	Null	"dsf"	An event indicating a failed device stop operation.
1	2	Null	"drf"	An event indicating a failed event-receiving operation.
1	0 or 2	Null	<callResults> <callResult> <callResult name="write-tag"> <code>0/1</code> <message/> </callResult> </callResults>	An event indicating a successful or failed tag-writing operation.

Instruction Event Accepted by the Samsys Driver

Applications send devices instruction events to tell them to perform certain tasks. Table 6–5 lists the instruction event accepted by the Samsys Driver.

Table 6–17 Instruction Event Accepted by the Samsys Driver

Type	Subtype	TagID	Data	Description
101	0	Null	<methodCalls> <methodCall name="write-tag""> <params> <param name='tagid'>%TAGID%</param> <param name='tagdata'>%TAGDATA%</param> </params> </methodCall> </methodCalls>	Write data to a tag identified by the specified tag ID.

Configuring a Simple Audio Driver Instance

The SimpleAudioDriver accepts instruction events and enables applications to use the Oracle Sensor Edge Server machine's sound card to play any `.wav` file. For security reasons, only files in a specific directory (defined in the *audioFiles* parameter) are played.

> **Note:** The driver uses the `sun.audio` package that ships with any standard JRE or JDK. The package only requires special configuration for special setups, such as multiple sounds cards in one machine. For such cases, refer to the JDK documentation from Sun Microsystems, Inc.

To configure a Simple Audio Driver instance, define the *audioFiles* path parameter by providing the path (a `String` value) to the audio files. The path is an absolute or relative path with no trailing slashes. Because the path is relative to the edge extension classes directory, you must traverse from this directory if you enter a relative path. For example, to locate the audio files on the Oracle Sensor Edge Server home path (meaning the sample level, the locale for such directories as `config` or `persistent`), use:

```
..\..\edge\audio
```

This value sets the parameter to use the `audio` directory on top of the Oracle Sensor Edge Server home path.

> **Tip:** Using an absolute path provides greater security than the relative path.

Audio Event Supported by the Simple Audio Driver

Example 6–18 describes Event Type 207 (Audio), which is supported by the Simple Audio Driver.

Table 6–18 Event Type 207 (Audio)

Type	Subtype	Direction	Title	Description
207	1	From applications (outbound)	Play Audio File	Play audio jobs defined in the `.xml` file in the Data field.

The XML file, which is embedded in the *DATA* field, uses the standard XML-RPC call format, and has the format described in Example 6–7.

Example 6–7 XML File Embedded in the DATA Field

```
<methodCalls>
    <methodCall>
      <params>
        <param name='paramName'>paramValue</param>
      </params>
    </methodCall>
  </methodCalls>
```

Where `paramName` is the name of the parameter, and `paramValue` is its value. `<methodName>` can be omitted since the Type and Subtype define this function. You can define `paramName` using `fileName` and `url`.

For security, the value defined by the `fileName` parameter is used as the name for the audio file. The `fileName` value cannot contain any extensions and the audio file must be a `.wav` file. For example, if the `fileName` parameter's value is defined as *welcome* (as in Example 6–8), then the audio file is called `welcome.wav`.

Example 6–8 XML File That Plays welcome.wav

```
<methodCalls>
    <methodCall>
      <params>
        <param name=fileName'>welcome</param>
      </params>
    </methodCall>
  </methodCalls>
```

The `url` parameter is the URL that points to the audio file.

> **Note:** `fileName` and `url` parameters are mutually exclusive, meaning that you cannot define both parameters for the same call. If both parameters are defined, then only the latter parameter is used.

Configuring Tyco Reader Driver-Based Devices

The Tyco Reader Driver supports all readers made by Tyco readers by Sensormatic that communicate with the Oracle Sensor Edge Server either through a serial or network (TCP / IP) connection.

Configuring a Tyco Reader Driver-based device requires that you define the following parameters:

- *Connection Identifier* -- The connection type and connection identifiers. See "Setting the URI Parameters for Devices and Dispatchers".
- *Data Collection Timeout* -- The timeout (in milliseconds) for collecting data to the physical device. This is an `int` value.

- *Status Query Interval* -- The time (in milliseconds) between to consecutive status queries. This is an `int` value.

Observation Events Generated by the Tyco Reader Driver

Observation events are events that are generated by the driver and are dispatched out to the system. Table 6–4 lists the events generated by the Tyco Reader Driver.

Table 6–19 Observation Events Generated by the Tyco Reader Driver

Type	Subtype	TagID	Data	Description
200	Various	Tag ID	The data field (payload) stored in a tag.	General tag observation event.
1	0	Null	"alive or dead"	Each event using data property to indicate whether the device is alive or not.
1	0	Null	"dis"	An event indicating a successful device startup operation.
1	2	Null	"dif"	An event indicating a failed device startup operation.
1	0	Null	"dss"	An event indicating an successful device stop operation.
1	2	Null	"dsf"	An event indicating a failed device stop operation.
1	2	Null	"drf"	An event indicating a failed event-receiving operation.
1	0 or 2	Null	<callResults> <callResult> <callResult name="write-tag"> <code>0/1</code> <message/> </callResult> </callResults>	An event indicating a successful or failed tag-writing operation.

The Instruction Event Accepted by the Tyco Reader Driver

Applications instruction events to devices to tell them to perform certain tasks. Table 6–5 lists the instruction event accepted by the Tyco Reader Driver.

Table 6–20 RFID Instruction Event Accepted by the Tyco Reader Driver

Type	Subtype	TagID	Data	Description
101	0	Null	<methodCalls> <methodCall name="write-tag"> <params> <param name='tagid'>%TAGID%</param> <param name='tagdata'>%TAGDATA%</param> <param name='antennaID'>%ANTENNA_ID%</param> </params> <param name='protocol'>EPCO/CC915/CC1356/ISO15693/ISO18000-6B</param> </params> </methodCall> </methodCalls>	Write data to a tag identified by the specified tag ID.

Configuring Filter Instances

The following sections describe how the pre-seeded filters generate events and their configuration parameters:

- "Configuring the Check Tag ID Filter"
- "Using the Cross-Reader Redundant Filter"
- "Using the Debug Filter"
- "Configuring the JavaScript Filter"
- "Configuring the Movement Filter"
- "Configuring the Pallet Pass Thru Filter"
- "Configuring the Pallet Shelf Filter"
- "Configuring the Pass Filter"
- "Configuring the Polygon Filter"
- "Configuring the Regex Filter"
- "Configuring the Shelf Filter"

Configuring the Check Tag ID Filter

A Check Tag is any normal tag used to test if the device (in this case, a reader) is reading tags. Because the Check Tag itself should be physically located within the field of the reader, it should always be read; when other tags move through the field of the reader, the device also reads the Check Tag in conjunction with them.

The Check Tag ID Filter ensures that the device reads a Check Tag periodically. Using this filter enables you to check the status of a device, its corresponding reader, and attached antennae. Because the Check Tag ID Filter is used solely for diagnostic purposes, it does not provide any events for dispatching to client devices. Instead, this filter generates an event if it does not detect that the device has read a Check Tag for a specified period of time.

Note: You can apply the Check Tag ID Filter only to devices.

Table 6–21 describes the parameters (and associated values) of the Check Tag ID filter.

Table 6–21 Parameters of the Check Tag ID Filter

Name	Value Type	Description
Check Tag Id	A `String` value.	The tag ID of the Check Tag, which is the ID that the filter searches for to see if the tag is being read.
Tag Check Time Window	An `int` value.	The period of time, in milliseconds, after which an event is generated if the filter has not seen the specified Check Tag.

To define the parameters for the Check Tag ID Filter, you must note the ID of the Check Tag itself (which must be placed within the field of the reading device). Enter this ID as the String value of Check Tag Id. If the filter does not detect that a device has read a Check Tag bearing the specified ID for the period defined in the *Tag Check Time Window* parameter, it generates an event. Table 6–22 describes the signature of the generated event.

Table 6–22 The Event Signature of the Check Tag ID Filter

Event Field	Value
`sourceName`	This field identifies the originator of the event. This is an optional field; its value is set by the client.
`correlationId`	The client sets the value for this field, which is used for message responses to a particular client (such as checking if a device functions). Any message sent back by the client has the same ID. This is an optional field.
`siteName`	The name of the site that generated this event.
`deviceName`	The name of the device that generated this event.
`time`	The time that the filter generated this event.
`type`	A Message Event (type 1). Note: Applications should subscribe to this message type for notifications of when devices fail.
`subtype`	Error report (subtype 2)
`id`	The ID of the Check Tag (this is the value defined in the filter's *Check Tag Id* parameter).
`data`	An additional error message (if any).

Using the Cross-Reader Redundant Filter

The Cross-Reader Redundant Filter blocks redundant events that are sent from the devices of a device group and does not generate any events. This filter considers events redundant if it finds they have the same tag ID. The filtering occurs during a window of time corresponding to a driver read cycle.

The Cross-Reader Redundant Filter is for group-level filtering only; it performs no functions if applied to a device. This filter has no parameters to configure.

Using the Debug Filter

The Debug Filter traces events passing through the system. Upon receiving events from its associated device, this filter writes events to a log file. This filter has a single parameter called *Event Output File*. To define this parameter, enter the full path of the log file to which you want the Debug Filter to write events. (The server must make this file writable.) The format of the Debug Filter's output is as follows:

```
"Devicename: <devicename> Type: <type> Subtype: <subtype>
EventTime: <time>TagIds:<tagid(,tagid)*>Data:<dat(,data)*>\n"
```

Each event is on a separate line; each line is separated by a newline character (LF or CRLF, depending on the operating system). The `<time>` value is a long as returned by the time(2) call.

This filter can only to a device or to a device group. This filter has no parameters to configure.

Configuring the JavaScript Filter

The Javascript Filter enables you to write filter logic in a scripting language. Changes made to the source scripts are loaded dynamically, thus eliminating the need to restart the server or any components. This filter relies on an external JavaScript engine that executes the source script, such as Mozilla Rhino (`http://www.mozilla.org/rhino/`).

The JavaScript filter has a single parameter called *scriptFileName*. To define this parameter, enter the relative or full path name of the source script that the driver loads and monitors.

Writing the Source Script

The JavaScript Filter looks for functions in the source script to perform the processing. The functions are:

- `function filter(events)` -- If specified, this function is called when the filter is asked to filter out the events. This method is only called if the filter is attached to a specific device. An array of events is passed in. You can modify the events within this array, remove them, or construct a new array from scratch. You must return the array of events that you want the system to continue to process. The parameter events are an array of edge events (`oracle.edge.common.Event[]`).
- `function groupFilter(events)` -- This call is invoked when the device group calls the filter for filtering. This is only used if the filter is attached to a group instead of a single device. The parameter events are an array of edge events (`oracle.edge.common.Event[]`).
- `function instructionFilter(instruction)` -- This is called when the system attempts to an instruction event to a device. You can modify the event, or return null if you want to block it all together. The parameter is a single event (`oracle.edge.common.Event`).

You can write any one of these functions. Other functions are ignored. The initialized block is called at filter startup time. The source script described in Example 6–9 prints all of the tag IDs that it encounters to the console.

Example 6–9 Source Script

```
function filter(events)
{
    for ( i=0; i<events.length; i++ )
```

```
    {
        id = events[i].getTagId();
        java.lang.System.out.println("Id:"+id )
    }
}
```

Configuring the Movement Filter

The Movement Filter smooths out movement tracking using **Real Time Location System (RTLS)** observations by averaging out spikes or sudden motion changes from errors or interference. The Movement Filter calculates the distance between the current and previous locations. If the distance is less than the distance threshold, then the current event is filtered out.

The Movement Filter has one parameter, *Distance Threshold*. To define this parameter, enter a positive number (an `int` value) that indicates the smallest movement that should be reported.

Table 6–23 describes the signature of the Movement Filter.

Table 6–23 Signature of the Movement Filter

Event Field	Value
`sourceName`	This field identifies the originator of the event. This is an optional field; its value is set by the client.
`correlationId`	The client sets the value for this field, which is used for message responses to a particular client (such as checking if a device functions). Any message sent back by the client has the same ID. This is an optional field.
`siteName`	The name of the site that generated this event.
`deviceName`	The name of the device that generated this event.
`time`	The time that the event was generated.
`type`	An observation event. The range is 200-299.
`subtype`	*RTLS Moved* (Subtype 4)
`id`	The ID of the tag.
`data`	The data payload of the tag.

Configuring the Pallet Pass Thru Filter

The Pallet Pass Thru Filter collects all of the events received during a specified period and sends them out as a single event. When a pallet or container passes through a gateway or through the field of transmission of a reader device, this filter generates a single event for all of these tags. This filter enables you to see what items a container or pallet may hold.

The Pallet Pass Thru Filter includes the following parameters:

- Exit Event Threshold Time
- Event Collect Control Time

Exit Event Threshold Time

To define this parameter, enter the time (an `int` value), in milliseconds, since the device last read a tag before it is considered to have moved out of the device's transmission field. The parameter settings, which range from 50 milliseconds to under two seconds, dictate the frequency (that is, the reader cycle) in which the device reports these "tag is seen" events. If you set this frequency too high, such as to two seconds, then the device may miss the tag altogether.

Event Collect Control Time

To define this parameter, enter the time (an `int` value), in milliseconds, for a device to complete a reading cycle of the tags included in a pallet or container before starting a new reading cycle. When this time elapses, the reading cycle concludes (that is, the device has read all of the new tags) and the Pallet Pass Thru Filter then generates an event with the following signature (described in Table 6–24).

Table 6–24 Signature of the Pallet Pass Thru Event

Event Field	Value
`sourceName`	This field identifies the originator of the event. This is an optional field; its value is set by the client.
`correlationId`	The client sets the value for this field, which is used for message responses to a particular client (such as checking if a device functions). Any message sent back by the client has the same ID. This is an optional field.
`siteName`	The name of the site that generated this event.
`deviceName`	The name of the device that generated this event.
`time`	The time that the event was generated.
`type`	An observation event. The range is 200-299.
`subtype`	RFID Pallet Pass (Subtype 6)
`id`	A comma-separated list of tag IDs.
`data`	A comma-separated list of datum.

Configuring the Pallet Shelf Filter

The Pallet Shelf Filter collects all of the events received during a set interval and then sends them as a single event. This filter enables you identify when new containers or pallets holding many items enters an area, or exits the field or gateway of a device reader.

The Pallet Shelf Filter has the following parameters:

- Exit Event Threshold Time
- Event Collect Control Time

Exit Event Threshold Time

To define this parameter, enter the time (an `int` value), in milliseconds, from the last time that the device read the tag before it is considered to have moved out of the device's transmission field. The Pallet Shelf Filter silently clears its cache once the interval defined for the *Exit Event Threshold Time* parameter elapses and does not generate an event.

Event Collect Control Time

To define this parameter, enter the time (an `int` value), in milliseconds, for a device to complete a reading cycle for the tags of a pallet or container before starting a new reading cycle. When this time elapses, the reading cycle concludes (that is, the device has read all of the new tags) and the Pallet Shelf Filter then generates an event.

Events Generated by the Pallet Shelf Filter

The Pallet Shelf Filter generates two events:

- MULTIPLE IN FIELD Event
- MULTIPLE OUT FIELD Event

MULTIPLE IN FIELD Event

The Pallet Shelf Filter generates the MULTIPLE IN FIELD event when the device first detects the tags. This event has the following signature (described in Table 6–25):

Table 6–25 Signature of the MULTIPLE IN FIELD Event

Event Field	Value
`sourceName`	This field identifies the originator of the event. This is an optional field; its value is set by the client.
`correlationId`	The client sets the value for this field, which is used for message responses to a particular client (such as checking if a device functions). Any message sent back by the client has the same ID. This is an optional field.
`siteName`	The name of the site that generated this event.
`deviceName`	The name of the device reading the pallet or container that generated this event.
`time`	The time that the Pallet Shelf Filter generated this event.
`type`	An observation event. The range is 200-299.
`subtype`	RFID Pallet In-Field (Subtype 4)
`id`	A comma-separated list of tag IDs.
`data`	A comma-separated list of datum.

MULTIPLE OUT FIELD Event

The Pallet Shelf Filter generates the MULTIPLE OUT FIELD event when the interval defined for the *Exit Event Threshold Time* parameter elapses. This event has the following signature (described in Table 6–26):

Table 6–26 Signature of the MULTIPLE OUT FIELD Event

Event Field	Value
`sourceName`	This field identifies the originator of the event. This is an optional field; its value is set by the client.

Table 6–26 (Cont.) Signature of the MULTIPLE OUT FIELD Event

Event Field	Value
`correlationId`	The client sets the value for this field, which is used for message responses to a particular client (such as checking if a device functions). Any message sent back by the client has the same ID. This is an optional field.
`siteName`	The name of the site generating this event.
`deviceName`	The name of the device reading the pallet or container that generated this event.
`time`	The time that the Pallet Shelf Filter generated this event.
`type`	An observation event. The range is 200-299.
`subtype`	RFID Pallet Out Filed (Subtype 5)
`id`	A comma-separated list of Tag IDs.
`data`	A comma-separated list of datum.

Configuring the Pass Filter

When a tag passes through the range of transmission, or through the gateway of a device reader, it generates a series of "tag is seen" events. The device reports these events periodically, starting when the tag enters the transmission field. The reporting stops when the tag moves out of the reader field.

Applications often do not require the series of events that a device reader generates; instead, these applications only need to know that a tag has passed through a device's gateway or range of transmission. The Pass Filter applies to such situations, as it reduces all of the "tag is seen" events into single events for each unique tag that passes through the field of a reader device.

The Pass Filter has one parameter, *Exit Event Threshold Time*. To define this parameter, enter the time (an `int` value), in milliseconds, since the device last read the tag before it is considered to have moved out of the device's transmission field. The parameter settings, which range from 50 milliseconds to under two seconds, dictate the frequency (that is, the reader cycle) in which the device reports these "tag is seen" events. If you set this frequency too high, such as to two seconds, then the device may miss the tag altogether.

When the device first reads a tag, the Pass Filter caches the tag's ID (tagid), notes the time that the tagid was read into the cache, and then immediately sends the pass-through event. The filter blocks subsequent reads for this cached tagid. Each time the filter receives a new read from the device, it updates the time that it read the tagid into the cache. If the sum of the caching time and the value set for *Exit Event Threshold Time* is less than the current time, then the Pass Filter clears the tagid from the cache. The next time the device reads this tag, the filter considers it a new event, caches its tagid and sends out a new pass-through event.

The pass through event has the following signature (described in Table 6–27).

Table 6–27 Signature of the Pass-Through Event

Event Field	Value
`sourceName`	This field identifies the originator of the event. This is an optional field; its value is set by the client.
`correlationId`	The client sets the value for this field, which is used for message responses to a particular client (such as checking if a device functions). Any message sent back by the client has the same ID. This is an optional field.
`siteName`	The name of the site that generated this event.
`deviceName`	The name of the device that generated this event.
`time`	The time that the event was generated.
`type`	An observation event. The range is 200-299.
`subtype`	A pass-through event (subtype 3).
`id`	The ID of the tag.
`data`	The data payload of the tag.

Configuring the Polygon Filter

The Polygon Filter filters out all movement observations reported by **Real Time Location System (RTLS)** devices and generates events only if the tag moves in or out of any predefined polygons. The polygons are defined using a set of *x*, *y* coordinates that define the vertices and parenthesis. For example: *((x,y), (x,y), ...), (....), ...*

The Polygon Filter has a single parameter, *PolygonCoordinates*, which you define by entering a `String` value in the format of *((x1, y1), (x2, y2)...,(xn, yn))*.

Configuring the Regex Filter

The Regex Filter performs a regular expression search that looks for tags to either remove or to allow to pass through the streams. This filter enables you to define a set of patterns for the filter to search for in any of the event's fields. When the filter finds matches to the search criteria, it allows the event to pass through the system; if it finds no matches, then it filters out the event. The RegexFilter is a generic filter and can work with any Event Type.

The Regex Filter has a single parameter called *allow*. To define this parameter, enter a regular expression (a `String`) for the events that are allowed to pass through the filter in the following format:

<field>~<pattern>&&<field>~<pattern>||...

Where `<field>` is the name of the event field that the filter matches. The filter supports the following fields:

- TAGID
- DATA
- CORRELATIONID
- SOURCENAME
- TYPE
- SUBTYPE
- DEVICENAME

- SITENAME
- TIME

> **Note:** the TIME field is converted to the `time() long` value before it is compared to the `<pattern>`.

`<pattern>` is a standard regular expression, && is a AND connective, and || is a OR connective. For example, to enable the RegexFilter instance to look for all events with the Event Type 103 and a tag with first three digits matching *0FE*, enter:

```
type~103&&tagid~^0FE
```

Configuring the Shelf Filter

The Shelf Filter is a device-level filter that generates events when a tag is detected within the field of a reader or when the tag has left the field. Like the Pass Filter, the Shelf Filter has a single parameter, *Exit Event Threshold Time*. To define this parameter, enter the time (an `int` value), in milliseconds, since the device last read the tag before it is considered to have moved out of the device's transmission field. Unlike the Pass Filter, however, the Shelf Filter silently clears its cache once the interval defined for the *Exit Event Threshold Time* parameter elapses and does not generate an event.

Events Generated by the Shelf Filter

The Shelf Filter generates two events:

- IN FIELD Event
- OUT FIELD Event

IN FIELD Event

The Shelf Filter generates this event when the device first detects the tag. This event has the following signature (described in Table 6–28).

Table 6–28 Signature of the IN FIELD Event

Event Field	Value
`sourceName`	This field identifies the originator of the event. This is an optional field; its value is set by the client.
`correlationId`	The client sets the value for this field, which is used for message responses to a particular client (such as checking if a device functions). Any message sent back by the client has the same ID. This is an optional field.
`siteName`	The name of the site that generated this event.
`deviceName`	The name of the device that generated this event.
`time`	The time that the Shelf Filter generated this event.
`type`	An observation event. The range for observation events is 200-299.

Table 6–28 (Cont.) Signature of the IN FIELD Event

Event Field	Value
subtype	An in-field event (subtype 2)
`id`	The ID of the tag.
`data`	The data payload of the tag.

OUT FIELD Event

The Shelf Filter generates this event when the interval defined for the *Exit Event Threshold Time* parameter elapses. This event has the following signature (described in Table 6–29).

Table 6–29 Signature of the OUT FIELD Event

Event Field	Value
`sourceName`	This field identifies the originator of the event. This is an optional field; its value is set by the client.
`correlationId`	The client sets the value for this field, which is used for message responses to a particular client (such as checking if a device functions). Any message sent back by the client has the same ID. This is an optional field.
`siteName`	The name of the site that generated this event.
`deviceName`	The name of the device that generated this event.
`time`	The time that the Shelf Filter generated this event.
`type`	An observation event. The range for an observation event is 200-299.
`subtype`	An out-field event (subtype 2).
`id`	The ID of the tag.
`data`	The data payload of the tag.

When a device first detects the tag, the Shelf Filter caches the ID of the tag and then generates an `IN FIELD` event. At this point, the tag is read during every reader cycle. While the tag may not be read during some of these cycles, it is read during others. When the device does not read the tag consistently for a period longer than that designated for the *Event Exit Threshold Time* parameter, then the filter removes the tag's ID from the cache and generates an `OUT FIELD` event. The devices stops reading the tag once it passes from the field of the device.

Managing Dispatchers for an Oracle Sensor Edge Server Instance

The main output of the Oracle Sensor Edge Server is filtered data events. The devices and respective filters both normalize and minimize this event data. Out of the box, the Oracle Sensor Edge Sever delivers event data through the following dispatcher methods:

- Web Services

- HTTP / Post
- EPC PML
- Application Level Events (ALE)
- Oracle Streams / Advanced Queueing

The following sections describe how to configure these dispatcher methods.

- "Configuring the Edge Dispatcher to Use Oracle Streams"
- "Configuring the Dispatcher to Send Event Messages to a Web Service"
- "Configuring the Dispatcher to Send Event Messages Through HTTP"
- "Configuring the PML Dispatcher"
- "Configuring the Template Dispatcher"
- "Configuring the ALEDispatcher"
- "Using the Null Dispatcher"

Configuring the Dispatcher to Send Event Messages to a Web Service

A client device or application can register a SOAP call which the Oracle Sensor Edge Server invokes when a new message must be delivered.

To configure the WebService Dispatcher to distribute event messages through Web Services, enter the service URL of the WSDL (Web Service Definition Language) document that describes the client call. This URL must point to the EndPoint (port) of the Web Service. For example, enter `http://localhost:8888/wsdl/mytest.wsdl`. This document must contain the `portType` of `EdgeClientCallback` and the call, `processEvent`, as its child element. Upon startup, the Oracle Sensor Edge Server attempts to connect and bind to the service defined in this WSDL document.

Configuring the Dispatcher to Send Event Messages Through HTTP

Configuring the dispatcher to route events to clients using HTTP 1.0 results in the Oracle Sensor Edge Server posting each event message to the client separately. Because the Oracle Sensor Edge Server performs these posts sequentially, if one post is blocked, then all following posts are also blocked.

To configure the Oracle Sensor Server to route events using HTTP define the following parameters:

- *URL* -- Enter the URL of the servlet, JSP, or CGI to which the Oracle Sensor Edge Server posts event messages whenever they are dispatched. To configure this dispatcher, enter the URL in the following format:

  `http://hostname:port/serverPath`

  If the Oracle Sensor Edge Server uses the HTTP dispatcher, then the client interface must tell the Oracle Sensor Edge Server how (and when) to call it.

- *proxyServer* -- Enter the proxy server IP or host name.
- *proxyPort* -- Enter the proxy server port. For example, enter *80*.

Configuring the PML Dispatcher

The PML Dispatcher posts events in the EPC PML format using the **Transport Library** for such protocols as HTTP, FTP, and FILE.

> **Note:** PML is a legacy format used to report observations.

To configure the Oracle Sensor Server to use the PML Dispatcher, define the destination URI. See "Setting the URI Parameters for Devices and Dispatchers".

Configuring the Template Dispatcher

The Template Dispatcher (also known as the EventFlowDispatcher), sends and receives events from different sources and processes them through a finite state machine configured using the `eflow.xml` configuration file. This dispatcher sends and receives both to and from HTTP, POJ, Web Services, HTTP, and direct data streams using the Transport Library.

To configure the Oracle Sensor Server to use the Template Dispatcher, enter the location of the `eflow.xml` file.

Configuring the ALEDispatcher

Use the ALEDispatcher with the ALE Web Services interface to notify any ALE subscribers when a report specification is satisfied.

Using the Null Dispatcher

The Null dispatcher, which is created by default, discards all of the events passed to it. These events are not saved or spooled. Use this dispatcher only if you want to prevent the Oracle Sensor Edge Server from dispatching events.

Configuring the Edge Dispatcher to Use Oracle Streams

Configuring the Oracle Sensor Edge Server to use Oracle Streams and Advanced Queuing enables you to control how the edge dispatcher retrieves and distributes event messages. Unlike the Web Services, or HTTP dispatcher options, event messages dispatched using the Oracle Streams dispatcher do not have to be relayed directly to an entry point. The Oracle Streams dispatcher supports rule-based process and agent technologies. In addition, the Streams dispatcher only supports UTF-8 encoding.

Tips:

- Because Oracle Streams enables the propagation and management of data, transactions, and events in a data stream on one -- or across many -- databases, this dispatcher option provides the greatest flexibility of the seeded dispatcher options.
- The Oracle Streams dispatcher requires JDK 1.4.x.

Event messages are data that is deposited to a staging area (an internal queue). This data, in turn, can be aggregated in different ways for different client devices and applications (the consumers of the event messages). Using Oracle Streams as the dispatcher, the Data and Event layer of the database, not the Oracle Sensor Edge Server or applications, determines what an event is and when it is generated. The Data

and Event layer provides a rule-based process that determines the appropriate event message for each client device or application.

Once the event messages are captured and placed into the staging queue, the event message data can be processed through the Rules Evaluation Job, which takes event messages from the staging queue and then compares them to the Oracle Sensor Edge Server rule set. Each rule has an action, which is executed if the rule applies. These actions include a PL/SQL callback for propagating the event message to other queues which could then be consumed by other applications.

In addition to these rule-based actions, the Rules Evaluation Job invokes applications by calling the Sensor Data Repository (SDR), which receives sensor data from the Oracle Sensor Edge Server or from other sources. The SDR includes a set of archive tables that store all of the filtered sensor events in the system. For more information, see Chapter 4, "Using the Sensor Data Repository".

Note: Applications requiring raw, unfiltered event data that has not been processed by the rules can connect to the staging area using AQ notification.

To configure the Streams Dispatcher, enter the maximum number of threads to create for processing outbound observations in the *threadnum* field. Because each thread holds a connection, this is also the number of connections to the Sensor Data Repository uses for outbound events (there is another connection used for in-bound events). 0 is an invalid value. See also "Manually Configuring Sensor Data Repository and Sensor Data Streams in Release 10.1.3".

Glossary

antenna

Each tag has at least one antenna. On the other side of the communication link, the reader must also have an antenna. Some readers can drive multiple antennae at the same time. Depending on the protocol, frequency and application, these antennae vary from thin strips of metal laid across a surface, to a portal doorway antenna that is meters tall

chip

A silicon chip, with embedded memory, is used in a tag. The chip implements the wireless protocol and access functions to its embedded memory. Note that in *Active Tags*, this is not a single chip but an entire board. See **tag**.

device

An edge device is an end point of a sensor- based architecture, such as a Radio-Frequency Identification (RFID) reader, a dry contact, an laser diode, carousel, scale, robotic picker, or indication devices such as light sticks or message boards. Sensors are hardware or software end points that make observations of certain changes in state. While this is usually a physical change, such as a laser diode detecting that something has blocked the line of its beam, it can also a software observation of a change occurring within software, such as when a monitor daemon running on an edge controller exits. Sensors also observe defects in software. A device is an instance of a driver.

event

An event is a message that is sent from either a sensor device or an application that signals that a state has changed. The Sensor Edge Server, which receives the data from these sensor devices or applications, normalizes the contents of these event messages by putting them in a common data format and then applies filters to strip them of extraneous information or unwanted events.

Oracle Sensor Edge Server

The server that resides between all of the readers and the application middle tier. It is responsible for interfacing with all of the readers and sending normalized data back to the application server.

Radio Frequency Identification (RFID)

RFID is the use of small transponders with embedded Electronic Serial Numbers (ESNs) or memory, which transmit identifiers across one or more frequencies.

reader

A reader reads from, and writes to, the tags to which it is connected. Readers usually have serial interfaces used to communicate with a host computer. There is no widely-accepted standard for this protocol. The process of retrieving data stored on an RFID tag by sending radio waves to the tag and then converting the waves the tag sends back into data is known as a read.

reader field

The area of coverage for a reader. If tags are outside of a reader field, then they cannot receive radio waves and cannot be read.

Real Time Location System (RTLS)

A technology that uses radio-frequency to produce real-time location information for tagged items.

tag

(Also known as an RFID tag.) A single unit that contains a chip, one or more antennae, and a power source. If it is battery-driven or from a external source, the tag is an *Active Tag*. If the power source is inductive-based (which means that it relies on photoelectric effect to generate power from remotely generated radio waves), the tag is a *Passive Tag*. A tag containing data that cannot be changed is a read-only tag. See **chip**.

Transport Library

A set of classes that abstracts a communication channel, such serial port or TCP / IP. The Transport Library allows the addition of third-party plug-ins.

transports.xml

The file located in the `edgeHome/config` directory which tells the **Transport Library** which transports are available. For example:

```
<transports> <transport name="com" type="stream"
class="oracle.edge.tools.io.SerialConnection"> <params> <param name='port'
value='com1' /> <param name='driver' value='com.sun.comm.Win32Driver'/> </params>
</transport>
<transport name="tcp" type="stream" class="oracle.edge.tools.io.TcpConnection" />
<transport name="file" type="stream" class="oracle.edge.tools.io.FileConnection"
/>
<transport name="stdout" type="stream"
class="oracle.edge.tools.io.StdoutConnection" />
<transport name="http" type="stream" class="oracle.edge.tools.io.HttpConnection" >
<params> <param name='proxyServer' value='www-proxy.us.oracle.com' /> <param
name='proxyPort' value='80'/> </params> </transport>
</transports>
```

Each `<transport>` element defines a new transport type. For each `<transport>` tag you must define name (a unique name for this transport, which is used in the URI) and class, which specifies which class to load. This class must implement the `oracle.edge.tools.io.Connection` interface.

You can also specify default parameters for transports using the `<param>` tag.

Index

A

B

C

D

E

F

H

I

J

K

L

M

N

O

P

Q

R

S

T