IBM® Tivoli® Netcool/OMNIbus Probe for JDBC
2.0

Reference Guide
July 20, 2017

SC27-5610-02

Notice

Before using this information and the product it supports, read the information in Appendix A, “Notices and Trademarks,” on page 35.

Edition notice

This edition (SC27-5610-02) applies to version 2.0 of IBM Tivoli Netcool/OMNIbus Probe for JDBC and to all subsequent releases and modifications until otherwise indicated in new editions.

This edition replaces the previous version SC27-5610-01.

Contents

About this guide

The following sections contain important information about using this guide.

Document control page

Use this information to track changes between versions of this guide.

They documentation is provided in softcopy format only. To obtain the most recent version, visit the IBM® Tivoli® Information Center:

http://publib.boulder.ibm.com/infocenter/tivihelp/v8r1/index.jsp?topic=/com.ibm.tivoli.namomnibus.doc/welcome_ptsm.htm

Table 1. Document modification history

Document version	Publication date	Comments
SC27-5610-00	March 14, 2013	First IBM publication.
SC27-5610-01	March 7, 2014	"Summary" on page 1 updated. Description for the **DisablePidFileLock** property added in "Properties and command line options" on page 26. Descriptions for the **InitialResync** and **ResyncInterval** properties updated to ... "Properties and command line options provided by the Java Probe Integration Library (probe-sdk-java) version 4.0" on page 28.
SC27-5610-02	July 20, 2017	"Example usage" on page 3 added.

Conventions used in this guide

All probe guides use standard conventions for operating system-dependent environment variables and directory paths.

Operating system-dependent variables and paths

All probe guides use standard conventions for specifying environment variables and describing directory paths, depending on what operating systems the probe is supported on.

For probes supported on UNIX and Linux operating systems, probe guides use the standard UNIX conventions such as **$***variable* for environment variables and forward slashes (**/**) in directory paths. For example:

```
$OMNIHOME/probes
```

For probes supported only on Windows operating systems, probe guides use the standard Windows conventions such as **%***variable***%** for environment variables and backward slashes (****) in directory paths. For example:

```
%OMNIHOME%\probes
```

For probes supported on UNIX, Linux, and Windows operating systems, probe guides use the standard UNIX conventions for specifying environment variables and describing directory paths. When using the Windows command line with these probes, replace the UNIX conventions used in the guide with Windows conventions. If you are using the bash shell on a Windows system, you can use the UNIX conventions.

Note : The names of environment variables are not always the same in Windows and UNIX environments. For example, %TEMP% in Windows environments is equivalent to $TMPDIR in UNIX and Linux environments. Where such variables are described in the guide, both the UNIX and Windows conventions will be used.

Operating system-specific directory names

Where Tivoli Netcool/OMNIbus files are identified as located within an *arch* directory under NCHOME or OMNIHOME, *arch* is a variable that represents your operating system directory. For example:

`$OMNIHOME/probes/`*`arch`*

The following table lists the directory names used for each operating system.

Note : This probe may not support all of the operating systems specified in the table.

Table 2. Directory names for the arch variable

Operating system	**Directory name represented by *arch***
AIX® systems	`aix5`
Red Hat Linux® and SUSE systems	`linux2x86`
Linux for System z	`linux2s390`
Solaris systems	`solaris2`
Windows systems	`win32`

OMNIHOME location

Probes and older versions of Tivoli Netcool/OMNIbus use the OMNIHOME environment variable in many configuration files. Set the value of OMNIHOME as follows:

- On UNIX and Linux, set `$OMNIHOME` to `$NCHOME/omnibus`.
- On Windows, set `%OMNIHOME%` to `%NCHOME%\omnibus`.

Chapter 1. Probe for JDBC

The Probe for JDBC extracts data from databases that support the JDBC standard.

This guide contains the following sections:

Summary

Each probe works in a different way to acquire event data from its source, and therefore has specific features, default values, and changeable properties. Use this summary information to learn about this probe.

Table 3. Summary

Probe target	Databases that support the JDBC standard.
Probe executable name	`nco_p_jdbc`
Probe installation package	`omnibus-`*`arch`*`-probe-nco-p-jdbc-`*`version`*
Package version	2.0
Probe supported on	For details of supported operating systems, see the following Release Notice on the IBM Software Support Website: https://www-304.ibm.com/support/docview.wss?uid=swg21665217
Properties file	`$OMNIHOME/probes/`*`arch`*`/jdbc.props`
Rules file	`$OMNIHOME/probes/`*`arch`*`/jdbc.rules`
Requirements	For details of any additional software that this probe requires, refer to the `description.txt` file that is supplied in its download package.
Connection method	JDBC
Remote connectivity	The probe can connect to a remote database.

Table 3. Summary (continued)	
Multicultural support	Available This probe supports multibyte characters. Whether the database to which you are connecting supports multibyte characters depends on the JDBC driver. Consult your JDBC driver documentation for multibyte character support information.
Peer-to-peer failover functionality	Available
IP environment	IPv4 and IPv6 **Note :** IPv6 support for the database connection depends on the JDBC driver. Consult your JDBC driver documentation for IPv6 support information.
Federal Information Processing Standards (FIPS)	IBM Tivoli Netcool/OMNIbus uses the FIPS 140-2 approved cryptographic provider: IBM Crypto for C (ICC) certificate 384 for cryptography. This certificate is listed on the NIST website at http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/1401val2004.htm. For details about configuring Netcool/OMNIbus for FIPS 140-2 mode, see the *IBM Tivoli Netcool/OMNIbus Installation and Deployment Guide*.

Installing probes

All probes are installed in a similar way. The process involves downloading the appropriate installation package for your operating system, installing the appropriate files for the version of Netcool/OMNIbus that you are running, and configuring the probe to suit your environment.

The installation process consists of the following steps:

1. Downloading the installation package for the probe from the Passport Advantage Online website.

   Each probe has a single installation package for each operating system supported. For details about how to locate and download the installation package for your operating system, visit the following page on the IBM Tivoli Knowledge Center:

   http://www-01.ibm.com/support/knowledgecenter/SSSHTQ/omnibus/probes/all_probes/wip/reference/install_download_intro.html
2. Installing the probe using the installation package.

   The installation package contains the appropriate files for all supported versions of Netcool/OMNIbus. For details about how to install the probe to run with your version of Netcool/OMNIbus, visit the following page on the IBM Tivoli Knowledge Center:

   http://www-01.ibm.com/support/knowledgecenter/SSSHTQ/omnibus/probes/all_probes/wip/reference/install_install_intro.html
3. Configuring the probe.

   This guide contains details of the essential configuration required to run this probe. It combines topics that are common to all probes and topics that are peculiar to this probe. For details about additional configuration that is common to all probes, see the *IBM Tivoli Netcool/OMNIbus Probe and Gateway Guide*.

Example usage

In this example, the JDBC Probe is used to read events from an Oracle database.

To use the JDBC Probe to read events from an Oracle database, you need to identify the equivalent of the `LastOccurrence` field of the ObjectServer in the table that the JDBC Probe reads. You can then use this field as the **`MarkerColumn`** property in the JDBC Probe properties file.

The select statement that you specify using the **`SelectSqlFile`** property must compare the **`MarkerColumn`** with the local system time of the database. So that the JDBC Probe can use the **`MarkerColumn`**, it must be converted to a number (if it is not a number already).

The following configuration assumes that `LastOccurrence` is a date field in the table `ORACLE_TABLE` on the Oracle database <ORACLE_SID> on the server <ORACLE_SERVER_IP>. With the <probe-user> having the correct permissions and settings to access the table:

1. Set the `CLASSPATH` environment variable to reflect the required Oracle JDBC client jar files:

   ```
   CLASSPATH=/opt/instantclient/classes12.jar:/opt/instantclient/ojdbc14.jar
   export CLASSPATH
   ```

2. Set the following properties in the `jdbc.props` file:

   ```
   SelectSqlFile : '$NCHOME/omnibus/probes/linux2x86/oracle11g_jdbc_probe.sql'
   MessageLog    : '$NCHOME/omnibus/log/jdbc_probe_oracle11g.log'
   PidFile       : '$NCHOME/omnibus/var/jdbc_probe_oracle11g.pid'
   JdbcDriver    : 'oracle.jdbc.driver.OracleDriver'
   JdbcUrl       : 'jdbc:oracle:thin:@<ORACLE_SERVER_IP>:1521:<ORACLE_SID>'
   DBUsername    : '<probe-user>'
   DBPassword    : '<probe-user-password>'
   DataBackupFile : '$NCHOME/omnibus/var/jdbc_probe_oracle11g'
   MarkerColumn   :
   "TO_NUMBER(TO_CHAR(LASTOCCURRENCE ,'yyyymmddhh24miss')) AS MARKER"
   ```

3. Define a select statement in the SQL file (`oracle11g_jdbc_probe.sql`) specified by the **`SelectSqlFile`** property suitable for the target table `ORACLE_ALERTS`:

   ```
   select TO_NUMBER(TO_CHAR(LASTOCCURRENCE ,'yyyymmddhh24miss')) as MARKER,
   IDENTIFIER, NODE, FIRSTOCCURRENCE, LASTOCCURRENCE from ORACLE_ALERTS
   ```

Data acquisition

Each probe uses a different method to acquire data. Which method the probe uses depends on the target system from which it receives data.

Acquiring data using the JDBC Probe consists of the following basic steps:

1. Setting the connection parameters.
2. Setting the authentication parameters.
3. Specifying an appropriate SELECT statement.
4. Setting the resynchronization parameters.

Note : Before running the probe, you must have installed the JDBC drivers for your operating system. A list of the JDBC drivers is included in the Gateway for JDBC Reference Guide.

See the following page on the information center:

http://publib.boulder.ibm.com/infocenter/tivihelp/v8r1/topic/com.ibm.netcool_OMNIbus.doc/gateways/jdbcgw/jdbcgw/wip/reference/jdbcgw_db_support.html

Full details of how the probe acquires data are described in the following topics:

- "Connecting to an event source using JDBC" on page 4
- "Connecting through either IPv4 or IPv6" on page 7
- "Authenticating with the data source" on page 7

Connecting to an event source using JDBC

Connecting to an event source involves downloading and installing the JDBC driver, and specifying appropriate values for the connection-related properties.

JDBC drivers

You must obtain the JDBC driver for the target database from the database vendor and install it according to the vendor's instructions. The drivers are usually provided as Java™ archives (`.jar`).

Setting the CLASSPATH variable

Environment variables are specific preset values that establish the working environment of the probe. From the environment variable specified, the probe receives path information for the directories in which library files are present.

Set the CLASSPATH variable to the full path of the JDBC jar file that you installed for your datasource. For example:

```
c:\\DB2\\db2jcc.jar
```

On UNIX or Linux operating systems, use the following command:

```
export CLASSPATH=/home/jdbc_driver
```

where *jdbc_driver* is the full path of the JDBC jar file that you installed for your datasource.

On Windows operating systems, use the following command:

```
set CLASSPATH=c:\jdbc_driver
```

where *jdbc_driver* is the full path of the JDBC jar file that you installed for your datasource.

Database connection properties

To enable the probe to communicate with the target database, you must specify values for the following properties:

- **JdbcDriver**: This property specifies the JDBC driver required to connect the database.
- **JdbcUrl**: This property specifies the URL of the target database.
- **DBUsername**: This property specifies the user name for the target database.
- **DBPassword**: This property specifies the password for the target database.

The format in which you specify the **JdbcDriver** and **JdbcUrl** properties depend on the type of database to which the probe is connecting.

The following table lists example values for the **JdbcDriver** and **JdbcUrl** properties for use with each database supported by the probe. Consult your driver documentation for more information about setting up database connections. Default values may be different depending on your setup.

Table 4. Example JDBC property values	
DB2® LUW	
JdbcDriver	`com.ibm.db2.jcc.DB2Driver`
JdbcUrl	`jdbc:db2://`*`host_name`*`:`*`port`*`/`*`db_name`* Where *host_name* is the name of the database host machine, *port* is the port number, and *db_name* is the name of the database. For example: `jdbc:db2://server.example.ibm.com:9999/REPORTER`
DB2 z/OS®	
JdbcDriver	`com.ibm.db2.jcc.DB2Driver`
JdbcUrl	`jdbc:db2://`*`host_name`*`:`*`port`*`/`*`db_name`* Where *host_name* is the name of the database host machine, *port* is the port number, and *db_name* is the name of the database. For example: `jdbc:db2://server.example.ibm.com:9999/REPORTER`
Informix®	
JdbcDriver	`com.informix.jdbc.IfxDriver`
JdbcUrl	`jdbc:informix-sqli://`*`host_name`*`:`*`port`*`/`*`db_name`*`:INFORMIXSERVER=`*`server_name`* Where *host_name* is the name of the database host machine, *port* is the port number, *db_name* is the name of the database, and *server_name* is the same as the *host_name*. For example: `jdbc:informix-sqli://server.example.ibm.com:1433/REPORTER:INFORMIXSERVER=server.example.ibm.com`
Microsoft SQL Server	
JdbcDriver	`com.microsoft.sqlserver.jdbc.SQLServerDriver`

Table 4. Example JDBC property values (continued)	
JdbcUrl	`jdbc:sqlserver://`*`host_name`*`:`*`port`*`;databaseName=`*`db_name`* Where *host_name* is the name of the database host machine, *port* is the port number, and *db_name* is the name of the database. The default *port* is 1433. For example: `jdbc:sqlserver://server.example.ibm.com:1433;databaseName=REPORTER`
MySQL	
JdbcDriver	`com.mysql.jdbc.Driver`

JdbcUrl	`jdbc:mysql://`*`host_name`*`[,`*`failover_host`*`]:`*`port`*`/`*`db_name`*`[?`*`param1=value1`*`&`*`param2=value2`*`]`
	Where *host_name* is the name of the database host machine, *failover_host* is the name of the optional failover host, *port* is the port number, *db_name* is the name of the database, and *param1* and *param2* are optional parameters. The default *port* is 3306. For example:
	`jdbc:mysql://server.example.ibm.com:3306/alerts`

Oracle	
JdbcDriver	`oracle.jdbc.driver.OracleDriver`
JdbcUrl	`jdbc:oracle:thin:@`*`host_name`*`:`*`port`*`:`*`db_name`* Where *host_name* is the name of the database host machine, *port* is the port number, and *db_name* is the name of the database. The default *port* is 1521. For example: `jdbc:oracle:thin:@server.example.ibm.com:1521:REPORTER`
Sybase	
JdbcDriver	`com.sybase.jdbc4.jdbc.SybDriver`

JdbcUrl	`jdbc:sybase:Tds:`*`host_name`*`:`*`port`*`/`*`db_name`*`[?`*`property=value`*`;]`
	Where *host_name* is the name of the database host machine, *port* is the port number, *db_name* is the name of the database, and *property* is an optional parameter. For example:
	`jdbc:sybase:Tds:server.example.ibm.com:1521/REPORTER`

Connecting through either IPv4 or IPv6

The probe can be installed and run in either an IPv4 environment or in an IPv6 environment. The environment that you are using and the database to which you are connecting determines the format that you must use for the **JdbcUrl** property.

When you specify the URL for the database, you must include the details of both the IP address and the database to which you are connecting. The format of the URL that you specify for the **JdbcUrl** property depends on the type of database that you are using. The examples for IPv4 and IPv6 given in this topic are for MS SQL databases. If you are using a different database, you must consult the documentation supplied with that database for details of the format that you need to use for its URL.

Specifying the JdbcUrl for MS SQL databases when operating in an IPv4 environment

If you are running the probe in an IPv4 environment and connecting to MS SQL databases, you must specify the **JdbcUrl** property in the following format:

```
JdbcUrl : "jdbc:sqlserver://ipv4_address:port;databaseName=database_name"
```

where:

- *ipv4_address* is the IP address of the machine on which the MS SQL database is running.
- *port* is the port number to which the probe connects.
- *database_name* is the name of the MS SQL database.

Example entry for MS SQL running in an IPv4 environment:

```
JdbcUrl : "jdbc:sqlserver://9.180.212.183:1433;databaseName=RealSecureDB"
```

Specifying the JdbcUrl for MS SQL databases when operating in an IPv6 environment

If you are running the probe in an IPv6 environment and connecting to MS SQL databases, you must specify the **JdbcUrl** property in the following format:

```
JdbcUrl : "jdbc:sqlserver://;serverName=ipv6_address\
\instance_name;port=port_number;databaseName=database_name"
```

where:

- *ipv6_address* is the IP address of the machine on which the MS SQL database is running.
- *instance_name* is the name running instance of the MS SQL database.
- *port_number* is the port to which the probe connects.
- *database_name* is the name of the MS SQL database.

Example entry for MS SQL running in an IPv6 enviroment:

```
JdbcUrl : "jdbc:sqlserver://;serverName=2001:15f8:106:194:d173:eed2:ee3e:5143\
\MSSQLSERVER;port=1433;databaseName=RealSecureDB"
```

Authenticating with the data source

The probe uses the values specified by the **DBUsername** and **DBPassword** properties to authenticate with the data source.

You can specify both the user name and password in plain text format, or you can use an encryption algorithm to secure the entry made in the properties file.

To encrypt either the user name or password in the properties file, use the following steps:

1. Generate a key and store it in a key file using the nco_keygen utility.

2. Set the generic **ConfigCryptoAlg** property in the probe properties files to the encryption method required.
3. Set the generic **ConfigKeyFile** property in the probe properties files to the path of the encryption key file.
4. Encrypt the user name and password using the nco_aes_crypt utility.
5. Add the encrypted values for the **DBPassword** property and the **DBUsername** property generated by the nco_aes_crypt utility to the probe properties file.

Note : If you run the probe in FIPS mode, you must either use no encryption, or you must use nco_aes_crypt with the cipher (-c) option AES_FIPS. For example:

```
$NCHOME/omnibus/bin/nco_aes_crypt -c AES_FIPS -k key_file string_value
```

The cipher option used here must match the option specified by the **ConfigCryptoAlg** property.

For information about using nco_keygen and nco_aes_crypt to encrypt the user name and password, see the following topics:

- "Generating a key in a key file" on page 8
- "Specifying the key file as a property" on page 8
- "Encrypting a string value with the key" on page 9
- "Adding an encrypted value to the properties file" on page 9

Generating a key in a key file

Run the nco_keygen utility to generate a key and store it in a key file. Command-line options are available for you to either specify a hexadecimal value for the key, or to specify a length in bits for automatic key generation.

Run the nco_keygen utility to generate a key and store it in a key file. Command-line options are available for you to either specify a hexadecimal value for the key, or to specify a length in bits for automatic key generation.

To generate a key within a key file:

Run nco_keygen as follows. Optional entries are shown in square brackets.

```
$NCHOME/omnibus/bin/nco_keygen -o key_file [-l length | -k key]
```

In this command:

key_file represents the output file path and file name to which the key is saved.

length represents the length in bits of the key, as specified by you. This number must be divisible by 8 to make a whole number of bytes. The default is 128. Only 128, 192, and 256 are valid key lengths for AES encryption.

key represents the value of the key in hexadecimal digits, as specified by you. You can use either the -l or -k command-line option, but not both.

If you use the -o command-line option to specify an output file name, and omit both the -l and -k options, a randomly-generated 128-bit key is written to the file.

The nco_keygen utility writes the key to the file, using the format length:key, where length is the number of bits in the key, represented as ASCII decimal numerals, and key is the key data.

Specifying the key file as a property

In the properties file in which you want to specify an encrypted string value, set the value of the **ConfigKeyFile** property to the file path and file name of the key file that was generated by the nco_keygen utility.

ConfigKeyFile is a generic Netcool/OMNIbus property in the probe properties file:

$OMNIHOME/probes/*arch*/*probename*.props

Encrypting a string value with the key

Use the `nco_aes_crypt` utility to encrypt a string value with the key that was generated by the `nco_keygen` utility.

To encrypt a string value:

Run `nco_aes_crypt` as follows:

`$NCHOME/omnibus/bin/nco_aes_crypt -c` *`cipher`* `-k` *`key_file string_value`*

In this command:

- *`cipher`* is the algorithm that is used to encrypt the string value. Specify one of the following values for cipher, based on your mode of operation:
  - FIPS 140–2 mode: Specify AES_FIPS.
  - Non-FIPS 140–2 mode: Specify either AES_FIPS or AES. Use AES (the default) only if you need to maintain compatibility with passwords that were encrypted using the tools provided in versions earlier than Tivoli Netcool/OMNIbus V7.2.1.
- *`key_file`* is the file path and name of the key file. This value must match that specified for the **ConfigKeyFile** property in the properties file.
- *`string_value`* is the user name or password that you want to encrypt.

The output is displayed in the console window in encrypted form, and is delimited with `@` symbols. You can now copy the output text, including the `@` symbols, for use within the relevant properties file. For example:

```
@44:CcnsqcefNPVeVIXFhflYv2z04lxNtvIPL5DKvP2QU+M=@
```

Adding an encrypted value to the properties file

After encrypting a string value, add it to the properties file within which you want to hide the actual value.

Add the properties whose values have been encrypted to the probe properties file along with the corresponding encrypted values.

Set the generic Netcool/OMNIbus **ConfigCryptoAlg** property in the probe properties file to the cryptographic algorithm to use when decrypting the string; for example, AES_FIPS or AES.

Note : This value must match that specified when you ran `nco_aes_crypt` with the `-c` setting, to encrypt the string value.

Configuring the probe to retrieve data from ISS Site Protector

You can use the JDBC Probe to acquire events from ISS SiteProtector. To do so, you must use the alternative rules file and some additional configuration files that have been written for this purpose.

To support ISS SiteProtector, you require the following files:

- `iss_siteprotector.rules`: This is the alternative rules file that you should specify in the **RulesFile** property instead of `jdbc.rules`.
- `sitepro.include.lookup`: This file is referenced by the rules file.
- `sitepro.post.include`: This include file allows the probe to use a modified ObjectServer schema.
- `select_rules.sql`: This file contains the mandatory select query that the probe uses to acquire data from ISS SiteProtector.

These files are supplied within the probes installation package. You should check the contents of the `select_rules.sql` query and make any changes required to suit your environment.

You will also need to make various updates to the `jdbc.props` file.

To configure the connection to the ISS SiteProtector, set the following properties in the `jdbc.props` file:

```
DBPassword      : 'password'
DBUsername      : 'user_name'
JdbcDriver         : 'com.microsoft.sqlserver.jdbc.SQLServerDriver'
JdbcUrl              : 'jdbc:sqlserver://localhost:1433;databaseName=RealSecureDB'
```

To configure the probe to use the select query written for ISS SiteProtector, set the following property in the `jdbc.props` file:

```
SelectSqlFile    : 'C:\\IBM\\Tivoli\\Netcool\\omnibus\\var\\select_rules.sql'
```

To configure the probe to use the rules file written for ISS SiteProtector, set the following property in the `jdbc.props` file:

```
RulesFile        :
'C:\\IBM\\Tivoli\\Netcool\\omnibus\\probes\\win32\\iss_siteprotector.rules'
```

Updating the rules file

`sitepro.include.lookup` and `sitepro.post.include` are referenced from the `iss_siteprotector.rules` file by `include` statements. You will need to update these `include` statements to reflect full paths to these files in your probe installation. Open the rules file, search for the two commented out include statements that reference `sitepro.include.lookup` and `sitepro.post.include` and update their respective paths.

For example, on Windows operation systems, replace the commented out include statements with:

`include "`*`$OMNIHOME`*`/probes/win32/sitepro.include.lookup"`

`include "`*`$OMNIHOME`*`/probes/win32/sitepro.post.include"`

Where *`$OMNIHOME`* is the full path to the probe installation.

On Unix and Linux operating systems, replace the commented out include statements with:

`include "`*`$OMNIHOME`*`/probes/includes/sitepro.include.lookup"`

`include "`*`$OMNIHOME`*`/probes/includes/sitepro.post.include"`

Where *`$OMNIHOME`* is the full path to the probe installation.

Configuring the ObjectServer schema

The `sitepro.post.include` file contains the following set of field/element definitions that have been commented out:

```
# @NsProtocol = $NsProtocol
# @NsEventType = $NsEventType
# @NsClass = $NsClass
# @NsCVE = $NsCVE
# @NsThreatCategory = $NsThreatCategory
# @NsThreatType = $NsThreatType
# @NsVirusName = $NsVirusName
# @NsVendor = $NsVendor
# @NsProduct = $NsProduct
# @NsVersion = $NsVersion
# @NsPatch = $NsPatch
# @NsRaw = $NsRaw
# @NsScore = $NsScore
# @NsConfidentiality = $NsConfidentiality
# @NsIntegrity = $NsIntegrity
# @NsAvailability = $NsAvailability
# @NsRate = $NsRate
```

```
# @NsAlertType = $NsAlertType
# @NsAlertTypeDesc = $NsAlertTypeDesc
```

You must edit the `sitepro.post.include` file to uncomment these definitions and then create the fields indicated in the ObjectServer. For details of creating fields, see the *Netcool/OMNIbus Installation and Deployment Guide*.

Handling open-form SQL statement queries to retrieve data from the event source

The probe can handle open-form SQL statement queries to retrieve data from the event source.

Mandatory SELECT statement

To enable the probe to receive any events from the data source, the **SelectSqlFile** property must be set to a file containing an SQL SELECT statement. The way in which you configure and prepare that SELECT statement depends on what table you are interested in and what data from that table you want to receive as an event.

The mandatory query in the file specified by **SelectSqlFile** property must contain a SELECT statement that can return data.

Note : The JDBC Probe only supports SQL files that have no more than 50,000 characters. If the SQL query file contains more than 50,000 characters, the probe will write the following message to the error log:

```
SQL file has exceeded the max characters limit..
```

Example SELECT statements

The following examples show simple SELECT statements.

Example 1

The following example returns events that contain the `eventid`, `event_name`, `event_desc`, `severity`, and `category` fields from all records in the `active_alarms` table that have the `resolved` field set to `0`.

```
SELECT eventid, event_name, event_desc, severity, category FROM active_alarms
WHERE resolved = 0;
```

Example 2

The following example retrieves all data from the `SensorData1` table (the main table used by ISS SiteProtector to store sensor data).

```
SELECT * FROM SensorData1;
```

The following examples retrieve data from the `SensorData1` table, and enrich and convert it using SQL clauses.

Example 3

This example uses `convert` clauses:

```
SELECT convert(int,SensorDataRowID) as SensorDataRowID,
convert(varchar(64),SensorDataID) as SensorDataID,
convert(varchar(120),AlertName) as AlertName, AlertDateTime, AlertID,
convert(varchar(100),SensorName) as SensorName, ProductID, AlertTypeID,
AlertPriority, AlertFlags, convert(varchar(10),SensorAddressInt) as
SensorAddressInt, convert(varchar(10),SrcAddressInt) as SrcAddressInt,
convert(varchar(10),DestAddressInt) as DestAddressInt, ProtocolID, SourcePort,
convert(varchar(64),SourcePortName) as SourcePortName,
convert(varchar(64),DestPortName) as DestPortName,
convert(varchar(64),UserName) as UserName, ProcessingFlag, Cleared, HostGUID,
```

```
convert(varchar(64),HostDNSName) as HostDNSName,
convert(varchar(64),HostNBName) as HostNBName,
convert(varchar(64),HostNBDomain) as HostNBDomain,
convert(varchar(64),HostOSName) as HostOSName,
convert(varchar(64),HostOSVersion) as HostOSVersion, HostOSRevisionLevel,
ObservanceID, VulnStatus, AlertCount, convert(varchar(64),ObjectName) as
ObjectName, ObjectType, OSGroupID, ComponentID, SensorGUID, LicModule,
convert(varchar(64),VLan) as VLan, convert(varchar(64),VirtualSensorName) as
VirtualSensorName from SensorData1
```

Example 4

This example uses `CASE` and `INNER JOIN` clauses.

```
SELECT CASE WHEN sd.SensorDataID IS NULL THEN '-1' ELSE sd.SensorDataID END AS
'SensorDataID',CASE WHEN sd.AlertName IS NULL THEN '' ELSE sd.AlertName END AS
'AlertName',CASE WHEN sd.AlertDateTime IS NULL THEN '-1' ELSE sd.AlertDateTime
END AS 'AlertDateTime',CASE WHEN sd.AlertID IS NULL THEN '-1' ELSE sd.AlertID
END AS 'AlertID',CASE WHEN sd.AlertPriority IS NUL L THEN '-1' ELSE
sd.AlertPriority END AS 'AlertPriority',case WHEN sd.Cleared IS NULL THEN ''
ELSE sd.Cleared END AS 'Cleared',CASE WHEN sd.ProtocolID IS NULL THEN '-1' ELSE
sd.ProtocolID END AS 'ProtocolID',CASE WHEN sd.SrcAddressInt IS NULL THEN '-1'
ELSE sd.SrcAddressInt END AS 'SourceAddressLong',CASE WHEN sd.DestAddressInt IS
NULL THEN '-1' ELSE sd.DestAddressInt END AS 'DestAddressLong',CASE WHEN
sd.SourcePort IS NULL THEN '-1' ELSE sd.SourcePort END AS 'SourcePort',CASE
WHEN sd.ObjectName IS NULL THEN '' ELSE sd.ObjectName END AS 'ObjectName',CASE
WHEN sd.SensorName IS NULL THEN '' ELSE sd.SensorName END AS 'SensorName',CASE
WHEN sd.SensorAddressInt IS NULL THEN '-1' ELSE sd.SensorAddressInt END AS
'SensorAddressLong',CASE WHEN secchk.ChkName IS NULL THEN '' ELSE
secchk.ChkName END AS 'ChkName',CASE WHEN secchk.ChkBriefDesc IS NULL THEN ''
ELSE secchk.ChkBriefDesc END AS 'ChkBriefDesc',CASE WHEN secchk.SecChkID IS
NULL THEN '-1' ELSE secchk.SecChkID END AS 'SecChkID' FROM SensorData sd INNER
JOIN Observances obs ON sd.ObservanceID = obs.ObservanceID INNER JOIN
SecurityChecks secchk ON obs.SecChkID = secchk.SecChkID
```

Optional SQL statements

As well as the mandatory SELECT statement, you can also specify SQL statements for the probe to perform before and after this statement. This processing is optional. You can specify what SQL statements the probe performs using the following properties:

- **PreSqlFile** - allows you to specify a file containing an SQL statement to perform before the mandatory SELECT statement.
- **PostSqlFile** - allows you to specify a file containing an SQL statement to perform after the mandatory SELECT statement.

Configuring periodic resynchronization time intervals

You can configure the probe to perform a resynchronization with the data source at startup, or at regular intervals while the probe is running.

If you set the **InitialResynch** property to `true`, the probe requests all active alarms from the data source at startup.

The **ResynchInterval** property controls the interval (in seconds) at which the probe requests to receive outstanding active alarms. If you set this property to `0`, the probe will not make periodic requests to receive outstanding active alarms.

Note : To enable the probe to receive any alarms from the data source, you must set the **InitialResynch** property to `true`, or the **ResynchInterval** property to a value greater than `0`.

Configuring partial resynchronization

The probe can perform a partial resynchronization by selecting all active alarms that have not yet been retrieved. This resynchronization is based on a timestamp associated with the alarms.

To enable the probe to perform a partial resynchronization, you must specify an appropriate column from the source database to act as a marker using the **MarkerColumn** property.

Before deciding which marker column to select, bear in mind the following guidelines:

- The column that you specify must be either an integer or a unix timestamp.
- The column should be an incremental row indicator or an incremental timestamp in the table.
- If the source database is not ordered by the column that you specify, you must order the records retrieved by this column.
- You must either include the column explicitly in the SELECT statement specified by the **SelectSqlFile** property, or it must be selected by a wildcard within the SELECT statement, for example:

  ```
  SELECT * from TABLE table_name
  ```

  For details about specifying the SELECT statement, see "Handling open-form SQL statement queries to retrieve data from the event source" on page 11.
- Ensure that the marker column is unique and do not duplicate the name in query SQL.
- Do not use comments /* */ inside the SELECT query.
- Define only one SQL query in the SQL file.
- Define only one marker indicator within the SQL query.
- Define only one marker column.
- If you need to configure a marker column as a conversion (using CONVERT, CAST, DATEDIFF, or DATEADD), you must use it with the AS keyword; for example:

  ```
  CONVERT(int, marker) AS marker_column
  ```

The probe can be configured to perform a partial resynchronization based on the last resynchronization marker stored in a data backup file. To do so, the probe uses the **DataBackupFile** property and the **MarkerColumn** property together. The probe records the last read alarm (as defined by the marker column specified by the **MarkerColumn** property) in the file specified by the **DataBackupFile** property. Before performing a resynchronization, the probe reads the data backup file and retrieves only those alarms that have been created since the previous resynchronization. If the data backup file is empty (as it will be during the initial run of probe), the probe will do a full resynchronization. If a marker column has not been specified, the probe ignores the **DataBackupFile** property.

Example configuration 1: Using a simple SELECT statement

By default, the probe will add a where clause for the marker column to the end of the mandatory SELECT statement.

For example, suppose the file specified by the **SelectSqlFile** property contains the following SQL command:

```
SELECT * from SensorData1
```

and the **MarkerColumn** property is set to SensorDataRowID

The probe will execute the following SQL command:

```
SELECT * from SensorData1 WHERE SensorDataRowID > ?
```

Where ? is a dynamic value retrieved from the last resynchronization cycle. If there have been no resynchronization cycles yet, the probe sets ? to 0.

Example configuration 2: Using a SELECT statement that contains a WHERE clause

If you are using a query that already contains a `where` clause, the probe will add a `where` clause for the marker column to the end of the mandatory SELECT statement.

Suppose the file specified by the **SelectSqlFile** property contains the following SQL command:

```
SELECT * from SensorData1 WHERE AlertPriority > 2
```

and the **MarkerColumn** property is set to `SensorDataRowID`

The probe will execute the following SQL command:

```
SELECT * from SensorData1 WHERE AlertPriority > 2 AND SensorDataRowID > ?
```

This will work in the same way as Example 1, but will only include alerts whose priority is greater than 2.

Example configuration 3: Using a marker indicator with the SELECT statement

If you are using a more complex query in which a `where` clause for the marker column cannot be added to the end of the query, you must include the indicator `::marker_column` in the SELECT query. This indicates to the probe where the `where` clause should be expanded.

For example, suppose the file specified by the **SelectSqlFile** property contains the following SQL command:

```
SELECT sp.sup_name, sp.street, sp.city, sp.zip, sp.sup_id FROM suppliers
sp ::marker_column order by sp.sup_id
```

and the **MarkerColumn** property is set to `sp.sup_id`

The probe will execute the following SQL command:

```
SELECT sp.sup_name, sp.street, sp.city, sp.zip, sp.sup_id FROM suppliers sp
WHERE sp.sup_id > ? order by sp.sup_id
```

Example configuration 4: Using a more complex SELECT statement

Suppose the file specified by the **SelectSqlFile** property contains the following SQL command:

```
SELECT a.name, a.id, b.salary, b.increment FROM employee a, emp_salary b WHERE
a.id = b.id ::marker_column AND n.name is NOT NULL order by a.id
```

and the **MarkerColumn** property is set to `b.salary`

The probe will execute the following SQL command:

```
SELECT a.name, a.id, b.salary, b.increment FROM employee a, emp_salary b WHERE
a.id = b.id AND b.salary > ? AND a.name is NOT NULL order by a.id
```

Note : You must place the marker indicator within the SQL statement in a location that will produce valid SQL when the `WHERE` clause is expanded. That location will be either directly after the `WHERE` keyword or after a completed `WHERE` clause. For example:

```
SELECT * from SensorData1 WHERE ::marker_column ObservanceID > 0 order by
SensorDataRowID
```

or

```
SELECT * from SensorData1 WHERE ObservanceID > 0 ::marker_column order by
SensorDataRowID
```

Example configuration 5: Converting a column to unix timestamp format

If you want to select a DateTime field in the source database as the marker column, you must convert it into unix timestamp format using the AS keyword within the SELECT statement.

For example, suppose you are using MySQL and the file specified by the **SelectSqlFile** property contains the following SQL command:

```
SELECT unix_timestamp(ts) AS timex, no_id from T1
```

Where the `ts` column is of type DateTime in the source table, and the `unix_timestamp()` function is converting this column to unix timestamp format.

Note : If you are using a database other than MySQL, you may need to use a different SQL conversion function. See the documentation supplied with your database for details.

The **MarkerColumn** property should be set to `unix_timestamp(ts) AS timex`.

The probe will execute the following SQL command:

```
SELECT unix_timestamp(ts), no_id from T1 WHERE unix_timestamp(ts) > ?
```

Where ? is a dynamic value retrieved from the last resynchronization cycle or from the recovery file specified by the **DataBackupFile** property.

Example configuration 6: Converting a column using DATEDIFF

You can convert a marker column using the DATEDIFF function.

For example, suppose the file specified by the **SelectSqlFile** property contains the following SQL command:

```
SELECT *, DATEDIFF(s, '19700101', AlertDateTime) AS AlertDateTime from
SensorData1
```

The **MarkerColumn** property should be set to "`DATEDIFF(s, '19700101', AlertDateTime) AS AlertDateTime`".

The probe will execute the following SQL command:

```
SELECT *, DATEDIFF(s, '19700101', AlertDateTime) AS AlertDateTime from
SensorData1 WHERE DATEDIFF(s, '19700101', AlertDateTime) > ?
```

Where ? is a dynamic value retrieved from the last resynchronization cycle or from the recovery file specified by the **DataBackupFile** property.

Other examples

Suppose the **MarkerColumn** is set to: "`a_emp_id`"

You could use the following SQL command in the file specified by the **SelectSqlFile** property:

```
"SELECT a.emp_id as a_emp_id, b.emp_id as b_emp_id FROM table a, table b WHERE
a.class_id = b.emp_id"
```

But you could not use following SQL command in the file specified by the **SelectSqlFile** property:

```
"SELECT a.emp_id, b.emp_id FROM table a, table b WHERE a.class_id = b.emp_id"
```

Suppose the **MarkerColumn** is set to: "`DATEDIFF(s, '19700101', sd.AlertDateTime) AS AlertDateUnixTime`"

You could use following SQL command in the file specified by the **SelectSqlFile** property:

```
"Select DATEDIFF(s, '19700101', sd.AlertDateTime) AS AlertDateUnixTime,
sd.AlertDateTime from SensorData1 sd"
```

Suppose the **MarkerColumn** is set to: "`DATEDIFF(s, '19700101', sd.AlertDateTime) AS AlertDateTime`"

You could not use following SQL command in the file specified by the **SelectSqlFile** property:

```
"Select DATEDIFF(s, '19700101', sd.AlertDateTime) AS AlertDateTime,
sd.AlertDateTime from SensorData1 sd"
```

Acquiring data from case-insensitive and case-sensitive databases

The probe supports database that operate in either a case-sensitive environment or in a case-insensitive environment.

Database case-sensitivity considerations for when specifying the select query

Most databases treat table column names as case-insensitive. However, some databases, for example Sybase, treat table column names as case-sensitive. When defining the query that the probe will use to select events from the database, you must make sure that you do not use the same column name twice in different cases; for example, the query cannot contain both `ColumnName` and `COLUMNNAME`.

Case-insensitive and case-sensitive databases

Setting the **`MarkerColumnSensitive`** property correctly allows you to use the JDBC Probe to acquire data from both types of database. In most cases, you will set the **`MarkerColumnSensitive`** property set to `false`. This will support situations in which the probe compares the column of the result set returned with the marker column, ignoring the case when matching.

The examples in the rest of this topic describe various scenarios and how to set the **`MarkerColumnSensitive`** property in each situation.

Example configuration 1: Case-insensitive environment, query matching case of marker column

Suppose the database operates in a case-insensitive environment (for example: Microsoft SQL), and the query result is in the same case as the marker column selected.

For example, suppose the **`MarkerColumn`** is set to:

```
"DATEDIFF(s, '19700101', sd.AlertDateTime) AS UnixTime"
```

And the file specified by the **`SelectSqlFile`** property contains the following SQL command:

```
"SELECT DATEDIFF(s, '19700101', sd.AlertDateTime) AS UnixTime, SensorDataRowID
FROM SensorData1 sd"
```

The probe will try to match the marker column `UnixTime` with the query result returned. MySQL will return the result as `UnixTime` and so the probe can find the marker column correctly.

In this scenario, it does not matter whether the **`MarkerColumnSensitive`** property is set to either TRUE or FALSE. This is because the probe can match the marker column either case-sensitive or case-insensitive. So leave this property set to its default value of FALSE.

Example configuration 2: Case-insensitive environment, query not matching case of marker column

Suppose the database operates in a case-insensitive environment (for example: DB2) and the query result is in a different case to that of the marker column selected.

For example, suppose the **`MarkerColumn`** is set to:

```
"(timestampdiff(2, char(lastmodified - timestamp('1970-01-01-00.00.00')))) AS
unixtime"
```

And the file specified by the **`SelectSqlFile`** property contains the following SQL command:

```
"SELECT (timestampdiff(2, char(lastmodified -
timestamp('1970-01-01-00.00.00')))) AS unixtime, user_id FROM user"
```

The probe will try to match the marker column `unixtime` with the query result returned. However, DB2 will return the query result in full capital case, that is: `UNIXTIME`, which is the standard behaviour of DB2.

In this scenario, you should set the **MarkerColumnSensitive** property to FALSE. This is because the probe will try to match the marker column `unixtime` with the returned result `UNIXTIME`, so must do so case-insensitive. In this case, probe able to find the matchable marker column with the query result that enable the partial resync performing correctly.

If the **MarkerColumnSensitive** property had been set to TRUE. The probe will be unable to match the marker column `unixtime` with the returned result `UNIXTIME`. The probe will write a warning message to the probe log and it will perform a full resynchronization (because the probe is unable to find a matchable marker column to use with the query result).

Example configuration 3: Case-sensitive environment, query not matching case of marker column

Suppose the database operates in a case-sensitive environment (for example: Sybase) and the query selects the same name but in a different case.

For example, suppose the **MarkerColumn** is set to: "`a.Identifier`"

And the file specified by the **SelectSqlFile** property contains the following SQL command:

```
"Select a.Identifier, as.IDENTIFIER, a.AlarmID FROM Alarm a, AlarmStatus as
WHERE a.AlarmID = as.AlarmID"
```

In a case-sensitive environment, the probe must compare the marker column case-sensitive.

In this scenario, you should set the **MarkerColumnSensitive** property to TRUE This will enable the probe to match the marker column `Identifier` with the correct case `Identifier` in the returned result, and will prevent the probe from mistakenly matching it with the wrong case `IDENTIFIER`.

Running pre-selection and post-selection processing queries on the event source

The probe can perform SQL queries on the source data before and after it has been selected by the query specified by the **SelectSqlFile** property. You specify appropriate queries to run using the **PreSqlFile** and **PostSqlFile** properties, respectively.

To specify a query that the probe performs before selecting events from the data source, use the **PreSqlFile** property. To specify a query that the probe performs after selecting events from the data source, use the **PostSqlFile** property.

The pre-selection and post-selection queries can contain any of the following SQL statements:

- `INSERT`
- `UPDATE`
- `DELETE`
- `INSERT INTO ... SELECT`
- `CREATE TABLE ... SELECT`
- `ALTER`
- `TRUNCATE`
- `DROP`

Within the pre-selection or post-selection queries, you cannot use `SELECT ... FROM ...` statements that return a result set. But you can use `INSERT into ... SELECT` or `SELECT INTO` statements that select from, or query, an existing table and insert new rows into that table.

If either the **PreSqlFile** or **PostSqlFile** query fails, the probe will still run the **SelectSqlFile** query, but will also write a ProbeWatch message to the log file. If the **SelectSqlFile** query fails, the probe will not run the **PostSqlFile** query.

Note : The complexity of the queries that you can specify using the **PreSqlFile** or **PostSqlFile** properties depends on the database from which the probe is extracting data. For example, if you are

connecting to MS SQL, you can specify files that contain multiple queries. However, if you are connecting to DB2, you can only specify files that contain a single query. If you are connecting to DB2 and you specify either a pre-selection file or post-selection file that contains multiple queries, the queries will fail and the probe will log the error.

Note also that the JDBC Probe only supports SQL files that have no more than 50,000 characters. If any of your SQL query files contains more than 50,000 characters, the probe will write the following message to the error log:

```
SQL file has exceeded the max characters limit..
```

Example 1: Performing an insert and a delete

The following example performs an insert and a delete:

The **PreSQLFile** query inserts an event with a SensorDataRowID of 8881 into the database table.

The **SelectSQLFile** query checks that the event has been created in the database table.

The **PostSQLFile** query deletes the event from the database table.

The query specified by **PreSQLFile** contains the following code:

```
SET IDENTITY_INSERT SensorData1 ON
INSERT INTO SensorData1
(SensorDataRowID,SensorDataID,AlertName,AlertDateTime,AlertID,SensorName,ProductID,
AlertTypeID,AlertPriority,AlertFlags,SensorAddressInt,SrcAddressInt,
VALUES (8881,8882,'TEST_INSERT','','',1,2,3,4,1,2,3,4,5,'','',1,'','','',
'','','',1,2,3,'',1,2,3,'','','',1,'',1,2,3,4,1,2,3,4,5,'','')
SET IDENTITY_INSERT SensorData1 OFF
```

The query specified by **SelectSQLFile** contains the following code:

```
SELECT * FROM SensorData1 WHERE SensorDataRowID='8881'
```

The query specified by **PostSQLFile** contains the following code:

```
DELETE FROM SensorData1 WHERE SensorDataRowID='8881'
```

Example 2: Performing an update and then performing a second an update

The following example performs an update and then performs a second update:

The **PreSQLFile** query updates existing events that have AlertName set to TEST_UPDATE, to TEST_UPDATE_HAS_BEEN_UPDATED.

The **SelectSQLFile** query selects the total number of events that have been updated.

The **PostSQLFile** query updates value of the AlertName column of the updated events to FLUSHED_TO_JDBC_PROBE.

The query specified by **PreSQLFile** contains the following code:

```
UPDATE SensorData1 SET AlertName='TEST_UPDATE_HAS_BEEN_UPDATED'
WHERE AlertName='TEST_UPDATE'
```

The query specified by **SelectSQLFile** contains the following code:

```
SELECT COUNT(AlertName) AS TOTAL_UPDATED_ALERT FROM SensorData1
WHERE AlertName='TEST_UPDATE_HAS_BEEN_UPDATED'
```

The query specified by **PostSQLFile** contains the following code:

```
UPDATE SensorData1 SET AlertName='FLUSHED_TO_JDBC_PROBE'
WHERE AlertName='TEST_UPDATE_HAS_BEEN_UPDATED'
```

Example 3: Creating a new table and deleting its contents

The following example creates a new database table, then deletes its contents:

The **PreSQLFile** query creates a simple database table.

The **SelectSQLFile** query checks that events are populated in the new table.

The **PostSQLFile** query clears all the data in the new table at the end of each resynchronization interval.

The query specified by **PreSQLFile** contains the following code:

```
CREATE TABLE new_table
(
P_Id int,
LastName varchar(255),
FirstName varchar(255),
Address varchar(255),
City varchar(255)
)
```

The query specified by **SelectSQLFile** contains the following code:

```
SELECT * FROM new_table
```

The query specified by **PostSQLFile** contains the following code:

```
DELETE FROM new_table
```

Example 4: Creating and dropping a new table

The following example creates a new database table, then drops it:

The **PreSQLFile** query creates a simple database table.

The **SelectSQLFile** query checks that events are populated in the new table.

The **PostSQLFile** query drops the new table at the end of each resynchronization interval.

The query specified by **PreSQLFile** contains the following code:

```
CREATE TABLE Persons
(
P_Id int,
LastName varchar(255),
FirstName varchar(255),
Address varchar(255),
City varchar(255)
)
```

The query specified by **SelectSQLFile** contains the following code:

```
SELECT * FROM Persons
```

The query specified by **PostSQLFile** contains the following code:

```
DROP TABLE Persons
```

Specifying whether the probe writes SQL warning messages to the probe log file

The probe can handle SQL queries for the selection of events from the source data. The probe can also perform SQL queries on the data before and after it has been selected. You can specify that any database-specific warning messages that are generated as a result of running SQL queries are written to the probe log file.

To specify that the probe writes any database-specific warning messages that result from running SQL queries to the probe log file, set the **SqlWarnings** property to `true`. By default, the **SqlWarnings**

property is set to `true`. If you do not want the probe to write the SQL messages to the log file, set the **`SqlWarnings`** property to `false`.

Each vendor has its own conditions that trigger SQL warnings. If you set the **`SqlWarnings`** property to `true`, which messages are written to the log file depends on which database the probe is extracting data from. If you set the **`SqlWarnings`** property to `false`, the probe will write no database-specific warning messages regardless of the success or failure of the SQL queries specified by the **`SelectSqlFile`**, **`PreSqlFile`**, and **`PostSqlFile`** properties.

Displaying unicode and non-unicode characters

The probe can support multibyte characters and so can display both unicode and non-unicode characters.

Before using the probe to process data that contains multibyte characters, perform the following steps:

1. Check that your database is configured to enable multibyte characters. See the documentation supplied with your database for details of multibyte character support.
2. Consult your JDBC driver documentation to confirm whether your driver supports multibyte characters and whether you need to make any changes to the environment settings of your server or workstation.

   Note : You specify to which database the probe connects using the **`JdbcUrl`** property. When processing data that contains multibyte characters, some JDBC drivers require you to specify explicitly UTF-8 encoding within URLs, others automatically detect character encoding. This will affect how you set the **`JdbcUrl`** property. See the documentation supplied with your JDBC driver for details of how you should specify URLs for databases that support multibyte characters.
3. Check that the probe server has UTF-8 support enabled and that the correct locale is set; for example, set the locale to Chinese.

   On Windows operating systems, use the following steps:

   a. Access the **Region and Language** section of the **Control Panel**.
   b. Select the **Formats** tab.
   c. Select **Format** > **Chinese (Simplified, PRC)**
   d. Select the **Administrative** tab.
   e. Select **Change system locale**
   f. Select **Current system locale** > **Chinese (Simplified, PRC)**
   g. Click **OK**.
   h. Click **OK**.

   On UNIX and Linux operating systems, set the system locale using the `LANG` and `LC_ALL` environment variables:

   ```
   export LANG=zh_CN.utf8
   export LC_ALL=zh_CN.utf8
   ```
4. Configure the ObjectServer to enable the insertion of UTF-8 encoded data. See the *Netcool/OMNIbus Installation and Deployment Guide*.
5. If you are running the probe on a Windows operating system, you must use the `-utf8enabled` command-line option each time you start the probe.

Customizing the timestamp that the probe adds to each event received

The probe can create a timestamp for each event received. This consists of two steps: specifying the column name from the source data that the probe will use to create the timestamp and specifying the format that the probe will use for the timestamp.

By default, Netcool/OMNIbus uses the `FirstOccurrence` and `LastOccurence` fields to indicate event timestamps. These two fields are in UTC UNIX timestamp format. Whatever column from the data source you choose to assign to these default OMNIbus fields, you must be format them as a timestamp using

either SQL or the rules file. You can specify that the probe uses one of the following formats for the timestamp:

- UNIX timestamp format - expressed as the number of seconds that have elapsed since Jan 1, 1970
- General textual representation format - expressed in a customizable combination of years, months, days, hours, minutes, and seconds; for example: 2013-01-29 10:45:00

There are two ways in which you can define the format of the timestamp: within the probe rules file or within the SQL SELECT statement. The method that you chose depends on how you want to convert or format the timestamp.

- **Scenario 1:** The column that you are using for the timestamp is in a general textual representation format and you want convert it to UNIX timestamp format. In this scenario, you can specify the timestamp using the rules file method or the SQL method.
- **Scenario 2:** The column that you are using for the timestamp is in UNIX timestamp format and want to convert it to a general textural representation format. In this scenario, you must use the rules file method.
- **Scenario 3:** The column that you are using for the timestamp is in a general textual representation format and you want to convert it to a different general textual representation format. In this scenario, you can either use SQL to convert to UNIX format and then use the rules file method to convert the timestamp to a different general textual representation format, or you can use SQL method.

Both methods are described at the end of this topic.

Method 1: Defining the format of the timestamp within the probe rules file

To define the format of the timestamp as described in **Scenario 1** within the probe rules file, use the following steps:

1. Configure the SQL statement in the file specified by the **SelectSqlFile** property to specify the name of the column that the probe will use to create a timestamp for each event. For example, the following MS SQL command selects, and makes available for converting into a timestamp, the `AlertDateTime` from ISS SiteProtector:

   ```
   Select AlertDateTime, SensorDataID, AlertName from SensorData1
   ```

2. Configure the rules file to specify the format that the probe will use for each timestamp field. For example, the following code in the rules file instructs the probe to convert the selected column from a `yyyy-MM-dd hh:mm:ss` textual representation into UNIX timestamp format:

   ```
   if( exists ( $AlertDateTime ) ) {
   ### AlertDateTime original format "yyyy-MM-dd hh:mm:ss",
   ### for example: 2013-01-29 10:45:00
   $AlertDateTime = datetotime($AlertDateTime, "yyyy-MM-dd hh:mm:ss")
   @FirstOccurrence =$AlertDateTime
   @LastOccurrence = $AlertDateTime
   }
   ```

Note : In **Scenario 2**, you can use the same method, but using the `timetodate` function instead of the `datetotime` function.

For details about using the `datetotime` function and the `timetodate` function within the probe rules file, see the Netcool/OMNIbus Probe and Gateway Guide.

Method 2: Defining the format of the timestamp within the SQL SELECT statement

To define the format of the timestamp as described in **Scenario 1** within the SQL SELECT statement, use the following steps:

1. Configure the SQL statement in the file specified by the **SelectSqlFile** property to specify the name of the column that the probe will use to create a timestamp and to convert it into UNIX timestamp format. For example, the following MS SQL command selects `AlertDateTime` and converts it into UNIX timestamp format:

```
Select DATEDIFF(s, '19700101', AlertDateTime) AS UTC_AlertDateTime,
SensorDataID, AlertName from SensorData1
```

2. Map the converted UNIX timestamp onto a Netcool/OMNIbus field. For example, the following code maps $UTC_AlertDateTime to a Netcool/OMNIbus field:

```
if( exists ( $UTC_AlertDateTime ) ) {
### AlerDateTime original format in UTC
@FirstOccurrence =$UTC_AlertDateTime
@LastOccurrence = $UTC_AlertDateTime
}
```

Note : In **Scenario 3**, you can use the same method, but using the CONVERT() function instead of the DATEDIFF() function. For example, to convert AlertDateTime from yyyy-MM-dd hh:mm:ss format to MMM dd yyyy hh:mmAM(or PM) format, use the following MS SQL SELECT statement:

```
SELECT CONVERT(VARCHAR(24),AlertDateTime,100) AS AlertDateTime FROM SensorData1
```

Specifying what the probe does during inactivity

The probe has a timeout facility that allows it to disconnect from the data source if it fails to receive the next alarm data within a predefined amount of time.

To specify how long the probe waits before disconnecting, use the **Inactivity** property. After this length of time, the probe disconnects from the data source, and flushes any outstanding events to the ObjectServer.

If you set the **Inactivity** property to 0, the probe never disconnects from the data source regardless of whether there is a period of inactivity.

Reconnecting to the event source and the probe backoff strategy

The reconnection functionality allows you to specify how the probe behaves if it loses its connection to the event source. You can specify whether the probe attempts to reconnect to the data source, the maximum number of reconnection attempts the probe makes, and the frequency with which the probe makes those attempts.

You configure the reconnection functionality using the **RetryCount** and **RetryInterval** properties.

To specify how many times the probe attempts to reconnect to the data source, use the **RetryCount** property. If you set the **RetryCount** property to 0 and the probe fails to establish a connection or loses an existing connection to the data source, the probe will not attempt to reconnect to the data source.

To specify the frequency (in seconds) with which the probe attempts to reconnect to the data source, use the **RetryInterval** property. If you set the **RetryInterval** property to 0 and the probe fails to establish a connection or loses an existing connection to the data source, the probe reverts to a backoff strategy. The probe tries to reestablish a connection after one second, then two seconds, then four seconds, then eight seconds, and so on, up to a maximum of 4096 seconds.

Note : If the probe has previously connected to the database, attempted a resynchronization, and subsequently lost its connection, the probe will not attempt to the reconnect to the event source. This is because there may be a problem with the SELECT statement specified to retrieve events. The probe will write an error message to the log file and shut down. You will need to consult the error log to determine why the probe disconnected from the data source.

Peer-to-peer failover functionality

The probe supports failover configurations where two probes run simultaneously. One probe acts as the master probe, sending events to the ObjectServer; the other acts as the slave probe on standby. If the master probe fails, the slave probe activates.

While the slave probe receives heartbeats from the master probe, it does not forward events to the ObjectServer. If the master probe shuts down, the slave probe stops receiving heartbeats from the master and any events it receives thereafter are forwarded to the ObjectServer on behalf of the master probe.

When the master probe is running again, the slave probe continues to receive events, but no longer sends them to the ObjectServer.

Configuring peer-to-peer functionality

When configuring two probes in a failover pair, you should specify unique values for the following properties so that the two probe instances do not attempt to overwrite each others files:

- **PidFile**
- **PropsFile**
- **DataBackupFile**

You should also specify a different rules file for each instance using the **RulesFile** property. You should, however, specify the same values for the following properties, because these properties relate to the database from which the two probes acquire events and how they acquire those events:

- **SelectSqlFile**
- **DBPassword**
- **DBUsername**
- **JdbcDriver**
- **JdbcUrl**
- **MarkerColumn**
- **InitialResynch**
- **ResynchInterval**

Example property file settings

You set the peer-to-peer failover mode in the properties files of the master and slave probes. The settings differ for a master probe and slave probe.

The following example shows the peer-to-peer settings from the properties file of a master probe:

```
Mode              : "master"
DataBackupFile : "C:\\IBM\\Tivoli\\Netcool\\omnibus\\var\\RecoveryFile_master"
RulesFile       : "master_rules_file"
MessageLog     : "master_log_file"
PeerHost       : "slave_hostname"
PeerPort       : 9988 # [communication port between master and slave probe]
PidFile        : "C:\\IBM\\Tivoli\\Netcool\\omnibus\\var\\jdbc_m.pid"
```

The following example shows the peer-to-peer settings from the properties file of the corresponding slave probe:

```
Mode           : "slave"
DataBackupFile : "C:\\IBM\\Tivoli\\Netcool\\omnibus\\var\\RecoveryFile_slave"
RulesFile       : "slave_rules_file"
MessageLog      : "slave_log_file"
PeerHost         : "master_hostname"
PeerPort         : 9988 # [communication port between master and slave probe]
PidFile        : "C:\\IBM\\Tivoli\\Netcool\\omnibus\\var\\jdbc_s.pid"
```

Running the probe as a Windows service

The Windows version of the probe can be run as a Windows service. Configuring the probe to run as a Windows service is a two-part process: First you need to register the probe as a Windows service, then you need to start the probe using the Services window within Windows Control Panel.

Setting the path to jvm.dll in the Windows environment

Before running any probe as a Windows service, you must have the path to `jvm.dll` set in the probe environment. This file forms a part of your Java installation and its location depends on the version of Java that you are running.

If you want to use IBM Java that is supplied with Netcool/OMNIbus V7.4.0, enter the following commands on the command line:

```
set OMNIBUS_JVM_DLL=C:\IBM\Tivoli\Netcool\platform\win32\jre_1.6.7\jre\bin\j9vm
\jvm.dll
```

If you want to use another version of Java (for example, Sun Oracle Java) you must set the path to the location of the `jvm.dll` file within that Java environment. To identify the path to set, search for `jvm.dll` using Windows Explorer or consult the documentation supplied with your version of Java.

Note : If you do not set `OMNIBUS_JVM_DLL`, the `nco_p_jdbc.bat` script will use the default `jvm.dll` delivered with Netcool/OMNIbus in the following location `%NCHOME%\platform\`*`arch`*`\jre_`*`version`*.

Registering the probe as a Windows service and running the probe

To register the probe as a Windows service, run the following command:

`%OMNIHOME%\probes\win32\`*`probe_name`* `/INSTALL /CMDLINE "`*`command_line_options`*`"`

Where *`probe_name`* is the name of the probe, for example `nco_p_jdbc.bat`.

Note : You must include double quotes ("") after `/CMDLINE` for the command line arguments, otherwise the Windows service will not work as expected.

To run the probe as a Windows service, use the following steps:

1. Configure the probe properties file.
2. Select **Control Panel** > **Administrative Tools**.
3. Double-click on **Services**.

   The **Services** window opens. This window lists all of the services that are currently installed on your machine.
4. Search for the probe by its name in the list of services.

   Note : If you did not specify an instance name within the command line options, the probe instance will appear in the list of services as **NCONcoPJdbcProbe**.
5. Click on its name and select **Start**.

To remove the probe service, run the following command:

`%OMNIHOME%\probes\win32\`*`probe_name`* `/REMOVE`

Running multiple instances of the probe as a Windows service

When you run multiple instances of the probe as a Windows service, you should use separate properties files and rules file. You should also specify different files for the **`PidFile`** and **`DataBackupFile`** properties.

To create a second JDBC probe instance as windows service, use one of the following methods:

Method 1

1. Make a copy of the `jdbc.props` and `jdbc.rules` files naming them `JDBCProbe2.props` file and a `JDBCProbe2.rules` respectively.
2. Edit the **rulesfile** property of the `JDBCProbe2.props` file to reference `JDBCProbe2.rules`.
3. Run the following command:

   ```
   nco_p_jdbc.bat /INSTALL /INSTANCE JDBCProbe2 /CMDLINE "-name JDBCProbe2"
   ```

Note : The `-name JDBCProbe2` option determines that the files `JDBCProbe2.props` and `JDBCProbe2.rules` will be used and that the instance of the probe will be called `JDBCProbe2`; the probe instance will appear in the list of services as **JDBCProbe2**.

Method 2

1. Make a copy of the `jdbc.props` and `jdbc.rules` files naming them `jdbc2.props` file and a `jdbc2.rules` respectively.
2. Edit the **rulesfile** property of the `jdbc2.props` file to reference `jdbc2.rules`.
3. Run the following command:

   ```
   nco_p_jdbc.bat /INSTALL /INSTANCE JDBCProbe2 /CMDLINE "-propsfile
   jdbc2.props"
   ```

Removing a running instance of the probe

To remove a probe instance, run the following command:

`nco_p_jdbc.bat /REMOVE /INSTANCE` *`instance_name`*

For example:

```
nco_p_jdbc.bat /REMOVE /INSTANCE JDBCProbe2
```

Running multiple instances of the JDBC Probe

You can run multiple instances of the probe on a single machine. This allows you to retrieve events from more than one event source.

When running multiple instances on a single host, you should specify unique values for the following properties so that the individual instances do not attempt to overwrite each others files:

- **PidFile** - this file stores the process ID of the probe.
- **PropsFile** - this is the properties file that the probe uses.
- **DataBackupFile** - this is the file the probe uses to record which events have already been retrieved.

You should also specify a different rules file for each instance using the **RulesFile** property. For each probe instance, you should customize the `@Summary` and `@Identifier` fields of its rules file so that you can differentiate between the events from the various data sources.

For example, suppose you have name two probe instances: `master1` and `master2`. If you change the `@Summary` and `@Identifier` fields in the rules file for `master1` to the following values:

```
@Summary = "master 1 == " + $AlertName + ....
@Identifier = "master1" + $AlertID + ..
```

and change the `@Summary` and `@Identifier` fields in the rules file for `master2` to the following values:

```
@Summary = "master 2 == " + $AlertName + ....
@Identifier = "master2" + $AlertID + ..
```

The probe will precede all alarms acquired by `master1` with `master 1` and will precede alarms acquired by `master2` with `master 2`.

Properties and command line options

You use properties to specify how the probe interacts with the device. You can override the default values by using the properties file or the command line options.

The following table describes the properties and command line options specific to this probe. For information about default properties and command line options, see the *IBM Tivoli Netcool/OMNIbus Probe and Gateway Guide*.

Table 5. Properties and command line options

Property name	Command line option	Description
DBPassword *string*	`-dbpassword` *string*	Use this property to specify the password associated with the **DBUsername** that the probe uses to log into the source database. The default is `" "`.
DBUsername *string*	`-dbusername` *string*	Use this property to specify the user name that the probe uses to log into the source database. The default is `" "`.
DisablePidFileLock *string*	`-disablepidfilelock` *string*	If you are running the probe on Windows, set this property to `true` to instruct the probe to overwrite any existing PID file when the probe starts. If you are running the probe on Windows and set this property to `false`, and if a PID file already exits for this probe in the `%OMNIHOME%\var` directory, the probe will fail to start. The default is `false`.
JdbcDriver *string*	`-jdbcdriver` *string*	Use this property to specify the JDBC driver that the probe uses to connect to the event source. The default is `" "`.
JdbcUrl *string*	`-jdbcurl` *string*	Use this property to specify the URL of the target database. The default is `" "`. **Note :** When you specify the URL for the database, you must include the details of both the IP address and the database to which you are connecting. The format of the URL that you specify for this property depends on the type of database that you are using. You must consult the documentation supplied with your database for details of the format that you need to use for its URL.

Table 5. Properties and command line options (continued)

Property name	Command line option	Description
MarkerColumn *string*	`-markercolumn` *string*	Use this property to specify the name of the column in the database that acts as a marker for partial resynchronization. The default is `""`.
MarkerColumnSensitive *string*	`-markercolumnsensitive` *string*	Use this property to specify whether the criteria specified by the mandatory SELECT statement is treated as case-sensitive when used to select data from the marker column. The default is `false` (which indicates that the criteria specified is case-insensitive). **Note :** In most situations, you should set this property to `false`. However, some databases (for example, Sybase) select data using case-sensitive criteria. For such databases, set this property to `true`.
PostSqlFile *string*	`-postsqlfile` *string*	Use this property to specify the file name of SQL statement file used to perform a post-SQL action after the mandatory SELECT statement. The default is `""`.
PreSqlFile *string*	`-presqlfile` *string*	Use this property to specify the file name of the SQL statement file used to perform a pre-SQL action before the mandatory SELECT statement. The default is `""`.
ResyncBatchSize *integer*	`-resyncbatchsize` *integer*	Use this property to specify the maximum number of alarms retrieved in each batch of resynchronization alarms. The default is `100`.
SelectSqlFile *string*	`-selectsqlfile` *string*	Use this property to specify the file name of SQL statement file used to perform the mandatory SELECT statement to retrieve events from the data source. The default is `""`.

Table 5. Properties and command line options (continued)		
Property name	**Command line option**	**Description**
SqlWarnings *string*	`-sqlwarnings` *string*	Use this property to specify whether the probe outputs non-critical database specific warnings to the log file. This property takes the following values: • false: The probe does not output non-critical database specific warnings to the log file. • true: The probe outputs non-critical database specific warnings to the log file. The default is `true`.

Properties and command line options provided by the Java Probe Integration Library (probe-sdk-java) version 4.0

All probes can be configured by a combination of generic properties and properties specific to the probe.

The following table describes the properties and command line options that are provided by the Java Probe Integration Library (probe-sdk-java) version 4.0.

Note : Some of the properties listed may not be applicable to your probe.

Table 6. Properties and command line options		
Property name	**Command line option**	**Description**
CommandPort *integer*	`-commandport` *integer*	Use this property to specify the port to which users can Telnet to communicate with the probe using the Command Line Interface (CLI) supplied. The default is `6970`.
CommandPortLimit *integer*	`-commandportlimit` *integer*	Use this property to specify the maximum number of Telnet connections that can be made to the probe. The default is `10`.
DataBackupFile *string*	`-databackupfile` *string*	Use this property to specify the path to the file that stores data between probe sessions. The default is `""`. **Note :** Specify the path relative to `$OMNIHOME/var`.

Table 6. Properties and command line options (continued)		
Property name	**Command line option**	**Description**
HeartbeatInterval *integer*	`-heartbeatinterval` *integer*	Use this property to specify the frequency (in seconds) with which the probe checks the status of the host server. The default is `60`.
Inactivity *integer*	`-inactivity` *integer*	Use this property to specify the length of time (in seconds) that the probe allows the port to receive no incoming data before disconnecting. The default is `0` (which instructs the probe to not disconnect during periods of inactivity).
InitialResync *string*	`-initialresync` *string*	Use this property to specify whether the probe requests all active alarms from the host server on startup. This property takes the following values: `false`: The probe does not request resynchronization on startup. `true`: The probe requests resynchronization on startup. For most probes, the default value for this property is `false`. If you are running the JDBC Probe, the default value for the **InitialResync** property is `true`. This is because the JDBC Probe only acquires data using the resynchronization process.
MaxEventQueueSize *integer*	`-maxeventqueue size`*integer*	Use this property to specify the maximum number of events that can be queued between the non native process and the ObjectServer. The default is `10000`. **Note :** You can increase this number to increase the event throughput when a large number of events is generated.

Table 6. Properties and command line options (continued)

Property name	Command line option	Description
ResyncInterval *integer*	`-resyncinterval` *integer*	Use this property to specify the interval (in seconds) at which the probe makes successive resynchronization requests. For most probes, the default value for this property is `0` (which instructs the probe to not make successive resynchronization requests). If you are running the JDBC Probe, the default value for the **ResyncInterval** property is `60`. This is because the JDBC Probe only acquires data using the resynchronization process.
RetryCount *integer*	`-retrycount` *integer*	Use this property to specify how many times the probe attempts to retry a connection before shutting down. The default is `0` (which instructs the probe to not retry the connection).
RetryInterval *integer*	`-retryinterval` *integer*	Use this property to specify the length of time (in seconds) that the probe waits between successive connection attempts to the target system. The default is `0` (which instructs the probe to use an exponentially increasing period between successive connection attempts, for example, the probe will wait for 1 second, then 2 seconds, then 4 seconds, and so forth).
RotateEndpoint *string*	`-rotateendpoint` *string*	Use this property to specify whether the probe attempts to connect to another endpoint if the connection to the first endpoint fails. This property takes the following values: false: The probe does not attempt to connect to another endpoint if the connection to the first endpoint fails. true: The probe attempts to connect to another endpoint if the connection to the first endpoint fails. The default is `false`.

Elements

The probe breaks event data down into tokens and parses them into elements. Elements are used to assign values to ObjectServer fields; the field values contain the event details in a form that the ObjectServer understands.

The elements that the probe generates depend on the source table from which the probe is retrieving data. The probe generates the element names from the columns names of that source table. Each field from each row of the source table becomes an element in the alarm. So the elements that the probe generates will differ from source table to source table.

Error messages

Error messages provide information about problems that occur while running the probe. You can use the information that they contain to resolve such problems.

The following table describes the error messages specific to this probe. For information about generic Netcool/OMNIbus error messages, see the *IBM Tivoli Netcool/OMNIbus Probe and Gateway Guide*.

Table 7. Error messages

Error	Description	Action
`Probe is connecting to JDBC data source`	The probe is attempting to connect to the data source using JDBC with the credentials specified by the **`DBUsername`**, **`DBPassword`**, **`JdbcUrl`**, and **`JdbcDriver`** properties.	The probe was started by the user and is now connecting to the data source.
`Probe is disconnecting from JDBC data source`	Probe is attempting to disconnect from the data source.	The shutting down process was initiated.
`Probe is shutting down ...`	The probe is shutting down.	The shutting down process was initiated.
`JDBC probe is an accessor probe. Please configure either InitialResync or ResyncInterval to enable the probe to be run and receiving data.`	The probe could not acquire data from the data source because the **`InitialResync`** property is set to `false` and the **`ResyncInterval`** property is set to `0`.	Either set the **`InitialResync`** property to `true` or set the **`ResyncInterval`** property to a value greater than `0`.
`SelectSqlFile is empty or not configured. Please configure SelectSqlFile to enable the probe to be run and receiving data.`	The **`SelectSqlFile`** property has either been omitted or the file specified does not contain a valid SQL SELECT statement.	Set the **`SelectSqlFile`** property to a file containing an SQL SELECT statement that retrieves events from the required table in the data source.
`Could not read the contents of sql file`	The probe failed to read the SQL file specified by the **`SelectSqlFile`** property.	Check that the file specified by the **`SelectSqlFile`** property is not corrupt. Check also that you have specified the correct path and file name.

Table 7. Error messages (continued)

Error	Description	Action
`SQL file not found or unable to open`	The probe failed to read the SQL file specified by the **SelectSqlFile** property.	Check the path of the file specified by the **SelectSqlFile** property and check that the file specified is not corrupt.
`Probe is unable to connect to JDBC data source`	The probe attempted to connect to the JDBC data source, but failed.	Check that you have set the **JdbcDriver**, **JdbcUrl**, **DBUsername**, and **DBPassword** properties correctly. See "Connecting to an event source using JDBC" on page 4.
`Exception occured during retrieve probe property from service provider.`	The probe failed to load property services from the OIDK library.	Check your probe environment to ensure that your property file has been configured correctly. If the problem persists, contact IBM software support.
`Data backup file features requires to work pairly with MarkerColumn property. Please refer to the References Guide on how to set the MarkerColumn & DataBackupFile accordingly`	The data recovery and backup functionality has not been configured correctly using the **DataBackupFile** and **MarkerColumn** properties. Both properties need to be set correctly for the probe to be able to perform partial resynchronization.	Check the values set for the **DataBackupFile** and **MarkerColumn** properties See "Configuring partial resynchronization" on page 13.
`Probe may not able to perform partial resync due to could not find the matchable marker column. Please ensure the column name is existed in both MarkerColumn property and the mandatory query.`	The probe could not find the matchable marker column specified by the **MarkerColumn** property within the mandatory select query result. The marker column name must exist in, and be in the same case as, the results returned by the select query specified by the **SelectSqlFile** property.	Check the values set for the **MarkerColumn** property and **SelectSqlFile** property. See "Configuring partial resynchronization" on page 13.
`Please double check your query and make sure the column name is correct`	The marker column specified by the **MarkerColumn** property is not included in the results returned by the mandatory SQL SELECT statement.	Check the query specified by the **SelectSqlFile** property and make sure that the column name is correct. See "Configuring partial resynchronization" on page 13.

Table 7. Error messages (continued)

Error	Description	Action
`MarkerColumnSensitive property has been set to true. Please ensure the column name defined in MarkerColumn is the same with the result return by select query with case-sensitive.`	The **MarkerColumnSensitive** property has been set to `true`.	Check that the column name specified by the **MarkerColumn** property is the same as the result returned by, and in the same case as, the SELECT statement in the file specified by the **SelectSqlFile** property. See "Acquiring data from case-insensitive and case-sensitive databases" on page 16.
`MarkerColumnSensitive property value is true. Please set it to false if you required the marker column to be compared in ignored-case (case-insensitive).`	The value of the **MarkerColumnSensitive** property has been set to `true` but the probe could not find the matched marker column.	You may need to set the **MarkerColumnSensitive** property it to `false` if the database operates in a case-insensitive environment. See "Acquiring data from case-insensitive and case-sensitive databases" on page 16.
`Probe unable to find the JDBC Driver class, please set JdbcDriver property and your environment CLASSPATH.`	The probe could not find the JDBC driver class specified by the **JdbcDriver** property.	Check that the **JdbcDriver** property is set to the path to the JDBC drivers and set the $CLASSPATH or %CLASSPATH% environment variable to include the path to the JDBC drivers. See "Connecting to an event source using JDBC" on page 4.
`Probe encountered SQL exception during pre-sql process. Please check your pre-sql query configured in PreSqlFile property.`	Probe has encountered an SQL exception during the pre-SQL selection processing.	Check the pre-SQL query specified by the **PreSqlFile** property.
`Probe encountered SQL exception during post-sql process. Please check your post-sql query configured in PostSqlFile property.`	Probe has encountered an SQL exception during the post-SQL selection processing.	Check the post-SQL query specified by the **PostSqlFile** property.
`Probe will skip the pre-sql process due to the file is blank`	The probe skipped the pre-sql process because the file specified by the **PreSqlFile** property is blank.	Change the value of the **PreSqlFile** property to specify a valid pre-SQL query.
`Probe will skip the post-sql process due to the file is blank`	The probe skipped the post-sql process because the file specified by the **PostSqlFile** property is blank.	Change the value of the **PostSqlFile** property to specify a valid post-SQL query.

ProbeWatch messages

During normal operations, the probe generates ProbeWatch messages and sends them to the ObjectServer. These messages tell the ObjectServer how the probe is running.

The following table describes the ProbeWatch messages that the probe generates. For information about generic Netcool/OMNIbus ProbeWatch messages, see the *IBM Tivoli Netcool/OMNIbus Probe and Gateway Guide*.

Table 8. ProbeWatch messages

ProbeWatch message	Description	Triggers or causes
`Running`	The probe is running normally.	The probe was started by the user.
`Unable to execute pre-SQL query`	The probe encountered errors and failed to process the pre-SQL query statements.	The execution of the pre-SQL query specified by the **`PreSqlFile`** property contains errors.
`Unable to execute post-SQL query`	The probe encountered erros and failed to process the post-SQL query statements.	The execution of the post-SQL query specified by the **`PostSqlFile`** property contains errors.
`Unable to load jdbc driver class`	The probe encountered errors when loading the JDBC driver class.	Either the `CLASSPATH` environment variable is missing or incorrect, or the path to the JDBC driver jars file has not been set.
`Unable to get events during resynchronization`	The probe encountered errors during the resynchronization process and unable to retrieve events from the data source.	The probe could not run the resynchronization correctly. Ensure that the connection related properties and the mandatory select query property are correctly configured.

Appendix A. Notices and Trademarks

This appendix contains the following sections:

- Notices
- Trademarks

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing 2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who want to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Software Interoperability Coordinator, Department 49XA

3605 Highway 52 N
Rochester, MN 55901
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, ibm.com, AIX, Tivoli, zSeries, and Netcool are trademarks of International Business Machines Corporation in the United States, other countries, or both.

Adobe, Acrobat, Portable Document Format (PDF), PostScript, and all Adobe-based trademarks are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

Intel, Intel Inside (logos), MMX, and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

SC27-5610-02