IBM® Tivoli® Netcool/OMNIbus Probe for CA Spectrum (CORBA)
1.0

Reference Guide
July 20, 2017

SC27-8702-02

Notice

Before using this information and the product it supports, read the information in Appendix A, “Notices and Trademarks,” on page 43.

Edition notice

This edition (SC27-8702-02) applies to version 1.0 of IBM Tivoli Netcool/OMNIbus Probe for CA Spectrum (CORBA) and to all subsequent releases and modifications until otherwise indicated in new editions.

This edition replaces SC27-8702-01.

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.

The IBM Tivoli Netcool/OMNIbus Probe for CA Spectrum (CORBA) documentation is provided in softcopy format only. To obtain the most recent version, visit the IBM® Tivoli® Knowledge Center:

https://www.ibm.com/support/knowledgecenter/SSSHTQ/omnibus/probes/common/Probes.html

Table 1. Document modification history

Document version	Publication date	Comments
SC27-8072-00	December 10, 2015	First IBM publication. Version 1.0 of the probe supports CA Spectrum version 9.4.
SC27-8072-01	March 10, 2016	Support extended to CA Spectrum version 10.0. Summary updated on page 2. This probe addresses the following Enhancement Requests: **RFE 82141:** Enhancement made to allow the CA Spectrum probe to work with Spectrum V10.
SC27-8072-02	July 20, 2017	The Command Line Interface section has been updated to remove references to earlier versions of Netcool/OMNIbus. This guide was updated to address APAR IV95828.

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 CA Spectrum V9.4 (CORBA)

The CA Spectrum element management system (EMS) uses SpectroSERVERs to collect, process, and store information about managed network elements.

The Probe for CA Spectrum (CORBA) collects alarm information from one or more SpectroSERVERs using a proprietary Common Object Request Broker Architecture (CORBA) interface.

The Probe for CA Spectrum (CORBA) can receive events from CA Spectrum version 9.4 onwards. It is only supported on Netcool/OMNIbus V8.1 and above. If you want to receive events from CA Spectrum versions 9.0 to 9.3, you must use the Probe for CA Spectrum V9 (CORBA). For details about accessing this version of the probe See https://www-304.ibm.com/support/docview.wss?uid=swg21641820.

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.

The following table provides a summary of the Probe for CA Spectrum (CORBA).

Table 3. Summary

Probe target	CA Spectrum version 9.4 CA Spectrum version 10.0
Probe executable name	`nco_p_ca_spectrum_corba`
Probe installation package	`omnibus-`*`arch`*`-probe-nco-p-ca-spectrum-corba-`*`version`*

Table 3. Summary (continued)	
Package version	1.0
Probe supported on	For details of supported operating systems, see the following Release Notice on the IBM Software Support website: http://www-01.ibm.com/support/docview.wss?uid=swg21970414
Properties file	`$OMNIHOME/probes/arch/ca_spectrum_corba.props`
Rules file	`$OMNIHOME/probes/arch/ca_spectrum_corba.rules`
Host configuration file	`$OMNIHOME/probes/arch/ca_spectrum_corba_host.xml`

Additional configuration files	`ca_spectrum_corba.xsd`
	This is an XML schema file the probe uses to validate the format of the host configuration file.
	`SpectrumCause.pl`
	This script used to generate probable cause lookup files.

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	CORBA
Remote connectivity	The probe can connect to a remote device using a CORBA interface.
Multicultural support	Not Available
Peer-to-peer failover functionality	Available
IP environment	IPv4 and IPv6
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*.

Setting environment variables

Environment variables are specific preset values that establish the working environment of the probe.

The following table indicates the minimum Java requirement to run the probe.

Table 4. Java version required by probe

Spectrum Version	Java Version	Compatibility
Spectrum Version 9.4	Java 7	JRE 7 available with OMNIbus 8.1
Spectrum Version 10.0	Java 8	Omnibus 8.1 is not available with JRE 8.

The selection of the Java Runtime Environment (JRE) used for the probe's runtime environment adheres to the following order of precedence:

1. The JRE specified by the $NCO_PROBE_JRE environment variable.
2. The JRE specified by the $JAVA_HOME environment variable.
3. The most recent JRE version available to Netcool/OMNIbus.

Note : If the version of Netcool/OMNIbus that you are using did not come with the minimum required Java version, you must perform one of the following actions before running the probe:

1. Download and install the required version of JRE from the Oracle download page.
2. Edit the `$OMNIHOME/probes/java/nco_p_ca_spectrum_corba.env` file to add the latest Java directory to the $NCO_PROBE_JRE environment variable.

Installing Spectrum utilities

The probe requires access to a number of Java utilities that are supplied with your CA Spectrum installation.

Copy the following files from `$SPECROOT/lib` to `$OMNIHOME/probes/java/nco_p_ca_spectrum_corba`:

- `cryptoJFIPS.jar`

- global*xx*.jar
- lm.jar
- sanct6.jar
- ssorb*xx*.jar
- ssortbutil*xx*.jar
- util*xx*.jar
- utilapp*xx*.jar
- utilnet*xx*.jar
- utilsrv*xx*.jar
- vbhelpher*xx*.jar
- vbjorb.jar
- vbsec.jar

Where $SPECROOT is the directory where CA Spectrum is installed and *xx* is the version of CA Spectrum that you are running.

Note : You will have to create the nco_p_ca_spectrum_corba folder manually.

For probe installations on Linux operating systems, you must also copy the following files to $OMNIHOME/probes/java/nco_p_ca_spectrum_corba:

- sanct6.jar
- sanctuary.jar

Firewall considerations

When using CORBA probes in conjunction with a firewall, the firewall must be configured so that the probe can connect to the target system.

Most CORBA probes can act as both a server (listening for connections from the target system) and a client (connecting to the port on the target system to which the system writes events). If you are using the probe in conjunction with a firewall, you must add the appropriate firewall rules to enable this dual behavior.

There are three possible firewall protection scenarios, for which you must determine port numbers before adding firewall rules:

1. If the host on which the probe is running is behind a firewall, you must determine what remote host and port number the probe will connect to.
2. If the host on which the target system is running is behind a firewall, you must determine the incoming port on which the probe will listen and to which the target system will connect.
3. If each host is secured with its own firewall, you must determine the following four ports:
   a. The outgoing port (or port range) for the probe.
   b. The hostname and port of the target system.
   c. The outgoing port on which the target system sends events if the probe is running as a client.
   d. The incoming port on which the probe listens for incoming events.

Note : Most, but not all, CORBA probes listen on the port specified by the **ORBLocalPort** property. The default value for this property is 0, which means that an available port is selected at random. If the probe is behind a firewall, the value of the **ORBLocalPort** property must be specified as a fixed port number.

CORBA probes that use EventManager or NotificationManager objects may use different hosts and ports from those that use NamingService and EntryPoint objects. If the probe is configured to get object references from a NamingService or EntryPoint object, you must obtain the host and port information from the system administrator of the target system. When you have this information, you can add the appropriate firewall rules.

Configuring SpectroSERVERs

You must configure the SpectroSERVERs that you are using to enable them to accept CORBA connections from the probe.

Configuring a SpectroSERVER to accept connections from the probe involves the following tasks:

1. Using the Spectrum Control Panel (SCP) or OneClick add the details of the probe host system to the SpectroSERVER's server list.

   Note : If you add a plus sign (+) to the sever list, all hosts will be able to connect to the SpectroSERVER.
2. Create a Spectrum User Model for the user ID that runs the probe.
3. Add the user ID to the associated UserList.

For information about how to configure your SpectroSERVERs, see the following CA Spectrum guides:

- *CORBA API Programmers Guide* (5010.pdf)
- *Installation Guide* (5136.pdf)

For more information about the Spectrum Control Panel (SCP), see *Control Panel User Guide* (5029.pdf).

Configuring the probe

Global properties that affect all monitored SpectroSERVERs are specified using the properties file `ca_spectrum_corba.props`. The details of each monitored SpectroSERVER are specified using host-specific properties in the host configuration file `ca_spectrum_corba_host.xml`.

Configuration options are described in the following topics:

- "Using the host configuration file" on page 5
- "Using lookup tables" on page 9
- "Configuring ObjectServer host properties" on page 11

Using the host configuration file

You must configure at least one SpectroSERVER in the host configuration file supplied with the probe.

The host configuration file `ca_spectrum_corba_host.xml` is an XML file that contains the details of each SpectroSERVER host that you want the probe to monitor. It is installed to the following directory:

`$OMNIHOME/probes/arch/`

Use the **`HostFile`** property to specify the path to the host configuration file.

In addition to specifying the details of each SpectroSERVER host, you can also use the host configuration file to specify how the probe handles the alarms it retrieves from each SpectroSERVER. The host configuration properties are XML elements, so they cannot be accessed using the command line interface, but you can edit the file in any text editor. The probe uses the XML schema file `ca_spectrum_corba.xsd` to validate the host configuration file.

The following table lists the available configuration properties contained in the host configuration file.

Table 5. Host configuration file properties

Property	Description

`<SpectroServer Name="" FQDN="" IP="">`
`...`
`</SpectroServer>`
	Use this property to specify the name, FQDN, and IP address of the SpectroSERVER host machine.
	Note : You must specify both the `Name` and the `Domain` value in lowercase.

Table 5. Host configuration file properties (continued)	
Property	**Description**
`<ModelAttributes Value="" />`	Use this property to specify a comma-separated list of model attributes that the probe should extract in addition to the default attributes. You can specify alarm attributes by their identifiers (base 10 or base 16) or by their unique canonical names as specified by the CA Spectrum type catalog. Retrieved attributes become tokens that are available to the rules file using their canonical names. The following example creates the tokens `$Security_String` and `$Notes`: `<ModelAttributes Value="0x10009, 0x11564" />` You can specify a maximum of one `ModelAttributes` property for each SpectroSERVER.
`<AlarmAttributes Value="" />`	Use this property to specify a comma-separated list of alarm attributes that the probe should extract in addition to the default attributes. You can specify alarm attributes by their identifiers (base 10 or base 16) or by their unique canonical names as specified by the CA Spectrum type catalog. Retrieved attributes become tokens that are available to the rules file using their canonical names. The following example creates the tokens `$Cleared_By_User_Name` and `$Occurrences`: `<AlarmAttributes Value= "Cleared_By_User_Name, 0x11fc5"/>` You can specify a maximum of one `AlarmAttributes` property for each SpectroSERVER.
`<TimeStampFile Value="" />`	Use this property to specify the location of the file in which the probe stores the timestamp of the last alarm it processed. For example: `<TimeStampFile Value="/opt/IBM/tivoli/netcool81/probes/solaris2/server2.timestamp" />` You can specify a maximum of one `TimeStampFile` property for each SpectroSERVER.
`<FetchEventFormatFields Value="false" />`	Use this property to specify whether the probe retrieves the event data fields attached to an alarm and makes them available to the rules file This property takes the following values: `true`: The probe retrieves the event data fields. `false`: The probe does not retrieve the event data fields. The default is `false`. Retrieved fields are made available to the rules file as tokens in the format `$Event_`*NAME*, where *NAME* is the canonical name of the attribute in the CA Spectrum type catalog. If an attribute has no entry in the type catalog, *NAME* is given as its identifier in hexadecimal format. You can specify a maximum of one `FetchEventFormatFields` property for each SpectroSERVER.

Table 5. Host configuration file properties (continued)	
Property	**Description**
<FetchEventString Value="false" />	Use this property to specify whether the probe creates event messages using the originating event field data attached to an alarm. This property takes the following values: `true`: The probe creates event messages. `false`: The probe does not creates event messages. The default is `false`. The event message is made available to the rules file as the token `$Event_Format_String`. This property requires that the `FetchEventFormatFields` property is set to `true` and that the `EventFormatFile` property contains a valid directory path. You can specify a maximum of one `FetchEventString` property for each SpectroSERVER.
<EventFormatFile Value="" />	Use this property to specify the local directory where the event format file that contains event data is located. The default event format file is typically located in the `$SPECROOT/SG-Support/CsEvFormat` directory (where $SPECROOT is the directory where CA Spectrum is installed). Custom event format files are typically located in the `$SPECROOT/custom/Events/CsEvFormat` directory. These event format files must be locally available on the host where the probe is installed. If the probe is running on a different host than CA Spectrum, you must copy the event format files to the host where the probe is installed. `EventFormatFile` should contain the directory path where all these event format files are kept. You can specify several instances of the `EventFormatFile` property for each SpectroSERVER. The probe reads each property in turn until it finds a match, so the order in which they are listed is important if you have duplicate event format files.
<FormatFileSuffix Value=""/>	Use this property to append locale information to an event. This property sets expected postfix appended to the name of Event File's in SpectroSERVER. Starting from Spectrum 9.3, event format files names in `$SPECROOT/SG-SUPPORT/CsEvFormat` are appended with locale information. For example: `Event01169988` changes to `Event01169988_en_US`

Table 5. Host configuration file properties (continued)	
Property	**Description**
`<ProbCauseLookupFile Value="" />`	Use this property to specify the location of the probable cause lookup file. For example: `<ProbCauseLookupFile Value="/opt/IBM/tivoli/netcool81/probes/solaris2/server1.lookup"/>` You can specify a maximum of one `ProbCauseLookupFile` property for each SpectroSERVER. You can generate probable cause lookup files using the `SpectrumCause.pl` script supplied with the probe. For more information, see "Using lookup tables" on page 9.
`<ResyncInterval Value="86400" />`	Use this property to specify the time interval (in seconds) at which the probe performs automatic resynchronizations. The default is `86400` seconds (24 hours). You can specify a maximum of one `ResyncInterval` property for each SpectroSERVER.
`<ResyncSQLCmd Value= "" />`	Use this property to specify an SQL command to update the ObjectServer during a resynchronization. The default is `UPDATE alerts.status SET Severity = 0 WHERE LastOccurrence &lt;= %ResyncTime AND Node = %SpectroServerName AND AlertKey NOT IN (%AlarmID_List)` You can specify a maximum of one `ResyncSQLCmd` property for each SpectroSERVER. For more information about this command, see "Resynchronization" on page 12. **Note :** The less-than sign (<) is a reserved character in XML. When used in values for this property, it must be replaced by the character entity "<". For example, in the SQL statement above, `LastOccurrence &lt;= %ResyncTime` represents `LastOccurrence <= %ResyncTime`. The XML character entity for the greater-than sign (>) is ">".

Example 1

The following example of a host configuration file configures two SpectroSERVERs for use with the probe. Alarms are handled according to the default settings on the SpectroSERVER hosts.

```
<SpectroServer Name="server1" FQDN="domain1" IP="server1.somedomain.com" />
<SpectroServer Name="server2" FQDN="domain2" IP="server2.somedomain.com" />
```

Example 2

The following example of a host configuration file configures two SpectroSERVERs and some of the alarm attributes.

```
<SpectroServer Name="server1" FQDN="domain1" IP="server1.somedomain.com">
    <ModelAttributes Value="0x10009,0x11564,0x23000e" />
  <AlarmAttributes Value="0x11f9b,0x002305b8" />
  <TimeStampFile Value="/opt/IBM/tivoli/netcool81/probes/
solaris2/server1.timestamp" />
  <FetchEventFormatFields Value="true" />
  <FetchEventString Value="true" />
  <EventFormatFile Value="/usr/SPECTRUM/SG-Support/CsEvFormat" />
  <EventFormatFile Value="/usr/SPECTRUM/custom/CsEvFormat" />
  <ProbCauseLookupFile Value="/opt/IBM/tivoli/netcool81/probes/
solaris2/server1.lookup" />
  <ResyncInterval Value="86400" />
  <ResyncSQLCmd Value="UPDATE alerts.status SET Severity =0 WHERE
                          LastOccurrence &lt;= %ResyncTime AND
                          Node = %SpectroServerName AND
                          AlertKey NOT IN (%AlarmID_List)" />
</SpectroServer>
```

```
<SpectroServer Name="server2" FQDN="domain2" IP="server2.somedomain.com">
      <ModelAttributes Value="0x10009,0x11564,0x23000e" />
      <TimeStampFile Value="/opt/IBM/tivoli/netcool81/probes/
solaris2/server2.timestamp"
/>
      <ProbCauseLookupFile Value="/opt/IBM/tivoli/netcool81/
probes/solaris2/server2.lookup"
/>
      <ResyncInterval Value="86400" />
      <ResyncSQLCmd Value="UPDATE alerts.status SET Severity =
0 WHERE
                          LastOccurrence &lt;= %ResyncTime AND
                          Node = %SpectroServerName AND
                          AlertKey NOT IN (%AlarmID_List)" />
</SpectroServer>
```

Using lookup tables

You can use the `SpectrumCause.pl` Perl script supplied with the probe to create lookup files. The lookup files contain tables that enable the rules file to match CA Spectrum probable cause descriptions to the alarms generated by the SpectroSERVER.

When you use lookup tables, the following tokens are generated in the rules file with data extracted from the CA Spectrum probable cause files:

- `$title`
- `$symptoms`
- `$prob_cause`
- `$recommended_actions`

To enable the probe to use the CA Spectrum probable cause files, use the following steps:

1. Run the `SpectrumCause.pl` script on the SpectroSERVER host, using the following command:

   ```
   perl SpectrumCause.pl spectrum_install_dir lookup_file_location
   ```

   Where *spectrum_install_dir* is the SpectroSERVER directory that contains the CA Spectrum probable cause files and *lookup_file_location* is the full directory path to the generated lookup file.
2. Copy the generated lookup file to the probe host, for example:

   `/opt/IBM/tivoli/netcool81/solaris2/server1.lookup`
3. If the probe is running, stop it.
4. Open the rules file `ca_spectrum_corba.rules` in a text editor.
5. Uncomment the following two lines in the rules file by removing the # character:

   ```
   #table spectrum_lookup_table="/etc/user/my.lookup"
   #default = {"unavailable", "unavailable", "unavailable", "unavailable"}
   ```

6. In the first line, replace the lookup table name with your own site-specific name and replace `/etc/user/my.lookup` with the directory path to the lookup file.

   For example:

   ```
   table
   server1_lookup_table="/opt/IBM/tivoli/netcool81/probes/solaris2/server1.lookup"
   default = {"unavailable", "unavailable", "unavailable", "unavailable"}
   ```

   If you are using multiple SpectroSERVERs, repeat this step for each lookup file that you are using. Each entry must have a unique lookup table name and lookup file name, and be followed by the `default` line.

   For example:

   ```
   table
   server1_lookup_table="/opt/IBM/tivoli/netcool81/probes/solaris2/server1.lookup"
   default = {"unavailable", "unavailable", "unavailable", "unavailable"}
   table
   server2_lookup_table="/opt/IBM/tivoli/netcool81/probes/solaris2/server2.lookup"
   default = {"unavailable", "unavailable", "unavailable", "unavailable"}
   ```

7. Uncomment the following lines:

   ```
           if (exists($CauseNum))
           {
               #
               # if (match(@Node, "your_spectrum_hostname_string"))
               # {
               #     [$title, $symptoms, $prob_cause, $recommended_actions] =
                                   lookup($CauseNum, spectrum_lookup_table)
               # }
           }
   ```

8. Replace *your_spectrum_hostname_string* with the name of the SpectroSERVER host as specified by the Name attribute of the `SpectroServer` property in the host configuration file `ca_spectrum_corba_host.xml`.
9. Replace *spectrum_lookup_table* with the lookup table name that you specified in Step 6.
10. If you are using multiple SpectroSERVERs, repeat Steps 7 to 9 for each lookup file that you are using and use an `else if` statement for each additional entry. For example:

    ```
    #Lookup table needs $CauseNum
           if (exists($CauseNum))
           {

               if (match(@Node, "spectroserver1"))
               {
                    [$title, $symptoms, $prob_cause, $recommended_actions] =
                    lookup($CauseNum, server1_lookup_table)
                }

               else if (match(@Node, "spectroserver2"))
               {
                    [$title, $symptoms, $prob_cause, $recommended_actions] =
                    lookup($CauseNum, server2_lookup_table)
                }
    ```

11. Save and close the rules file.
12. In the host configuration file, add the location of the lookup file to the `Value` attribute of the `ProbCauseLookupFile` property.
13. Restart the probe.

Configuring ObjectServer host properties

The probe uses a separate connection to the ObjectServer when it is performing a resynchronization. You must configure this connection in the `ca_spectrum_corba.properties` file.

To allow the probe to perform resynchronization operations with the ObjectServer, you must specify appropriate values for the following properties in the `ca_spectrum_corba.properties` file:

- **`OSHost`**: This property specifies the host name of the server on which the ObjectServer is running. It must have the same value as the ObjectServer host name specified in the `$NCHOME/etc/omni.dat` interfaces file.
- **`OSPassword`**: This property specifies the password that the probe uses to connect to the ObjectServer.
- **`OSPort`**: This property specifies the port through which the probe connects to the ObjectServer. It must have the same value as the ObjectServer port specified in the `$NCHOME/etc/omni.dat` interfaces file.
- **`OSUserName`**: This property specifies the user name used to connect to the ObjectServer.
- **`Server`**: This is a generic Netcool/OMNIbus property that specifies the name of the ObjectServer host.

Running the probe

Probes can be run in a variety of ways. The way you chose depends on a number of factors, including your operating system, your environment, and the any high availability considerations that you may have.

For details about how to run the probe, visit the following page on the IBM Tivoli Knowledge Center:

http://www-01.ibm.com/support/knowledgecenter/SSSHTQ/omnibus/probes/all_probes/wip/concept/running_probe.html

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.

The Probe for CA Spectrum (CORBA) gathers events from SpectroSERVERs using a proprietary Common Object Request Broker Architecture (CORBA) interface.

When the probe starts, it connects to a SpectroSERVER using the details specified by the `SpectroServer` property in the host configuration file (`$OMNIHOME/probes/`*`arch`*`/ca_spectrum_corba_host.xml`). On connection, the probe listens for new alarms from the SpectroSERVER.

The probe then initializes the CORBA status monitor to monitor the status of the CORBA services. The probe checks the status of the CORBA services at intervals specified by the **`SpectroServerPollInterval`** property.

Initially, the probe retrieves a list of all active alarms from the SpectroSERVER. On subsequent connections, the probe retrieves all active alarms if the **`InitialResync`** property is enabled (set to `true`). If the **`InitialResync`** property is disabled (set to `false`), the probe only retrieves alarms that were generated while the probe was disconnected.

When the **`InitialResync`** property is disabled (the default setting), you must use the `TimeStampFile` property in the host configuration file to specify a timestamp file. This prevents the probe from performing a full resynchronization every time it starts up.

If you are running two SpectroSERVERs in a failover configuration, the probe indicates in the log file whether it is connected to the primary SpectroSERVER or the secondary SpectroSERVER.

Data acquisition is further described in the following topics:

- "Configuring the event details extraction buffering mechanism" on page 12
- "Specifying the event extraction method that the probe uses" on page 12
- "Resynchronization" on page 12

- “Filtering alarms” on page 14
- “Backoff strategy” on page 14
- “Command line interface” on page 16
- “Peer-to-peer failover functionality” on page 15

Configuring the event details extraction buffering mechanism

If you set the <FetchEventFormatFields> XML property to true, when the probe receives an alarm from the SpectroSERVER, it makes a call to Archive Manager to retrieve non-OE event fields attached to the alarm. Whatever alarms the probe is listening to, it first writes them to a buffer, then starts processing the alarms one event at a time. Once completed, the probe will take another alarm from the buffer to process, and so forth.

The **EventRetryCount** and **EventRetryTimeout** properties specify how the probe retries the extraction of non-OE events from the Archive Manager if the Archive Manager service is down. Use the **EventRetryCount** property to specify how may times the probe attempts to extract event details from Archive Manager before moving on to the next alarm. Use the **EventRetryTimeout** property to specify how long the probe waits to receive the event details before timing out and retrying the event extraction request.

Specifying the event extraction method that the probe uses

The probe can either extract all events attached to an alarm (which requires making an additional call to Archive Manager) or it can extract details of just the Originating Event.

You can specify which method the probe uses to extract details of the events attached to an alarm by setting the **EventExtraction** property to one of the following values:

- AM: Using this method, the probe can retrieve details of one or more events that raised the alarm. It requires the probe making an additional call to Archive Manager (AM).
- OE: Using this method, the probe extracts from the alarm details of the originating event that raised the alarm from the Originating Event (OE) byte stream. It does not require the probe making an additional call to Archive Manager and so avoids any delays that this may incur.

Note : You can run two SpectroSERVERs in a failover configuration, in which case the probe can be configured to failover to a secondary SpectroSERVER if the primary SpectroSERVER fails. When the probe is connected to the secondary SpectroSERVER, it will always use the OE method to extract details of events attached to an alarm regardless of setting of the **EventExtraction** property.

Resynchronization

If the probe loses the connection to a SpectroSERVER, there are three methods for resynchronizing cleared alarms in the SpectroSERVER with the ObjectServer. Resynchronization can be performed using the command line interface (CLI), the SpectroSERVER resynchronization interval, and during the SpectroSERVER failover process.

Note : Events that are deleted from the SpectroSERVER while the probe is disconnected remain in the ObjectServer and must be manually cleared. If the probe is using a timestamp file, some duplication of alarms may occur.

Resynchronization to the ObjectServer is performed using an SQL statement specified by the ResyncSQLCmd property in the host configuration file. You can configure the Value attribute of the ResyncSQLCmd property using the tokens listed in the following table.

Table 6. ResyncSQLCmd property tokens	
Token	**Description**
%AlarmID_List	The list of active alarm identifiers collected during resynchronization.
%Manager	The same value as the generic Netcool/OMNIbus **Manager** property.
%MinSpectrumSeverity	The same value as the **MinSpectrumSeverity** property.
%MaxSpectrumSeverity	The same value as the **MaxSpectrumSeverity** property.
%Name	The same value as the generic Netcool/OMNIbus **Name** property.
%OSHost	The same value as the **OSHost** property.
%OSPort	The same value as the **OSPort** property.
%ResyncTime	The time at which the probe receives the requested active alarm list.
%Server	The same value as the generic Netcool/OMNIbus **Server** property.
%ServerBackup	The same value as the generic Netcool/OMNIbus **ServerBackup** property.
%SpectroServerName	The name of the SpectroSERVER host as specified by the Name attribute of the SpectroServer property in the host configuration file.
%SpectroServerDomain	The domain of the SpectroSERVER host as specified by the Domain attribute of the SpectroServer property in the host configuration file.
%SpectroServerIP	The IP address of the SpectroSERVER host as specified by the IP attribute of the SpectroServer property in the host configuration file.

Resynchronization using the CLI

You can perform a manual resynchronization using the **resync** *spectrumName* command, where *spectrumName* is the name of the SpectroSERVER host as specified by the Name attribute of the SpectroServer property in the host configuration file. For more information about using the CLI, see "Command line interface" on page 16.

Using a resynchronization interval

You can use the ResyncInterval property in the host configuration file to set an automatic resynchronization at specified intervals. The default is 86400 seconds (24 hours). You can specify different resynchronization intervals for each monitored SpectroSERVER.

Using a SpectroSERVER failover configuration

If you are running two SpectroSERVERs in a failover configuration, you can use the **ResyncOnFailover** property to trigger a resynchronization operation when the probe fails over to the secondary SpectroSERVER. There is a possibility that the secondary SpectroSERVER might not contain all available alarms while the primary SpectroSERVER is down. To avoid data loss, you can use the **ResyncPrimaryOnly** property to specify that the probe resynchronizes with the primary SpectroSERVER only.

You can use the **ResyncThreadMax** property to specify the maximum number of threads that the probe spawns to perform the resynchronization operation. Specify a value of `0` to allow the probe to use an unlimited number of threads. The default is 5.

Filtering alarms

You can restrict the alarms that the probe retrieves to a range of CA Spectrum severity values. Alarm filtering is only available when the **InitialResync** property is disabled (set to `false`).

Use the **MinSpectrumSeverity** property to specify the minimum severity level threshold. The probe will not monitor alarms below this severity level. The default is `0` (CA Spectrum severity Normal).

Use the **MaxSpectrumSeverity** property to specify the maximum severity level threshold. The probe will not monitor alarms above this severity level. The default is 6 (CA Spectrum severity Initial).

The following table lists the available CA Spectrum severity levels:

Table 7. CA Spectrum severity levels

Severity level	Description
0	Normal
1	Minor
2	Major
3	Critical
4	Maintenance
5	Suppressed
6	Initial

Backoff strategy

The Probe for CA Spectrum (CORBA) is multi-headed. The probe can connect 1-to-1 or make multiple connections to multiple SpectroSERVERS.

The **Retry** strategy is two fold:

The probe's Host Connection Retry Strategy manages 1 or more connections to SpectroSERVERS of a probe instance.

The probe's Retry Strategy manages a probe's life cycle.

When all connections (heads) are disconnected and removed from the probe, the probe's Retry Strategy will take effect..

1. The probe's Host Connection Retry Strategy:

SpectrumRetry property enables a probe to wait for an existing connection to reestablish in the event of a connection loss due to an issue in the SpectroSERVER without the need to fully restart a probe.

When **SpectrumRetry** is set to `true`:

- An existing connection waits indefinitely for a connection to reestablish in case SpectroSERVER is restarted. When SpectrumRetry is set to false:

When **SpectrumRetry** is set to `false`.

When there are no more connections left in the probe, Probe's Retry Strategy as described below will take effect.

2. The Probe's Retry Strategy:

RetryCount and **RetryInterval** are responsible for managing the probe's retry and backoff strategy.

RetryCount refers to number of times the probe will attempt to retry its connection with SpectroSERVER until it is successful or stop attempting when **RetryCount** limit is reached.

RetryInterval determines the timing strategy used to reconnect the probe to SpectroSERVER.

a. **RetryInterval** `0` is set to causes the probe to use an exponential increment time in seconds until **RetryCount** is reached.

For example:

RetryCount = 5

RetryInterval = 5

RetryInterval is set to 5 the probe attempts reconnection every five seconds until the **RetryCount** limit is reached. (There are a total of 5 attempts and a 5 second waiting time before each attempt is triggered).

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.

Example property file settings for peer-to-peer failover

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.

Note : In the examples, make sure to use the full path for the property value. In other words replace `$OMNIHOME` with the full path. For example: `/opt/IBM/tivoli/netcool`.

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

```
Server      :    "NCOMS"
RulesFile   :    "master_rules_file"
MessageLog  :    "master_log_file"
PeerHost    :    "slave_hostname"
PeerPort    :    6789 # [communication port between master and slave probe]
Mode        :    "master"
PidFile     : "master_pid_file"
```

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

```
Server      :    "NCOMS"
RulesFile   :    "slave_rules_file"
MessageLog  :    "slave_log_file"
PeerHost    :    "master_hostname"
PeerPort    :    6789 # [communication port between master and slave probe]
Mode        :    "slave"
PidFile     : "slave_pid_file"
```

Command line interface

The probe is supplied with a command line interface (CLI) that allows you to manage the probe while it is running.

Managing the probe over an HTTP/HTTPS connection

IBM Tivoli Netcool/OMNIbus Version 7.4.0 (and later) includes a facility for managing the probe over an HTTP/HTTPS connection. This facility uses the **nco_http** utility supplied with Tivoli Netcool/OMNIbus.

The HTTP/HTTPS command interface replaces the Telnet-based command line interface used in previous version of IBM Tivoli Netcool/OMNIbus.

The following sections show:

- How to configure the command interface.
- The format of the **nco_http** command line.
- The format of the individual probe commands.
- The messages that appear in the log files.
- How to store frequently-used commands in a properties file.

For more information on the HTTP/HTTPS command interface and the utilities it uses, see the chapter on remotely administering probes in the *IBM Tivoli Netcool/OMNIbus Probe and Gateway Guide*.

Configuring the command interface

To configure the HTTP/HTTPS command interface, set the following properties in the `ca_spectrum_corba.props` property file:

NHttpd.EnableHTTP: Set this property to `true`.
NHttpd.ListeningPort: Set this property to `8080`.

Optionally, set a value for the following property as required:

NHttpd.ExpireTimeout: Set this property to the maximum time (in seconds) that the HTTP connection remains idle before it is disconnected.

The *IBM Tivoli Netcool/OMNIbus Probe and Gateway Guide* contains a full description of these and all properties for the HTTP/HTTPS command interface.

Format of the nco_http command line

The format of the **nco_http** command line to send a command to the probe is:

`$OMNIHOME/bin/nco_http -uri probeuri:probeport/probes/ca_spectrum_corba -datatype application/json -method post -data '{"command":"command-name","params":[command-parameters]}'`

Where:

- *probeuri* is the URI of the probe.
- *probeport* is the port that the probe uses to listen for HTTP/HTTPS commands. Specify the same value as that set for the **NHttp.ListeningPort**.

- *command-name* is the name of the command to send to the probe. The following command names are available:

  acknowlegeAlarm
  clearAlarm
  resync
  updateStatus
  updateTroubleShooterName
  updateTroubleShooterModel
  updateTroubleShooterTicket
  updateEventList
  unacknowlegeAlarm
- *command-parameters* is a list of zero or more command parameters. For commands that have no parameters, this component is empty. The command descriptions in the following section define the parameters that each takes.

Commands supported by the probe over HTTP/HTTPS

The following sections define the structure of the JavaScript Object Notation (JSON)-formatted commands that you can send to the probe. There is an example of each command.

Generic HTTP Command Structure

All the examples use a probe URI of `http://localhost` and a HTTP listening port of `8080`.

`/nco_http -uri http://localhost8080/probes/ca_spectrum_corba -datatype application/json` **Command Parameters**

Where the **Command Parameters** portion of the **Generic HTTP Command Structure** is substitute with samples from each command's description as follows:

acknowledgeAlarm

Use the **acknowledgeAlarm** command to acknowledge an alarm.

The format of the `-data` option for the **acknowledgeAlarm** command is:

```
-data '{"command":"acknowledgeAlarm", "params":[{"alarm_id":"alarmId",
"spectrumName":"MySpectrumServerName"}]}' -method POST
```

clearAlarm

Use the **clearAlarm** command to clear an alarm.

The format of the `-data` option for the **clearAlarm** command is:

```
-data '{"command":"clearAlarm", "params":[{"alarm_id":"alarmId",
"spectrumName":"MySpectrumServerName"}]}' -method POST
```

resync

Use the **resync** command to perform a complete resynchronization with the endpoint.

The format of the `-data` option for the **resync** command is:

```
-data '{"command":"resync", "params":
[{"spectrumName":"MySpectrumServerName"}]}' -method POST
```

unacknowledgeAlarm

Use the **unacknowledgeAlarm** command to unacknowledge an alarm.

The format of the `-data` option for the **unacknowledgeAlarm** command is:

```
-data '{"command":"unacknowledgeAlarm", "params":[{"alarm_id":"alarmId",
"spectrumName":"MySpectrumServerName"}]}' -method POST
```

updateEventList

Use the **updateEventList** command to event list of an alarm by specifying the alarm identifier, the event list identifier, and the SpectroSERVER host name.

The format of the `-data` option for the **updateEventList** command is:

```
-data '{"command":"updateEventList", "params":[{"alarm_id":"alarmId",
"eventIDList":"MyEventIDList","spectrumName:"MySpectrumName"}]}'
```

updateStatus

Use the **updateStatus** command to update the status of an alarm by specifying the alarm identifier, the new status, and the SpectroSERVER host name.

The format of the `-data` option for the **updateStatus** command is:

```
-data '{"command":"updateStatus", "params":[{"alarm_id":"alarmid",
"status":"MyStatus","spectrumName":"MySpectrumServerName"}]}' -method POST
```

updateTroubleShooterModel

Use the **updateTroubleShooterModel** command to update the troubleshooter type of an alarm by specifying the alarm identifier, the new troubleshooter type, and the SpectroSERVER host name.

The format of the `-data` option for the **updateTroubleShooterModel** command is:

```
-data '{"command":"updateTroubleShooterModel","params":[{"alarm_id"alarmId",
"name":"MyTroubleShooterName","spectrumName":"MySpectrumServerName"}]}' -method
POST
```

updateTroubleShooterName

Use the **updateTroubleShooterName** command to update the troubleshooter name of an alarm by specifying the alarm identifier, the new troubleshooter name, and the SpectroSERVER host name.

The format of the `-data` option for the **updateTroubleShooterName** command is:

```
-data '{"command":"updateTroubleShooterName","params":[{"alarm_id":"alarmId",
"name":"MyTroubleShooterName","spectrumName":"MySpectrumServerName"}]}' -method
POST
```

updateTroubleTicket

Use the **updateTroubleTicket** command to update the trouble ticket number for an alarm by specifying the alarm identifier, the new ticket identifier, and the SpectroSERVER host name.

The format of the `-data` option for the **updateTroubleTicket** command is:

```
-data '{"command":"updateTroubleTicket","params":["alarm_id":"alarmID",
"ticketID":"MyTicketID","spectrumName":"MySpectrumServerName"}]}' -method POST
```

Messages in the log file

The nco_http utility can make extensive entries in the probe's log file indicating the progress of each operation. These messages can help isolate problems with a request, such as a syntax problem in a command.

To obtain the detailed log information, set the probe's **MessageLevel** property to debug. This enables the logging of the additional information that tracks the progress of a command's execution. For example, the following shows the progress of a **resync** command:

```
Information: I-UNK-000-000: Calling NSProbeBidirCB:
Information: I-UNK-000-000: NSProbeBidirCNB: Thread id is 0x13e19a0
{command:resync,params:[{"spectrumName":SpectroServer1"]}
Debug: D-JPR-000-000: com.ibm.tivoli.netcool.omnibus.probe.services.impl.
SimpleCommandService$BidiThread.parseRequest ENTERING
Debug: D-JPR-000-000: com.ibm.tivoli.netcool.omnibus.probe.bidi.CommandHandler.executeCommand ENTERING
Debug: D-JPR-000-000: com.ibm.tivoli.netcool.omnibus.probe.bidi.CommandHandler.executeCommand EXITING
Information: I-JPR-000-000: Command Port received Resync request for Spectrum Server : SpectroServer1
Information: I-JPR-000-000: Attempting to subscribe Resync
request into queue for Spectrum Server : SpectroServer1
Information: I-JPR-000-000: Spectrum Server SpectroServer1 Successfully subscribed into Resync queue
Debug: D-NHT-105-008: [HTTP Listener]: Data available from 'localhost' (::1) on socket '16'.
Debug: D-NHT-105-028: [WorkPoolThread]: Successfully got a work item. Items left on work queue is '0'.
Debug: D-JPR-000-000: [ThreadMonitor] Thread added in  Monitor list.
Information: I-JPR-000-000: [ResyncWorker SpectroServer1] Thread started.
Information: I-UNK-000-000: Probewatch: Start resync for SpectroServer1
Information: I-JPR-000-000: [ResyncWorker SpectroServer1] Status: SUCCESS get active alarm list.
Debug: D-JPR-000-000: [ResyncWorker SpectroServer1] SQL command: UPDATE alerts.status
SET Severity = 0 WHERE LastOccurrence <= 1447057398 AND Node = 'SpectroServer1'
AND AlertKey NOT IN ('5620dec4-0029-1000-023b-008010b4d48f')
Debug: D-JPR-000-000: OmnibusDBManager executing sql command
Information: I-UNK-000-000: Probewatch: Complete resync for SpectroServer1
```

These messages can also help to isolate problems with a command. For example, the following shows the log messages for an unackAlarm command that contained an invalid alarm identifier.

```
Information: I-UNK-000-000: Calling NSProbeBidirCB
Information: I-UNK-000-000: NSProbeBidirCB: Thread id is 0xb84550
Information: I-JPR-000-000: [1 SpectroServer1] Successful poll of CORBA interface, primary SpectroServer
{"command":"unacknowledgeAlarm","params":
[{"alarmID":"56079f60-0006-1000-0371-008010b4d48f","spectrumName":"SpectroServer1"}]}
Debug: D-JPR-000-000: com.ibm.tivoli.netcool.omnibus.probe.services.impl.
SimpleCommandService$BidiThread.parseRequest ENTERING
Debug: D-JPR-000-000: com.ibm.tivoli.netcool.omnibus.probe.bidi.CommandHandler.executeCommand ENTERING
Debug: D-JPR-000-000: com.ibm.tivoli.netcool.omnibus.probe.bidi.CommandHandler.executeCommand EXITING
Information: I-JPR-000-000: Command Port received unacknowledge alarm request for alarm :
56079f60-0006-1000-0371-008010b4d48f
Information: I-JPR-000-000: Attempting to unacknowledge : 56079f60-0006-1000-0371-008010b4d48f
Error: E-JPR-000-000: [1 SpectroServer1]
Error un-acknowledging alarmstruct com.aprisma.spectrum.core.idl.
CsCAlarmDomainPackage.CsCAttrErrorListOfAlarms {
com.aprisma.spectrum.core.idl.CsCAlarmDomainPackage.CsCAlarmAttrErrorList[]
list={struct com.aprisma.spectrum.core.idl.CsCAlarmDomainPackage.CsCAlarmAttrErrorList {
byte[] alarmID={86,7,-97,96,0,6,16,0,3,113,0,-128,16,-76,-44,-113},
com.aprisma.spectrum.core.idl.CsCAttribute.CsCAttrErrorList
attrErrorList=struct com.aprisma.spectrum.core.idl.CsCAttribute.CsCAttrErrorList {
com.aprisma.spectrum.core.idl.CsCAttribute.CsCAttrError[]
list={struct com.aprisma.spectrum.core.idl.CsCAttribute.CsCAttrError {
int attributeID=73549,
com.aprisma.spectrum.core.idl.CsCError.CsCError_e error=DOES_NOT_EXIST
com.aprisma.spectrum.core.idl.CsCError.CsCError_e error=PARTIAL_FAILURE
com.aprisma.spectrum.core.idl.CsCError.CsCError_e error=PARTIAL_FAILURE
Error: E-JPR-000-000: Alarm 56079f60-0006-1000-0371-008010b4d48f Error in unacknowledge
```

Storing commands in the nco_http properties file

You can use the **nco_http** utility's properties file ($OMNIHOME/etc/nco_http.props) to hold frequently used command characteristics.

If you have a particular command that you send to the probe regularly, you can store characteristics of that command in the **nco_http** properties file. Once you have done that, the format of the **nco_http** command line is simplified.

You can use one or more of the following **nco_http** properties to hold default values for the equivalent options on the **nco_http** command line:

Data
DataType

Method
URI

Specify the value of each property in the same way as you would on the command line. Once you have these values in place you do not need to specify the corresponding command line switch unless you want to override the value of the property.

The following is an example of the use of the properties file and the simplification of the **nco_http** command that results. In this example, the **nco_http** properties file contains the following values (note that line breaks appear for presentational purposes only; when editing the properties use one line for each property value):

```
Data : '{"command":"ackAlarm", "params":[{"alarmId":"alarm1",
"emsId":"EMS1", "managedElementId":"ME1", "username":"root"}]}'
DataType : 'application/JSON'
Method : 'POST'
```

To use this set of values use the following **nco_http** command:

```
$OMNIHOME/bin/nco_http -uri http://test1.example.com:6789
```

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 more information about generic Netcool/OMNIbus properties and command line options, see the *IBM Tivoli Netcool/OMNIbus Probe and Gateway Guide*.

Table 8. Properties and command line options

Property name	Command line option	Description
CommandPort *integer*	`-commandport` *integer*	This property is no longer supported.
CommandPortLimit *integer*	`-commandportlimit` *integer*	This property is no longer supported.

Table 8. Properties and command line options (continued)		
Property name	**Command line option**	**Description**
EventExtraction *string*	`-eventextraction` *string*	Use this property to specify which method the probe uses to extract details of the events attached to an alarm. This property takes the following values: `AM`: The probe makes an additional call to Archive Manager (AM) to retrieve details of one or more events that raised the alarm. `OE`: The probe extracts from the alarm details of the originating event that raised the alarm from the Originating Event (OE) byte stream, and so does not make an additional call to Archive Manager. The default is `AM`. **Note :** When the probe is connected to the secondary SpectroSERVER, it will always use the OE method to extract details of events attached to an alarm regardless of setting of the **EventExtraction** property.
EventRetryCount *integer*	`-eventretrycount` *integer*	Use this property to specify how many times the probe attempts to extract from the Archive Manager the event details associated with an alarm event before moving on to the next alarm. The maximum that you can set this property to is 5. If you set this property to `0`, the probe will not retry the event details extraction. The default is 3. **Note :** If you set this property to a value other than between `0` and 5, the probe will use the default value of 3.

Table 8. Properties and command line options (continued)

Property name	Command line option	Description
EventRetryTimeout *integer*	`-eventretrytimeout` *integer*	Use this property to specify the time (in seconds) that probe waits to receive from the Archive Manager the event details associated with an alarm. If this time is exceeded, the probe retries the event extraction request. The number of times that the probe retries the event extraction is specified by the **EventRetry** property. The maximum that you can set this property to is 5 seconds. If you set this property to `0`, the probe will not timeout while waiting for the event details. The default is 3. **Note :** If you set this property to a value other than between `0` and 5, the probe will use the default value of 3.
FlushBufferInterval *integer*	`-flushbufferinterval` *integer*	Use this property to specify how often (in seconds) the probe flushes all alerts in the buffer to the ObjectServer. The default is `0` (the probe does not flush alerts to the ObjectServer).
HeartbeatInterval *integer*	`-heartbeatinterval` *integer*	Use this property to specify the frequency (in seconds) with which the probe checks the status of each connection for the targeted Spectrum server. The default is `0`.
HostFile *string*	`-hostfile` *string*	Use this property to specify the location of the host configuration file `ca_spectrum_corba_host.xml`. The default is `""`.
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).

Table 8. Properties and command line options (continued)		
Property name	**Command line option**	**Description**
InitialResync *integer*	`-initialresync` *string*	Use this property to specify whether the probe retrieves all active alarms when it connects or only those that have been created since the last connection. This property takes the following values: `false`: The probe only retrieves alarms generated since the last connection. `true`: The probe retrieves all active alarms; no alarm filter is applied by the probe. The default is `false`.
MaxSpectrumSeverity *integer*	`-maxspectrumseverity` *integer*	Use this property to specify the maximum severity level above which the probe will not retrieve alarms. The default is 6 (CA Spectrum severity Initial).
MinSpectrumSeverity *integer*	`-minspectrumseverity` *integer*	Use this property to specify the minimum severity level below which the probe will not retrieve alarms. The default is `0` (CA Spectrum severity Normal).
ORBLocalHost *string*	`-orblocalhost` *string*	Use this property to specify the local host name or IP address used by the server-side ORB to place the server's host name or IP address into the IOR of a remote object. The default is `""`.
ORBLocalPort *string*	`-orblocalport` *string*	Use this property to specify the local port to which the Object Request Broker (ORB) listens for connections from the probe. The default is `0` (the ORB selects an available port at random).
OSHost *string*	`-oshost` *string*	Use this property to specify the host name of the ObjectServer to which the probe connects during resynchronization operations. The default is `localhost`.

Table 8. Properties and command line options (continued)		
Property name	**Command line option**	**Description**
OSPassword *string*	`-ospassword` *string*	Use this property to specify the password of the ObjectServer to which the probe connects during resynchronization operations. The default is `""`. Use the `nco_g_crypt` utility supplied with Netcool/OMNIbus to encrypt the password. . For information about using this utility, see the *IBM Tivoli Netcool/OMNIbus Administration Guide*.
OSPort *integer*	`-osport` *integer*	Use this property to specify the port number of the ObjectServer to which the probe connects during resynchronization operations. The default is `4100`.
OSUserName *string*	`-osusername` *string*	Use this property to specify the user name used to connect to the ObjectServer during resynchronization operations. The default is `root`.
ResyncInterval *integer*	`-resyncinterval` *string*	This property sets a global **ResyncInterval** value for the entire probe. It is found in the `ca_spectrum_corba.props` file. Any probe's target connection that does not have **ResyncInterval** configured in the probe's host properties file (`ca_spectrum_corba_host.xml`) will refer to this value. The default value is `0`. **Note :** Use for multiple target environment where some target connections will have the same **ResyncInterval** value, while other subsets from it would not. In the event both **ResyncInterval** properties are configured, the **ResyncInterval** property from `ca_spectrum_corba_host.xml` always takes precedence over the **ResyncInterval** property from `ca_spectrum_corba.props`.

Table 8. Properties and command line options (continued)		
Property name	**Command line option**	**Description**
ResyncOnFailover *integer*	`-resynconfailover` *string*	Use this property to specify whether the probe performs a resynchronization during the SpectroSERVER failover process. This property takes the following values: `true`: The probe performs a resynchronization. `false`: The probe does not perform a resynchronization. The default is `false`.
ResyncPrimaryOnly *integer*	`-resyncprimaryonly` *string*	Use this property to make the probe resynchronize with the primary SpectroSERVER only during the SpectroSERVER failover process. This property takes the following values: `true`: The probe performs a resynchronization with the primary SpectroSERVER only. `false`: The probe resynchronizes with either SpectroSERVER. The default is `false`.
ResyncThreadMax *integer*	`-resyncthreadmax` *integer*	Use this property to specify the maximum number of threads that the probe spawns to perform a resynchronization. If you set this property to a value of `0` the probe uses an unlimited number of threads. The default is `5`.
RetryCount *integer*	**-retrycount** *integer*	**RetryCount** is the number of attempts the probe as a whole will attempt to reestablish all connections to the SpectroSERVER. The default is `0`.
RetryInterval *integer*	`-retryinterval` *integer*	Use this property to define the length of retry interval (in seconds) for reconnecting to the SpectroSERVER after being disconnected. The default is `30`.

Table 8. Properties and command line options (continued)		
Property name	**Command line option**	**Description**
SpectroServer PollInterval *integer*	`-spectroserver pollinterval` *integer*	Use this property to specify the interval (in seconds) at which the probe polls the CORBA services to check their availability. The default is 20 seconds.
SpectrumRetry *string*	`-spectrumretry` *string*	Use this property to specify whether the probe attempts to reconnect to a SpectroSERVER after losing the connection. This property takes the following values: `true`: The probe's connection waits for connection to reestablish with the SpectroSERVER. `false`: The probe does not wait for connection to reestablish with SpectroSERVER. The default is `true`. For more information about using **SpectrumRetry** scenarios see "Backoff strategy" on page 14.

Properties and command line options provided by the Java Probe Integration Library (probe-sdk-java) version 8.0

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 8.0.

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

Table 9. Properties and command line options		
Property name	**Command line option**	**Description**
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 9. 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 `1`.
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 `0`. **Note :** You can increase this number to increase the event throughput when a large number of events is generated.

Table 9. 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).

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 following table describes the elements that the probe generates. Not all the elements described are generated for each event. The elements that the probe generates depend on the event type.

Table 10. Elements	
Element name	**Element description**
`$Acknowledged`	This element indicates if the alarm has been acknowledged by a CA Spectrum operator.

Table 10. Elements (continued)

Element name	Element description
$AlarmID	This element contains the identifier of the alarm. The probe sends this element to the rules file as a zero-padded 8-character hexadecimal string. **Note :** The alarm ID changes when an event fails over to a secondary SpectroSERVER.
$AlarmStatus	This element displays the status of the alarm.
$AlarmSource	This element indicates the source of the alarm. The following values are currently supported: 0: Specifies that the alarm is current. 1: Specifies that the alarm is residual from a previous run of the SpectroSERVER.
$ClearMe	This element indicates whether the event is a resolution event. A value of true sets the ObjectServer field @Type to 2 in the rules file. This indicates that it is a resolution event.
$CauseNum	This element shows the probable cause number of the event. The probe sends this element to the rules file as a zero-padded 8-character hexadecimal string.
$ClearedBy	This element contains the user name associated with the user who cleared the alarm through the CA Spectrum API. This element is not present when an alarm is created but it is added prior to removal.
$CreationDate	This element contains the date when the alarm was generated.
$DomainID	This element contains the domain ID of the SpectroSERVER.
$EventIDList	This element contains the list of SpectroSERVER event identifiers that triggered the alarm.
$IPAddress	This element contains the IP address of the reporting node.
$ModelHandle	This element contains a model handle. A model handle is a 32-bit number in which the high-order 12 bits make up the landscape and the remaining 20 bits make up the model identifier.
$ModelID	This element shows the identifier of the model.
$ModelName	This element contains a description of the model.

Table 10. Elements (continued)	
Element name	**Element description**
$ModelType	This element shows the type of the device that raised the alarm.
$Occurrences	This element shows the number of times that the event has occurred.
$PrimaryAlarm	This element indicates the priority of the alarm in the SpectroSERVER. A value of `true` identifies the alarm as a primary alarm. Primary alarms have the highest priority.
$Pre-existing	This element indicates whether the received event is already present in the SpectroSERVER.
$Primary	This element indicates whether the probe is connected to a primary SpectroSERVER or to a secondary SpectroSERVER. A value of false indicates that the probe is connected to a secondary server.
$Priority	This element indicates the priority in the SpectroSERVER.
$Prob_cause	This element indicates the probable cause of an issue with the alarm in the SpectroSERVER.
$Recommended_actions	This element indicates the recommended actions to take to resolve an issue with the alarm in the SpectroSERVER.
$Severity	This element indicates the severity of the alarm in the SpectroSERVER.
$Symptom	This element indicates the symptom of the alarm in the SpectroSERVER.
$Title	This element indicates the title of the alarm in the SpectroSERVER.
$TroubleShooter	This element displays the name of the specified troubleshooter. This element maps to the `TroubleShooter` SpectroSERVER field, which can be updated using the**updateTroubleShooterName** CLI command.
$TroubleShooterModel	This element displays the model type of the specified troubleshooter. This element maps to the *T*`roubleShooterModel` SpectroSERVER field, which can be updated using the *updateTroubleShooterModel* CLI command.

Table 10. Elements (continued)	
Element name	**Element description**
`$TroubleTicketID`	This element displays the identifier of the trouble ticket associated with an alarm. This element maps to the `TroubleTicketID` SpectroSERVER field, which can be updated using the **`updateTroubleTicket`** CLI command.
`$UserClearable`	This element indicates whether or not a client can clear the alarm. This element is provided so that client applications can indicate to users that the alarm is not clearable. A client cannot control whether an alarm is clearable.

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 11. Error messages

Error	Description	Action

`Exception when attempting to run status monitor`
`Probe shutting down because CORBA Status monitor not polling successfully.`
 	
`Exception thrown when attempting to start CORBA service`
`Probe shutting down as unable to access CORBA service, please ensure that server configuration is complete and try again`
`Probe shutting down because Probe unable to access CORBA service`
 	
`Exception when attempting to establish a filter for alarms`
`Probe shutting down as unable to receive alarms,please ensure that the timestamp file pointed to does not contain corrupted data`
`Probe shutting down because unable to establish Alarm Watch filter`
 	
`Exception when attempting to establish an Alarm Watch`
`Probe shutting down as unable to receive alarms,please ensure that server configuration is complete and try again`
`Probe shutting down because unable to establish Alarm Watch`
	The probe could not initialize the connection with the CORBA interface and is shutting down.	Check the SpectroSERVER configuration.
		
		Verify that the values set for the `Domain`, `Name`, and `IP` attributes of the `SpectroServer` property are correct in the host configuration file.
		
		Check the firewall settings in your environment.
		
		Check that the appropriate additions were made to the SpectroSERVER server list.
		
		Verify that a Spectrum User Model exists for the user ID that runs the probe.

Table 11. Error messages (continued)

Error	Description	Action

`Error occurred attempting to pass on details of cleared alarms to the Object Server. Probe shutting down because Problem processing cleared alarms.`
`Error occurred attempting to pass updates to the Object Server Probe shutting down because Problem processing updated alarms.`
`Error occurred attempting to pass details of new alarms to the Object Server Probe shutting down because Problem processing new alarms`
`Error in connecting to CORBA interface Error external to CORBA framework Probe shutting down because External error in connection to CORBA interface`
`Exception thrown when querying whether we are accessing Primary SpectroSERVER Probe shutting down because Problem querying SpectroSERVER.`
`Connection to CORBA interface has been lost. Error internal to CORBA interface Probe shutting down because Internal error in CORBA interface, connection lost`
`Error shutting down connection to CORBA interface`
	The probe connection with the CORBA interface has failed.	Restart the probe.
		Check whether the SpectroSERVER configurations have changed.
		Verify that the SpectroSERVER is running.

Table 11. Error messages (continued)

Error	Description	Action

`Failed to pick up probe property TimeStampFile Not using timestamp file, proceeding to collect all active alarms.`
`Timestamp of alarms not stored for re-sync`
`Unable to create timestamp file`
`Unable to find time stamp file`
`Unable to write to timestamp file`
`Error closing timestamp file`
`Error opening timestamp file for reading`
	The probe failed to use the timestamp file. These errors are not fatal error messages.	Check the specific message and verify that the timestamp file is accessible.

Table 11. Error messages (continued)

Error	Description	Action

`Alarm + alarmID + Error clearing alarm`
`Error clearing alarm`
 	
`Alarm + alarmID + Error in Event ID List update`
`Error updating the event ID List`
 	
`Alarm + alarmID + Error in Trouble Shooter Model update`
`Error updating the trouble shooter model`
 	
`Alarm + alarmID + Error in Trouble Ticket update`
`Error updating the trouble ticket`
 	
`Alarm + alarmID + Error in Troubleshooter Name update`
`Error updating the trouble shooter name`
 	
`Alarm + alarmID + Error in status update`
`Error updating the alarm status`
 	
`Alarm + alarmID +Error in unacknowledge`
`Error un-acknowledging alarm`
 	
`Alarm + alarmID + Error in acknowledge`
`Error acknowledging alarm`
	The probe failed to update an event.	Verify that the alarm you are trying to update exists in the SpectroSERVER.
		Note : If the alarm is not found, the SpectroSERVER shows an error message.

`Invalid spectrum alarm severity: MaxSpectrumSeverity is greater than 6. Valid spectrum alarm severity value is from 0 to 6.`	The value set for the **MaxSpectrumSeverity** property is greater than the maximum valid value. The valid values for this property are between 0 to 6.	Set the **MaxSpectrumSeverity** property to a value between 0 to 6.

Table 11. Error messages (continued)

Error	Description	Action
`Invalid spectrum alarm severity: the values of MinSpectrumSeverity and MaxSpectrumSeverity are not in order. Valid spectrum alarm severity value is from 0 to 6.`	The value set for the **MinSpectrumSeverity** property is greater than that set for the **MaxSpectrumSeverity** property.	Set the **MinSpectrumSeverity** property to a value between 0 to 6, and less than that set for the **MaxSpectrumSeverity** property.
`Invalid spectrum alarm severity: MinSpectrumSeverity is less than 0. Valid spectrum alarm severity value is from 0 to 6.`	The value set for the **MinSpectrumSeverity** property is less than 0, which is not valid.	Set the **MinSpectrumSeverity** property to a value between 0 to 6.
`Host parser exception : file:///<host configuration file path> LineNumber=62 ColumnNumber=21 Msg=cvc-complex-type.2.4.b: The content of element 'ProbeHeadProperty' is not complete. One of '{SpectroServer}' is expected. Error in parsing host configuration file. Probe will shutdown`	No SpectroServer is specified in host configuration file.	Specify at least one SpectroServer in the host configuration file.
`Host parser exception : file:///<host configuration file path> LineNumber=51 ColumnNumber=37 Msg=cvc-complex-type.2.4.a: Invalid content was found starting with element '<property name in host config file>'. One of '{AlarmAttributes, TimeStampFile, FetchEventFormatFields, FetchEventString, EventFormatFile, ProbCauseLookupFile, ResyncInterval, ResyncSQLCmd}' is expected. Error in parsing host configuration file. Probe will shutdown`	Duplicate entries were found in the host configuration file for at least one of the following properties: **AlarmAttributes**, **TimeStampFile**, **FetchEventFormatFields**, **FetchEventString**, **EventFormatFile**, **ProbCauseLookupFile**, **ResyncInterval**, **ResyncSQLCmd**.	Specify only one entry in the host configuration file for any of the properties listed.

Table 11. Error messages (continued)		
Error	**Description**	**Action**
`[servername] failed to retrieve related events:` `[servername] no events could be fetched successfully for this alarm`	The Archive Manager service is down and the probe could not retrieve events for the alarm from the Archive Manager.	Restart the Archive Manager service.
`Unknown host configuration file, <invalid host path>` `Error in parsing host configuration file. Probe will shutdown`	The probe could not detect the host configuration file because the path is invalid.	Set the a valid path to the host configuration file for the **Host** property.

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 12. ProbeWatch messages		
ProbeWatch message	**Description**	**Triggers/causes**
`Error in Retry property value. Probe will shutdown`	The **Retry** property is set to invalid value and so the probe is shutting down.	An invalid value was set for the **Retry** property.
`Probe has lost connection to CORBA interface, error external to CORBA interface`	The probe has lost the connection to the CORBA interface.	The connection between the SpectroSERVER and the CA Spectrum EMS is down.
`Probe has lost connection to CORBA interface. Error internal to CORBA interface`	The probe has lost the connection to the CORBA interface.	The connection between the probe and the server is down due to incorrect probe settings or an inactive SpectroSERVER.
`New successful connection to the CORBA interface, connected to the Primary SpectroSERVER`	The probe connected to the primary SpectroSERVER.	The connection to the primary SpectroSERVER was successful.

Table 12. ProbeWatch messages (continued)

ProbeWatch message	Description	Triggers/causes
`New successful connection to the CORBA interface, connected to the Secondary SpectroSERVER`	The probe connected to the secondary SpectroSERVER.	The primary SpectroSERVER has failed, and the probe has connected to the secondary SpectroSERVER.
`Start resync for` *`spectroServerName`*	The probe started the resynchronization process with the Spectrum server indicated.	Either a resynchronization was requested by a CLI command, or one was initiated automatically following the elapsing of a resynchronization interval, or the failover of the Spectrum server.
`Complete resync for` *`spectroServerName`*	The probe's resynchronization with the Spectrum server indicated has completed.	The resynchronization with the Spectrum server indicated completed successfully.
`Probe successfully connected to <target host>`	The probe started and is running successfully.	The probe process was started by the operator.
`Running ...`	The probe started and is running successfully.	The probe process was started by the operator.
`Probe is shutting down Going Down ...`	The probe is shutting down.	The system has been requested to shut down the probe because the probe process was terminated manually by the operator.
`<target host> head is shutting down, remove from active host list`	CA Spectrum EMS is shutting down after the probe connected to SpectroSERVER. This ProbeWatch will be shown when the retry value for the probe is set to 0.	CA Spectrum EMS that is connecting to the probe is shutting down. This is either because the Spectrum service has been stopped manually by operator or has aborted.
`Error in spectrum alarm severity values. Probe will shutdown.`	Invalid values have been set for either the **MaxSpectrumSeverity** property or the **MinSpectrumSeverity** property .	Either the value set for the **MinSpectrumSeverity** property is higher than that set for the **MaxSpectrumSeverity**, or the value set for either **MinSpectrumSeverity** or **MaxSpectrumSeverity** is not between 0 and 6.
`Received connection from 127.0.0.1`	The probe connected to command port session and is ready to receive CLI commands.	The operator managed to successfully telnet to the command port.

Table 12. ProbeWatch messages (continued)		
ProbeWatch message	**Description**	**Triggers/causes**
`Error in parsing host configuration file. Probe will shutdown`	The probe failed to parse the host configuration file.	Either there are missing values in the host configuration file (for example a missing SpectroServer host), duplicate entries were found in the host configuration file for at least one of the following properties: **AlarmAttributes**, **TimeStampFile**, **FetchEventFormatFields**, **FetchEventString**, **EventFormatFile**, **ProbCauseLookupFile**, **ResyncInterval**, **ResyncSQLCmd**, or the path to the host configuration file specified for the **Host** property is an invalid path.

Troubleshooting

Various issues arise as users work with the probe. Troubleshooting information is provided to help you diagnose and resolve such issues.

Fails to Connect

The Probe for CA Spectrum (CORBA) fails to connect to the SpectroSERVER when configured with a host file. You may receive the following message when configuring a host file:

```
07/01/10 10:36:39: Debug: [1 server1] Connecting to SpectroServer
07/01/10 10:36:42: Debug: [1 server1] Exception =
org.omg.CORBA.OBJECT_NOT_EXIST:   vmcid: 0x0  minor code: 0  completed: No
07/01/10 10:36:42: Error: Exception thrown when attempting to start CORBA service:
07/01/10 10:36:42: Error: Unable to access CORBA service, head is shutting down
```

This indicates that there is a connection issue between the probe and the SpectroSERVER. A possible reason is that the probe cannot find the location of the SpectroSERVER specified in the `ca_spectrum_corba_host.xml` host file.

To resolve this problem, set the domain property in host file to the SpectroSERVER host FQDN which is reachable from probe host.

For the information about the required format of the connection information in the host file, see “Using the host configuration file” on page 5.

Migration Guidelines

The table below provides a comparison between older version of CA Spectrum probe that are useful when migrating to new version of the probe.

Table 13. Migration Changes

Existing: **nco_p_spectrum_corba_v9**	**New:** **nco_p_ca_spectrum_corba**	**Changes**
AllAlarmsOnRestart	**InitialResync**	This property takes the following values: `false`: The probe only retrieves alarms generated since the last connection. `true`: The probe retrieves all active alarms; no alarm filter is applied by the probe. The default is `false`.
EventRetry	**EventRetryCount**	Name change
OS.Host	**OSHost**	Name change
OS.Password	**OSPassword**	Name change
OS.Port	**OSPort**	Name change
OS.Username	**OSUsername**	Name change
Retry	**SpectrumRetry**	Use this property to specify whether the probe attempts to reconnect to a SpectroSERVER after losing the connection. This property takes the following values: `true`: The probe's connection waits for connection to reestablish with the SpectroSERVER. `false` : The probe does not wait for connection to reestablish with SpectroSERVER. The default is `true`.
ResyncOnFailOver	**ResyncOnFailOver**	Use this property to specify whether the probe performs a resynchronization during the SpectroSERVER failover process. This property takes the following values: `true`: The probe performs a resynchronization. `false`: The probe does not perform a resynchronization. The default is `false`.

Table 13. Migration Changes (continued)		
Existing: **nco_p_spectrum_corba_v9**	**New:** **nco_p_ca_spectrum_corba**	**Changes**
ResyncOnPrimaryOnly	**ResyncOnPrimaryOnly**	Use this property to make the probe resynchronize with the primary SpectroSERVER only during the SpectroSERVER failover process. This property takes the following values: `true`: The probe performs a resynchronization with the primary SpectroSERVER only. `false`: The probe resynchronizes with either SpectroSERVER. The default is `false`.

Table 14. Configuration File Name Changes	
Existing Configuration Files	**New Configuration Files**
`spectrum_corba_v9.rules`	`ca_spectrum_corba.rules`
`spectrum_corba_v9.props`	`ca_spectrum_corba.props`
`spectrum_corba_v9.xsd`	`ca_spectrum_corba.xsd`
`spectrum_corba_v9_host.xml`	`ca_spectrum_corba_host.xml`
`nco_p_spectrum_corba_v9.env`	`nco_p_ca_spectrum_corba.env`

Known issues

At the time of release, several known issues were reported that you should be aware of when running the probe.

Using timestamp files

The `TimeStampFile` property in the host configuration file enables you to specify a file in which the probe can log the timestamp of the last alarm it processed. When the **`InitialResync`** property is disabled, and a timestamp file is specified, the alarms received by the probe are filtered according to this timestamp.

The timestamp filter works in the following way. Assume that the probe shuts down and logs timestamp T to the timestamp file. Then the probe starts up again and sets up two timestamp filters, TS1 and TS2. The value of the TS1 filter is equal to the value of the last logged timestamp minus 48 hours (TS1 = T - 48 hours). The value of the TS2 filter is equal to T (TS2 = T).

The purpose of the TS1 filter is to enable alarms up to 48 hours old, that were cleared or updated while the SpectroSERVER was connected to the probe, to be forwarded to the probe. So all alarms created after TS1 are forwarded to the probe. However, the probe only accepts as *active* those alarms that were created after TS2. The TS2 filter prevents alarms that have already been processed by the probe from being resynchronized.

The issue arises because, although the probe will accept *updates* to alarms that were created after TS1, it cannot logically accept *updates* made in the SpectroSERVER to older alarms that it has already filtered out. To avoid this issue, do not use the `TimeStampFile` property in the host configuration file.

Error using the UpdateEventList command

The `UpdateEventList alarmID eventIDList spectrumName` CLI command enables you to update the event list of an alarm by specifying the alarm identifier, the event list identifier and the SpectroSERVER host name.

This command feature is currently unavailable because when issued to the SpectroSERVER, the command produces an exception with an error of `TYPE_RESTRICTION` at the Spectrum API level.

SpectroSERVER automatically disconnecting the probe

When the total time taken by archive manager to retrieve events exceeds the time set in the `MAX_EVENT_ID_REQUEST_TIMEOUT` field on SpectroSERVER, SpectroSERVER will disconnect the probe, but will not inform the probe that it has been disconnected. The probe will fail to retrieve any further events from SpectroSERVER until you manually restart the probe.

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.

SC27-8702-02