Netcool/OMNibus
Version 7 Release 4

ObjectServer HTTP Interface Reference Guide

IBM
Before using this information and the product it supports, read the information in "Notices" on page 55.

This edition applies to version 7, release 4 of IBM Tivoli Netcool/OMNibus (product number 5724-S44) and to all subsequent releases and modifications until otherwise indicated in new editions.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
About this publication

Tivoli Netcool/OMNIbus is a service level management (SLM) system that delivers real-time, centralized monitoring of complex networks and IT domains.

The IBM Tivoli Netcool/OMNIbus Administration Guide provides detailed information about administrative tools, functions, and capabilities of Tivoli Netcool/OMNIbus. In addition, it is designed to be used as a reference guide to assist you in designing and configuring your environment.

Intended audience

This publication is intended for administrators who are responsible for configuring Tivoli Netcool/OMNIbus.

What this publication contains

This publication contains the following sections:

- **Chapter 1, “Overview of the ObjectServer HTTP interface,” on page 1** Describes the HTTP interface and how to enable it by setting ObjectServer properties.
- **Chapter 2, “HTTP interface URIs,” on page 7** Describes the URIs that give access to table data and to rows in tables, execute SQL commands via HTTP, and give access to system information.
- **Chapter 3, “Common behaviors,” on page 29** Lists the HTTP and HTTPS version support, query parameters, authentication mechanisms, and so on.
- **Appendix A, “Example JSON payloads,” on page 35** Sample JSON payloads.
- **Appendix B, “HTTP request and response examples,” on page 39** Sample HTTP requests and responses.
- **“List of abbreviations” on page ix** Terms and abbreviations that are used in this publication.

Publications

This section lists publications in the Tivoli Netcool/OMNIbus library and related documents. The section also describes how to access Tivoli publications online and how to order Tivoli publications.

Your Tivoli Netcool/OMNIbus library

The following documents are available in the Tivoli Netcool/OMNIbus library:

- **IBM Tivoli Netcool/OMNIbus Installation and Deployment Guide, SC14-7526**
  Includes installation and upgrade procedures for Tivoli Netcool/OMNIbus, and describes how to configure security and component communications. The publication also includes examples of Tivoli Netcool/OMNIbus architectures and describes how to implement them.
- **IBM Tivoli Netcool/OMNIbus Administration Guide, SC14-7527**
Describes how to perform administrative tasks using the Tivoli Netcool/OMNibus Administrator GUI, command-line tools, and process control. The publication also contains descriptions and examples of ObjectServer SQL syntax and automations.

- **IBM Tivoli Netcool/OMNibus Web GUI Administration and User’s Guide, SC14-7528**
  Describes how to perform administrative and event visualization tasks using the Tivoli Netcool/OMNibus Web GUI.

- **IBM Tivoli Netcool/OMNibus User’s Guide, SC14-7529**
  Provides an overview of the desktop tools and describes the operator tasks related to event management using these tools.

- **IBM Tivoli Netcool/OMNibus Probe and Gateway Guide, SC14-7530**
  Contains introductory and reference information about probes and gateways, including probe rules file syntax and gateway commands.

- **IBM Tivoli Monitoring for Tivoli Netcool/OMNibus Agent User’s Guide, SC14-7532**
  Describes how to install the health monitoring agent for Tivoli Netcool/OMNibus and contains reference information about the agent.

- **IBM Tivoli Netcool/OMNibus Event Integration Facility Reference, SC14-7533**
  Describes how to develop event adapters that are tailored to your network environment and the specific needs of your enterprise. This publication also describes how to filter events at the source.

- **IBM Tivoli Netcool/OMNibus Error Messages Guide, SC14-7534**
  Describes system messages in Tivoli Netcool/OMNibus and how to respond to those messages.

- **IBM Tivoli Netcool/OMNibus Web GUI Administration API (WAAPI) User’s Guide, SC22-7535**
  Shows how to administer the Tivoli Netcool/OMNibus Web GUI using the XML application programming interface named WAAPI

- **IBM Tivoli Netcool/OMNibus ObjectServer HTTP Interface Reference Guide, SC27-5613**
  Describes the URLs and common behaviors of the Application Programming Interface (API) that is called the ObjectServer HTTP Interface. Describes how to enable the API and provides examples of JSON payloads, and HTTP requests and responses.

  Describes the services, resources, and common behaviors of the Open Services for Lifecycle Collaboration (OSLC) Application Programming Interface (API) that is called the ObjectServer OSLC Interface. Describes how to enable the API and provides examples of service provider definitions, RDF/XML payloads, and HTTP requests and responses.

### Accessing terminology online

The IBM Terminology Web site consolidates the terminology from IBM product libraries in one convenient location. You can access the Terminology Web site at the following Web address:


### Accessing publications online

IBM posts publications for this and all other Tivoli products, as they become available and whenever they are updated, to the Tivoli Information Center Web site at:
Note: If you print PDF documents on other than letter-sized paper, set the option in the File > Print window that allows Adobe Reader to print letter-sized pages on your local paper.

Ordering publications

You can order many Tivoli publications online at the following Web site:


You can also order by telephone by calling one of these numbers:

- In the United States: 800-879-2755
- In Canada: 800-426-4968

In other countries, contact your software account representative to order Tivoli publications. To locate the telephone number of your local representative, perform the following steps:
1. Go to the following Web site:
2. Select your country from the list and click Go. The Welcome to the IBM Publications Center page is displayed for your country.
3. On the left side of the page, click About this site to see an information page that includes the telephone number of your local representative.

Accessibility

Accessibility features help users with a physical disability, such as restricted mobility or limited vision, to use software products successfully.

With this product, you can use assistive technologies to hear and navigate the interface. You can also use the keyboard instead of the mouse to operate some features of the graphical user interface.

Tivoli technical training

For Tivoli technical training information, refer to the following IBM Tivoli Education Web site:

http://www.ibm.com/software/tivoli/education

Support information

If you have a problem with your IBM software, you want to resolve it quickly. IBM provides the following ways for you to obtain the support you need:

Online
Go to the IBM Software Support site at http://www.ibm.com/software/support/probsub.html and follow the instructions.

IBM Support Assistant
The IBM Support Assistant (ISA) is a free local software serviceability workbench that helps you resolve questions and problems with IBM software products. The ISA provides quick access to support-related
information and serviceability tools for problem determination. To install the ISA software, go to http://www.ibm.com/software/support/isa.

Documentation
If you have a suggestion for improving the content or organization of this guide, send it to the Tivoli Netcool/OMNibus Information Development team at:

mailto://L3MMDOCS@uk.ibm.com

Conventions used in this publication

This publication uses several conventions for special terms and actions and operating system-dependent commands and paths.

Typeface conventions

This publication uses the following typeface conventions:

**Bold**
- Lowercase commands and mixed case commands that are otherwise difficult to distinguish from surrounding text
- Interface controls (check boxes, push buttons, radio buttons, spin buttons, fields, folders, icons, list boxes, items inside list boxes, multicolumn lists, containers, menu choices, menu names, tabs, property sheets), labels (such as Tip: and Operating system considerations:)
- Keywords and parameters in text

**Italic**
- Citations (examples: titles of publications, diskettes, and CDs)
- Words defined in text (example: a nonswitched line is called a point-to-point line)
- Emphasis of words and letters (words as words example: "Use the word that to introduce a restrictive clause.",; letters as letters example: "The LUN address must start with the letter L.")
- New terms in text (except in a definition list): a view is a frame in a workspace that contains data
- Variables and values you must provide: ... where myname represents....

**Monospace**
- Examples and code examples
- File names, programming keywords, and other elements that are difficult to distinguish from surrounding text
- Message text and prompts addressed to the user
- Text that the user must type
- Values for arguments or command options

Operating system-dependent variables and paths

This publication uses the UNIX convention for specifying environment variables and for directory notation.

When using the Windows command line, replace $variable with %variable% for environment variables, and replace each forward slash (/) with a backslash (\) in directory paths. For example, on UNIX systems, the $NCHOME environment
variable specifies the path of the Netcool® home directory. On Windows systems, the %NCHOME% environment variable specifies the path of the Netcool home directory. The names of environment variables are not always the same in the Windows and UNIX environments. For example, %TEMP% in Windows environments is equivalent to $TMPDIR in UNIX environments.

If you are using the bash shell on a Windows system, you can use the UNIX conventions.

**Operating system-specific directory names**

Where Tivoli Netcool/OMNIbus files are identified as located within an arch directory under NCHOME, arch is a variable that represents your operating system directory, as shown in the following table.

<table>
<thead>
<tr>
<th>Directory name represented by arch</th>
<th>Operating system</th>
</tr>
</thead>
<tbody>
<tr>
<td>aix5</td>
<td>AIX® systems</td>
</tr>
<tr>
<td>hpux11hpia</td>
<td>HP-UX Itanium-based systems</td>
</tr>
<tr>
<td>linux2x86</td>
<td>Red Hat Linux and SUSE systems</td>
</tr>
<tr>
<td>linux2s390</td>
<td>Linux for System z®</td>
</tr>
<tr>
<td>solaris2</td>
<td>Solaris systems</td>
</tr>
<tr>
<td>win32</td>
<td>Windows systems</td>
</tr>
</tbody>
</table>

**Fix pack information**

Information that is applicable only to the fix pack versions of Tivoli Netcool/OMNIbus are prefaced with a graphic. For example, if a set of instructions is preceded by the graphic Fix Pack 1, it means that the instructions can only be performed if you installed fix pack 1 of your installed version of Tivoli Netcool/OMNIbus. In the release notes, descriptions of known problems that are prefaced with Fix Pack 1 are solved in fix pack 1, and so on.

**Note:** Fix packs are distributed separately for the server components and the Web GUI component.

**List of abbreviations**

The API documentation for the ObjectServer HTTP interface and the ObjectServer OSLC interface use the following abbreviations and terms.

**HTTP** Hyper Text Transfer Protocol. HTTP version 1.1 is defined in RFC2616. Unless otherwise noted, the term HTTP is used in this document to mean both HTTP and HTTPS.

**HTTPS** Hyper Text Transfer Protocol Secure, as defined in RFC2818.


**JSON** JavaScript Object Notation, as defined in ECMA-262.
| **MIME** | Multipurpose Internet Mail Extensions. MIME media types are defined in [IANA MIME Media Types](http://tools.ietf.org/html/rfc2046). |
| **OSLC** | Open Services for Lifecycle Collaboration, as defined at [http://open-services.net](http://open-services.net). |
| **REST** | Representational State Transfer, as originally and informally described in *Architectural Styles and the Design of Network-based Software Architectures*. |
| **URI** | Uniform Resource Identifier, as defined in RFC3986. |
| **XML** | eXtensible Markup Language, as defined by W3C. |
Chapter 1. Overview of the ObjectServer HTTP interface

The HTTP interface is a lightweight Application Programming Interface (API) that is hosted in the ObjectServer. The HTTP interface provides access to table data in the ObjectServer through a structured URI format that uses HTTP. POST, PATCH, GET, and DELETE requests. Requests are supported against table URIs or row URIs. Access to the URI is authenticated by a known ObjectServer user through basic HTTP authentication. The interface can be secured through an HTTPS connection. You can enable the interface by setting properties in the ObjectServer.

The ObjectServer hosts another API that is called the OSLC interface. This API is an event server provider that presents a resource-linked data view of events and the associated journal and detail resources. For more information about the OSLC interface, see the IBM Tivoli Netcool/OMNIbus ObjectServer OSLC Interface Reference Guide.

Base URI

The base URI for the HTTP interface is as follows.

http://host:port/objectserver/restapi/

Related tasks:

"Enabling the HTTP interface and OSLC interface in the ObjectServer"

The ObjectServer HTTP and OSLC interfaces are disabled by default, because the interfaces need to be configured for a secure setup.

Enabling the HTTP interface and OSLC interface in the ObjectServer

The ObjectServer HTTP and OSLC interfaces are disabled by default, because the interfaces need to be configured for a secure setup.

Before you begin

Work out which ObjectServers in your environment need to be accessed via HTTP or HTTPS. Not all ObjectServers in an environment need to grant access to ObjectServer data through an HTTP-based mechanism.

About this task

Because the hosting of the HTTP and OSLC interfaces in the ObjectServer requires an embedded HTTP server, the ObjectServer can serve files to HTTP clients. Although the ObjectServer can serve pages, it is not optimized for page-serving, unlike an Apache web server. For this reason, do not use the ObjectServer to host anything other than rudimentary HTML or JavaScript pages.

Procedure

1. To enable the interfaces, set the NRestOS.Enable property to TRUE.
2. To configure the embedded HTTP server so that the interfaces are active on an HTTP port, specify the listening port for the connection type. For example, to make the interfaces listen on port 8080, set the properties as follows:

\[\text{NHttpd.EnableHTTP : TRUE}\
\text{NHttpd.ListeningPort : 8080}\]
3. If you want the interfaces to be active on an HTTPS port on 9090, set the properties that are shown in the following example. Because an HTTPS port is SSL encrypted, a certificate file that contains an appropriate certificate needs to be created and protected by a password.

```
NHttpd.SSLEnable : TRUE
NHttpd.SSLListeningPort : 9090
NHttpd.SSLCertificate : "certificatelabel"
NHttpd.SSLCertificatePwd : "password"
```

4. To enable file-serving from the ObjectServer, set the `NHttpd.EnableFileServing` property. The root of the served pages is defined by the `NHttpd.DocumentRoot` property.

5. **Fix Pack 2** To generate the members resource reference list in the RDF/XML payload of Event, Journal, and Detail query capability responses in both `Collection` and `ResponseInfo` resource instances, set the `NRestOS.OSLCRDFMsgFormat` to “MIGRATION”. For more information about this parameter and why you might need to set it, see the section *Updates to the HTTP interface and OSLC interface* in the Release Notes.

**Related concepts:**

- Chapter 1, “Overview of the ObjectServer HTTP interface,” on page 1

The HTTP interface is a lightweight Application Programming Interface (API) that is hosted in the ObjectServer. The HTTP interface provides access to table data in the ObjectServer through a structured URI format that uses HTTP, POST, PATCH, GET, and DELETE requests. Requests are supported against table URIs or row URIs. Access to the URI is authenticated by a known ObjectServer user through basic HTTP authentication. The interface can be secured through an HTTPS connection. You can enable the interface by setting properties in the ObjectServer.

### ObjectServer properties that control the HTTP interface and OSLC interface

ObjectServer properties that control the HTTP and OSLC interfaces.

The following table lists the ObjectServer properties that control the HTTP interface and the OSLC interface.

**Table 2. Properties and command-line options for controlling the HTTP interface and the OSLC interface**

<table>
<thead>
<tr>
<th>Property</th>
<th>Command-line option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NRestOS.Enable</td>
<td>-nrestosenable</td>
<td>Enables the HTTP interface and the OSLC interface to the ObjectServer. The default is FALSE, which means that the interfaces are disabled.</td>
</tr>
<tr>
<td>NRestOS.OSLCResourceConfigFile</td>
<td>-nrestosolcrescfg</td>
<td>The path to the OSLC resource configuration file. This JSON file defines how columns from the ObjectServer schema are mapped to properties in the OSLC event domain. The default is <code>$OMNIHOME/etc/restos/resourcecfg.json</code>.</td>
</tr>
</tbody>
</table>
The following table lists the ObjectServer properties that control the embedded HTTP server.

Table 3. Properties and command-line options for controlling the embedded HTTP server

<table>
<thead>
<tr>
<th>Property</th>
<th>Command-line option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NHttpd.AccessLog string</td>
<td>-nhttpd_accesslog</td>
<td>Specifies the name and location of the log file where the server logs all requests that it processes. The default is $OMNIHOME/log/NCOMS_http_access.log.</td>
</tr>
<tr>
<td>NHttpd.Authentication Domain string</td>
<td>-nhttpd_authdomain</td>
<td>Specifies the authentication domain that is used when requesting authentication details over the HTTP or HTTPS connection. The default is omnibus.</td>
</tr>
<tr>
<td>Fix Pack 2 NHttpd.ConfigFile string</td>
<td>-nhttpd_configfile</td>
<td>Specifies the path to a JSON configuration file. The default is $OMNIHOME/etc/libnhttpd.json, which enables mimeType settings and HTTP headers in HTTP response files.</td>
</tr>
<tr>
<td>NHttpd.DocumentRoot string</td>
<td>-nhttpd_docroot</td>
<td>Specifies the document root of the embedded web service. The default is $OMNIHOME/etc/restos/docroot.</td>
</tr>
<tr>
<td>NHttpd.EnableFileServing TRUE</td>
<td>-nhttpd_enablefs</td>
<td>Use this property to enable default file serving by the ObjectServer. This allows the ObjectServer to act as a simple HTTP server that serves files from the local filesystem. The default is FALSE.</td>
</tr>
<tr>
<td>NHttpd.ExpireTimeout unsigned</td>
<td>-nhttpd_exptimeout</td>
<td>Specifies the maximum time, in seconds, that an HTTP 1.1 connection remains idle before it is dropped. The default is 15.</td>
</tr>
</tbody>
</table>
Table 3. Properties and command-line options for controlling the embedded HTTP server (continued)

<table>
<thead>
<tr>
<th>Property</th>
<th>Command-line option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NHttpd.ListeningHostname</td>
<td>-nhttpd_hostname</td>
<td>Specifies the listening host name or IP address that can be used as the hostname part of a URI to the ObjectServer HTTP or HTTPS interface. The default is localhost.</td>
</tr>
<tr>
<td>integer</td>
<td>-nhttpd_sslport</td>
<td>Specifies the port on which the ObjectServer listens for HTTPS requests. The default is 0.</td>
</tr>
<tr>
<td>string</td>
<td>-nhttpd_sslcert</td>
<td>Specifies the name of the SSL certificate of the server. The default is ''.</td>
</tr>
<tr>
<td>string</td>
<td>-nhttpd_sslcertpwd</td>
<td>Specifies the password required to access the SSL certificate file. The default is ''.</td>
</tr>
<tr>
<td>TRUE</td>
<td>-nhttpd_sslenable</td>
<td>Enables the use of SSL support. The default is FALSE.</td>
</tr>
<tr>
<td>FALSE</td>
<td>-nhttpd_sslenable</td>
<td></td>
</tr>
<tr>
<td>Fix Pack 2</td>
<td>nrestososlcrmf</td>
<td>Set this property to the string MIGRATION to redevelop any utilities that are based on the ObjectServer OSLC interface so that the members resource reference list is generated in a Collection resource instance instead of a ResponseInfo resource instance in the RDF/XML payload of the Event, Detail, and Journal query capability. The MIGRATION setting means that the members resource reference list is generated in both a Collection and a ResponseInfo resource instance. Redevelop your OSLC utilities to generate the members resource reference list only in the Collection resource instance. After the code that generates the list in ResponseInfo resource instance is removed, reset this property.</td>
</tr>
</tbody>
</table>
Enabling and configuring the IBM JazzSM service provider registry

If your environment uses Jazz for Service Management (JazzSM), you can configure the ObjectServer to register with the JazzSM service provider registry. The ObjectServer is registered as an event OSLC service provider. Registrations to JazzSM registries are configured and managed by the OSLC service provider registry table, registry.oslcsp.

About this task

Access to this table is granted only to the root user and administrators that have the OSLCAdmin role. Registrations cannot be updated. Registration records can be only inserted and deleted, not updated.

Procedure

- To create a registration, insert a registration entry into the registry.oslcsp table. The following example shows a sample SQL INSERT command for the JazzSM service provider registry that runs on the host jazzsm.company.com, on port 9080, with the default credentials:

  ```sql
  INSERT INTO registry.oslcsp (Name, RegistryURI, RegistryUsername, RegistryPassword)
  ```

  After the insert is made, the ObjectServer attempts to register the OSLC interface of the local ObjectServer with the defined JazzSM service provider registry. If the registration is successful, the registration URI that was created is stored in the RegistrationURI field. The Registered field is set to 1. If the registration is not successful, the Registered field is set to 0.

- To remove a registration from a JazzSM service provider registry, delete the registration entry from the table. For example, to remove the registration that is shown in the previous example, use the SQL DELETE command that is shown in the following example:

  ```sql
  DELETE FROM registry.oslcsp WHERE Name='MyRegistration';
  ```

  If the registration record contains a registration URI that is registered with the defined JazzSM service provider registry, the ObjectServer deletes the record after you delete the row from the table.

What to do next

If a registration fails, see the ObjectServer log file.
**registry.oslc table**

This table is used to configure and manage registrations of OSLC service providers to IBM® JazzSM service registries.

*Table 4. OSLC service provider registration table registry.oslcsp.*

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>VARCHAR(64)</td>
<td>A user-defined name for the registration table entry.</td>
</tr>
<tr>
<td>RegistryURI</td>
<td>VARCHAR(1024)</td>
<td>The OSLC service provider services record of the registry service. RegistryURI is the primary key of the table.</td>
</tr>
<tr>
<td>RegistryUsername</td>
<td>VARCHAR(64)</td>
<td>The user that is used to authenticate with the JazzSM service provider registry.</td>
</tr>
<tr>
<td>RegistryPassword</td>
<td>VARCHAR(64)</td>
<td>The password that is used to authenticate with the JazzSM service provider registry.</td>
</tr>
<tr>
<td>Registered</td>
<td>integer</td>
<td>Indicates whether the entry has a registration record with the JazzSM registry service. Possible values are as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 0: The entry does not have a registration.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1: The entry has a registration.</td>
</tr>
<tr>
<td>RegistrationURI</td>
<td>VARCHAR(1024)</td>
<td>The URI of the registration record in the JazzSM service provider registry for this ObjectServer.</td>
</tr>
<tr>
<td>LastRegistered</td>
<td>time</td>
<td>The date and time of the last successful registration to the JazzSM service provider registry.</td>
</tr>
</tbody>
</table>
Chapter 2. HTTP interface URIs

The ObjectServer HTTP interface includes URIs that give access to table data and to rows in tables, execute SQL commands via HTTP, and give access to system information.

Table collection services

Use the table collection services URI to access any table in the ObjectServer data store, such as system or user tables. The table collection services URI is the top-level URI.

The format of this URI is as follows.

http://host:port/objectserver/restapi/database/table

Where database is the name of the ObjectServer database, and table is the name of the table in that database.

The table collection services URI supports the following HTTP methods: GET, POST, PATCH, and DELETE.

Rows in the referenced table can be fetched, updated, or deleted by a single request. A row can also be inserted into the table by a POST request. Only a single row can be inserted. Bulk insertion is not possible.

Example table collection URIs are as follows:

- http://localhost/objectserver/restapi/alerts/status
- http://localhost/objectserver/restapi/catalog/tables
- http://localhost/objectserver/restapi/alerts/conversions

Table collection services: GET request

The elements of an HTTP GET request to a table collection to retrieve rows from an ObjectServer table.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP method</td>
<td>GET</td>
</tr>
</tbody>
</table>
Table 5. Table collection services: GET request (continued)

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query parameters</td>
<td></td>
</tr>
<tr>
<td>filter</td>
<td>Defines the conditions that a row in the table must satisfy. This parameter is the WHERE clause of an SQL SELECT statement.</td>
</tr>
<tr>
<td>collist</td>
<td>Defines the columns of the table in the results of the HTTP response. This parameter is the column list component of an SQL SELECT statement.</td>
</tr>
<tr>
<td>orderby</td>
<td>Defines the sort order of the result set. This parameter is the ORDER BY clause of an SQL SELECT statement.</td>
</tr>
</tbody>
</table>

Request headers

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization</td>
<td>Required</td>
</tr>
<tr>
<td>Host</td>
<td>Required</td>
</tr>
<tr>
<td>Accept</td>
<td>application/json</td>
</tr>
<tr>
<td>Request body</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

Related reference:

“Authentication mechanisms” on page 31

Any connection to the ObjectServer HTTP interface needs a set of Tivoli Netcool/OMNIbus user credentials for authentication. Only basic HTTP authentication is supported.

“Example table collection GET request” on page 40

Select all rows from the alerts.status table.

Table collection services: GET response

The elements of an HTTP GET response to a table collection in an ObjectServer.

Table 6. Table collection services: GET response

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response headers</td>
<td></td>
</tr>
<tr>
<td>Server</td>
<td>The name of the HTTPd engine.</td>
</tr>
<tr>
<td>Date</td>
<td>The date or time of the response.</td>
</tr>
<tr>
<td>Connection</td>
<td>The connection state of the connection. Possible states are Close or Keep-Alive.</td>
</tr>
<tr>
<td>Content-Type</td>
<td>application/json</td>
</tr>
<tr>
<td>Normal HTTP response codes</td>
<td>200 (Created): The response body contains a JSON success message.</td>
</tr>
</tbody>
</table>
Table 6. Table collection services: GET response (continued)

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error HTTP response codes</td>
<td>500 (Internal Server Error): The server failed to complete the request due to an unexpected internal problem. The response body contains the JSON error message with more information. Other common HTTP error response codes are 401 (Unauthorized), 403 (Forbidden), and 406 (Not Acceptable).</td>
</tr>
</tbody>
</table>

Related reference:
“HTTP response codes” on page 29
The common set of HTTP response codes for an HTTP method from the ObjectServer HTTP interface.
“Error JSON message payload” on page 32
The ObjectServer HTTP interface might return a JSON error message payload in any nonsuccess response code, such as 500. This error message gives information about the internal ObjectServer return code failure that is related to the request.
“Example table collection GET response” on page 40

Table collection services: POST request

Table 7. Table collection services: POST request

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP method</td>
<td>POST</td>
</tr>
<tr>
<td>Query parameters</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>
| Request headers | Authorization  
|                 | Required     
|                 | Host         
|                 | Required     |
| Accept          | application/json |
| Content-Type    | application/json |
| Request body    | JSON row set containing a single row to insert. Bulk insert is not supported. |

Related reference:
“Example JSON row set: POST” on page 37
This example JSON row set defines a row to be inserted into the alerts.status table of the ObjectServer.
“Example table collection POST request” on page 39
Insertion of a row into the alerts.status table.
**Table collection services: POST response**

The elements of an HTTP POST response for the creation of a row within a table collection in the ObjectServer.

*Table 8. Table collection services: POST response*

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Response headers</strong></td>
<td></td>
</tr>
<tr>
<td>Server</td>
<td>The name of the HTTPd engine.</td>
</tr>
<tr>
<td>Date</td>
<td>The date or time of the response.</td>
</tr>
<tr>
<td>Connection</td>
<td>The state of the connection. Possible states are Close or Keep-Alive.</td>
</tr>
<tr>
<td>Location</td>
<td>The URI of the created resource.</td>
</tr>
<tr>
<td>Content-Type</td>
<td>application/json</td>
</tr>
<tr>
<td><strong>Normal HTTP response codes</strong></td>
<td>201 (Created): The URI of the inserted row is contained in the HTTP header Location of the response. The response body contains a JSON success message.</td>
</tr>
<tr>
<td><strong>Error HTTP response codes</strong></td>
<td>400 (Bad Request): The JSON row set definition of the row to insert is invalid. The row is invalid because it contains missing, unknown or incorrect columns, or because insufficient or incorrect column values are provided. 409 (Conflict): Primary key collision. The row already exists in the table. 500 (Internal Server Error): The server failed to complete the request due to an unexpected internal problem. The response body contains the JSON error message with more information. Other common HTTP error response codes are 401 (Unauthorised), 403 (Forbidden)406 (Not acceptable), and 415 (Unsupported Media Type).</td>
</tr>
</tbody>
</table>
Table collection services: PATCH request

The elements of an HTTP PATCH request to a table collection to update rows in an ObjectServer table.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP method</td>
<td>PATCH</td>
</tr>
<tr>
<td>Query parameters</td>
<td></td>
</tr>
<tr>
<td>filter</td>
<td>Defines the conditions that a row in the table must satisfy. This parameter is the WHERE clause of an SQL SELECT statement.</td>
</tr>
<tr>
<td>Request headers</td>
<td></td>
</tr>
<tr>
<td>Authorization</td>
<td>Required</td>
</tr>
<tr>
<td>Host</td>
<td>Required</td>
</tr>
<tr>
<td>Accept</td>
<td>application/json</td>
</tr>
<tr>
<td>Request body</td>
<td>JSON row set that contains a single row. The row defines which columns to update, and which values to update the columns to.</td>
</tr>
</tbody>
</table>

Related reference:
“HTTP response codes” on page 29
The common set of HTTP response codes for an HTTP method from the ObjectServer HTTP interface.
“Example JSON row set: PATCH” on page 36
This example updates the Location, LastOccurrence, Acknowledged, OwnerUID and OwnerGID columns of the matched rows in the alerts.status table.
“Example table collection PATCH request” on page 42
Update the Location, LastOccurrence, Acknowledged, OwnerUID and OwnerGID columns of all rows in the alerts.status table.
**Table collection services: PATCH response**

The elements of an HTTP PATCH response for the update of rows in a table collection in the ObjectServer.

*Table 10. Table collection services: PATCH response*

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| **Response headers**          | **Server**
                                 | The name of the HTTPd engine.                                                                                                                                   |
|                               | **Date**
                                 | The date or time of the response.                                                                                                                                 |
|                               | **Connection**
                                 | The state of the connection. Possible states are Close or Keep-Alive.                                                                                     |
| **Content-Type**              | application/json                                                                                                                                             |
| **Normal HTTP response codes**| 200 (OK): The response body contains a JSON success message.                                                                                              |
| **Error HTTP response codes** | 400 (Bad Request): The JSON row set definition of the row to insert is invalid. The row is invalid because it contains missing, unknown or incorrect columns, or because insufficient or incorrect column values are provided.  |
|                               | 500 (Internal Server Error): The server failed to complete the request due to an unexpected internal problem. The response body contains the JSON error message with more information. |

Other common HTTP error response codes are 401 (Unauthorised), 403 (Forbidden), 406 (Not acceptable), and 415 (Unsupported Media Type).

**Related reference:**

- [“HTTP response codes” on page 29](#)
  The common set of HTTP response codes for an HTTP method from the ObjectServer HTTP interface.
- [“Success JSON message payload” on page 32](#)
  The ObjectServer HTTP interface returns a JSON success message in the payload. The HTTP response code informs the requester whether the request was successful, and the JSON message gives additional information. This information includes the number of rows in a table collection that were affected by the request.
- [“Error JSON message payload” on page 32](#)
  The ObjectServer HTTP interface might return a JSON error message payload in any nonsuccess response code, such as 500. This error message gives information about the internal ObjectServer return code failure that is related to the request.
- [“Example table collection PATCH response” on page 42](#)
Table collection services: DELETE request

The elements of an HTTP DELETE request to a table collection to delete rows in an ObjectServer table.

Table 11. Table collection services: DELETE request

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP method</td>
<td></td>
</tr>
<tr>
<td>Query parameters</td>
<td></td>
</tr>
<tr>
<td>filter</td>
<td>Defines the conditions that a row in the table must satisfy. This parameter is the WHERE clause of an SQL SELECT statement.</td>
</tr>
<tr>
<td>Request headers</td>
<td></td>
</tr>
<tr>
<td>Authorization</td>
<td>Required</td>
</tr>
<tr>
<td>Host</td>
<td>Required</td>
</tr>
<tr>
<td>Accept</td>
<td>application/json</td>
</tr>
<tr>
<td>Request body</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

Related reference:

“Authentication mechanisms” on page 31

Any connection to the ObjectServer HTTP interface needs a set of Tivoli Netcool/OMNibus user credentials for authentication. Only basic HTTP authentication is supported.

“Example table collection DELETE request” on page 43

Delete all rows in the alerts.status table.

Table collection services: DELETE response

The elements of an HTTP DELETE response for the deletion of rows in a table collection in the ObjectServer.

Table 12. Table collection services: DELETE response

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response headers</td>
<td></td>
</tr>
<tr>
<td>Server</td>
<td>The name of the HTTPd engine.</td>
</tr>
<tr>
<td>Date</td>
<td>The date or time of the response.</td>
</tr>
<tr>
<td>Connection</td>
<td>The state of the connection. Possible states are Close or Keep-Alive.</td>
</tr>
<tr>
<td>Content-Type</td>
<td>application/json</td>
</tr>
<tr>
<td>Normal HTTP response codes</td>
<td>200 (Created): The response body contains a JSON success message.</td>
</tr>
</tbody>
</table>
Table 12. Table collection services: DELETE response (continued)

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error HTTP response codes</td>
<td>400 (Bad Request): The JSON row set definition of the row to insert is invalid. The row is invalid because it contains missing, unknown or incorrect columns, or because insufficient or incorrect column values are provided. 500 (Internal Server Error): The server failed to complete the request due to an unexpected internal problem. The response body contains the JSON error message with more information. Other common HTTP error response codes are 401 (Unauthorised), 403 (Forbidden) 406 (Not acceptable), and 415 (Unsupported Media Type).</td>
</tr>
</tbody>
</table>

Related reference:
- “HTTP response codes” on page 29
  The common set of HTTP response codes for an HTTP method from the ObjectServer HTTP interface.
- “Success JSON message payload” on page 32
  The ObjectServer HTTP interface returns a JSON success message in the payload. The HTTP response code informs the requester whether the request was successful, and the JSON message gives additional information. This information includes the number of rows in a table collection that were affected by the request.
- “Error JSON message payload” on page 32
  The ObjectServer HTTP interface might return a JSON error message payload in any nonsuccess response code, such as 500. This error message gives information about the internal ObjectServer return code failure that is related to the request.
- “Example table collection DELETE response” on page 43

Row element services

Use this URI to access and reference specific rows in ObjectServer tables.

The following methods are supported.

**RowSerial**

Every row in every table in the ObjectServer has an integer value that uniquely identifies that row in the table in which the row resides. When you use an HTTP GET request to fetch rows to the table collection services URI, the RowSerial value of each row is automatically returned as a column in the JSON row set document. The RowSerial value can be used to build a row element URI to that row, as the following example shows.

http://host:port/objectserver/restapi/database/table/rowserial/

Where *database* is the name of the ObjectServer database, *table* is the name of the ObjectServer table, and *rowserial* is the unique row serial number of the row.

The RowSerial method supports the following HTTP methods: GET, PATCH, and DELETE.
Because RowSerial is an integer, it is efficient for building URIs and evaluating ObjectServers. RowSerial URIs are valid only for the ObjectServer from which they were originally determined. Access to rows via RowSerial is the preferred method.

**Key field**

Every row in every table in the ObjectServer has a key field that uniquely identifies that row in the table in which it resides. It is therefore possible to reference a row by using the key field instead of the RowSerial value, as shown in the following example.

http://host:port/objectserver/restapi/database/table/kf/keyfield/

Where `database` is the name of the ObjectServer database, `table` is the name of the ObjectServer table, and `keyfield` is the unique row serial number of the row.

The key field method supports the following HTTP methods: GET, PATCH, and DELETE.

**Notes on key field construction**

Wherever permissible, a key field URI is valid in any ObjectServer in environments in which the HTTP interface is enabled, so that you can build and use a common path to a row across multiple ObjectServers.

Key fields for rows that are valid across multiple ObjectServers differ, depending on the ObjectServer table in which they reside. For example, in the alerts.status table, the primary key is the Identifier column. Because the value of Identifier is not unique to a single event instance across multiple ObjectServers, it is not used to build the key field. Instead, the key field for rows in the alerts.status table is built from the values of the ServerSerial column and the ServerName column.

**Note:** It is complex to generate key fields, especially in tables with multiple-column primary keys, because certain characters must be encoded for use in a URI. Unless you need a URI that is valid across multiple ObjectServers, use the RowSerial URI.

The following table shows how the key field is constructed for the alerts.status, alerts.journal, alerts.details tables, and for all other general tables in the ObjectServer.
<table>
<thead>
<tr>
<th>Table</th>
<th>Type</th>
<th>How the key field is constructed</th>
</tr>
</thead>
</table>
| alerts.status | Event    | The ServerSerial field is combined with the ServerName field and separated by a colon. The following example shows how the fields are combined. 1234:NCOMS  

The following example shows how the field is constructed after it is encoded for use in a URI. 1234%3ANCOMS

The following example shows a sample URI. http://localhost/objectserver/restapi/alerts/status/kf/12510%3ANCOMS |
| alerts.journal| Journal  | The key field is defined by the KeyField column of the journal table. Because journals are reserialized differently in different ObjectServers, a journal key field is not valid across multiple ObjectServers. The following example shows a sample KeyField. 2684:0:1341416084  

The following example shows how the field is constructed after it is encoded for use in a URI. 12684%3A0%3A1341416084

The following example shows a sample URI. http://localhost/objectserver/restapi/alerts/journal/kf/12684%3A0%3A1341416084 |
| alerts.details| Details  | The key field is defined by the KeyField column of the details table. The following example shows a sample KeyField. EventIdentifier@@@@4####4  

The following example shows how the field is constructed after it is encoded for use in a URI. EventIndentifier%40%40%40%40%23%23%23%23

The following example shows a sample URI. http://localhost/objectserver/restapi/alerts/details/kf/EventIndentifier%40%40%40%40%23%23%23%23 |
Table 13. Construction of key field values for row elements  (continued)

<table>
<thead>
<tr>
<th>Table</th>
<th>Type</th>
<th>How the key field is constructed</th>
</tr>
</thead>
<tbody>
<tr>
<td>All other general tables in the ObjectServer</td>
<td>General</td>
<td>The key field is the set of columns that make up the primary key of the table. If the primary key of the table consists of multiple keys, specify the key values in the same order as in the table schema. Separate the key values with the sequence #KF#. The following example shows a sample key field in a table that has two primary columns, with the values ColValue01 and 654. ColValue01#KF#654 The following example shows how this field is constructed after it is encoded for use in a URI ColValue01%23KF%23654 The following example shows this key field in a URI <a href="http://localhost/objectserver/restapi/alerts/mytable/kf/ColValue01%23KF%23654">http://localhost/objectserver/restapi/alerts/mytable/kf/ColValue01%23KF%23654</a></td>
</tr>
</tbody>
</table>

Row element GET request

The elements of an HTTP GET request to a row element to retrieve a specific row from an ObjectServer table.

Table 14. Row element: GET request

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP method</td>
<td>GET</td>
</tr>
<tr>
<td>Query parameters</td>
<td><strong>collist</strong></td>
</tr>
<tr>
<td>Request headers</td>
<td>Authorization</td>
</tr>
<tr>
<td></td>
<td>Host</td>
</tr>
<tr>
<td>Accept</td>
<td>application/json</td>
</tr>
<tr>
<td>Request body</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>
Any connection to the ObjectServer HTTP interface needs a set of Tivoli Netcool/OMNibus user credentials for authentication. Only basic HTTP authentication is supported.

```
Row element GET response
```

The elements of an HTTP GET response for the retrieval of a specific row from an ObjectServer table.

Table 15. Row element: GET response

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response headers</td>
<td></td>
</tr>
<tr>
<td>Server</td>
<td>The name of the HTTPd engine.</td>
</tr>
<tr>
<td>Date</td>
<td>The date or time of the response.</td>
</tr>
<tr>
<td>Connection</td>
<td>The state of the connection. Possible states are Close or Keep-Alive.</td>
</tr>
<tr>
<td>Content-Type</td>
<td>application/json</td>
</tr>
<tr>
<td>Normal HTTP response codes</td>
<td></td>
</tr>
<tr>
<td>200 (Created)</td>
<td>The response body contains a JSON success message.</td>
</tr>
<tr>
<td>Error HTTP response codes</td>
<td></td>
</tr>
<tr>
<td>404 (Not Found)</td>
<td>The requested row was not found in the table because the row was deleted.</td>
</tr>
<tr>
<td>500 (Internal Server Error)</td>
<td>The server failed to complete the request due to an unexpected internal problem. The response body contains the JSON error message with more information.</td>
</tr>
<tr>
<td>Common HTTP error response codes</td>
<td>401 (Unauthorized), 403 (Forbidden), and 406 (Not Acceptable).</td>
</tr>
</tbody>
</table>
Row element PATCH request

The elements of an HTTP PATCH request to a specific row element in an ObjectServer table.

Table 16. Row element: PATCH request

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP method</td>
<td>PATCH</td>
</tr>
<tr>
<td>Query parameters</td>
<td>Not applicable</td>
</tr>
<tr>
<td>Request headers</td>
<td>Authorization: Required Host: Required</td>
</tr>
<tr>
<td>Accept</td>
<td>application/json</td>
</tr>
<tr>
<td>Request body</td>
<td>A JSON row set that contains a single row. The row defines which columns in the row to update.</td>
</tr>
</tbody>
</table>

Related reference:
[“Authentication mechanisms” on page 31](#)

Any connection to the ObjectServer HTTP interface needs a set of Tivoli Netcool/OMNIbus user credentials for authentication. Only basic HTTP authentication is supported.

[“Example JSON row set: PATCH” on page 36](#)

This example updates the Location, LastOccurrence, Acknowledged, OwnerUID and OwnerGID columns of the matched rows in the alerts.status table.

[“Example row element PATCH request” on page 47](#)

Update the Location, LastOccurrence, Acknowledged, OwnerUID and OwnerGID columns of a specific row in the alerts.status table.

Row element PATCH response

The elements of an HTTP PATCH response for the update of a specific row in a table of the ObjectServer.

Table 17. Row element: PATCH response

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response headers</td>
<td>Server: The name of the HTTPd engine. Date: The date or time of the response. Connection: The state of the connection. Possible states are Close or Keep-Alive.</td>
</tr>
<tr>
<td>Content-Type</td>
<td>application/json</td>
</tr>
<tr>
<td>Normal HTTP response codes</td>
<td>200 (Created): The response body contains a JSON success message.</td>
</tr>
</tbody>
</table>
Table 17. Row element: PATCH response (continued)

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| Error HTTP response codes        | 400 (Bad Request): The JSON row set definition of the row to insert is invalid. The row is invalid because it contains missing, unknown or incorrect columns, or because insufficient or incorrect column values are provided.  
|                                  | 404 (Not Found): The requested row was not found in the table because the row was deleted.                                              
|                                  | 500 (Internal Server Error): The server failed to complete the request due to an unexpected internal problem. The response body contains the JSON error message with more information. |
| Other common HTTP error response codes | are 401 (Unauthorised), 403 (Forbidden) 406 (Not acceptable), and 415 (Unsupported Media Type).                                            |

Related reference:
- "HTTP response codes" on page 29
- "Success JSON message payload" on page 32
- "Error JSON message payload" on page 32

Row element DELETE request

The elements of an HTTP DELETE request to a specific row element in an ObjectServer table.

Table 18. Row element: DELETE request

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP method</td>
<td>DELETE</td>
</tr>
<tr>
<td>Query parameters</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>
| Request headers | Authorization  
|                | Required  
|                | Host       
|                | Required  
| Accept        | application/json  
| Request body  | Not applicable |
Any connection to the ObjectServer HTTP interface needs a set of Tivoli Netcool/OMNIbus user credentials for authentication. Only basic HTTP authentication is supported.

Row element DELETE response

The elements of an HTTP DELETE response for the deletion of a specific row in a table of the ObjectServer.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response headers</td>
<td></td>
</tr>
<tr>
<td>Server</td>
<td>The name of the HTTPd engine.</td>
</tr>
<tr>
<td>Date</td>
<td>The date or time of the response.</td>
</tr>
<tr>
<td>Connection</td>
<td>The state of the connection. Possible states are Close or Keep-Alive.</td>
</tr>
<tr>
<td>Content-Type</td>
<td>application/json</td>
</tr>
<tr>
<td>Normal HTTP response codes</td>
<td>200 (Created): The response body contains a JSON success message.</td>
</tr>
<tr>
<td>Error HTTP response codes</td>
<td>404 (Not Found): The requested row was not found in the table because the row was deleted.</td>
</tr>
<tr>
<td></td>
<td>500 (Internal Server Error): The server failed to complete the request due to an unexpected internal problem. The response body contains the JSON error message with more information.</td>
</tr>
<tr>
<td></td>
<td>Other common HTTP error response codes are 401 (Unauthorized), 403 (Forbidden) 406 (Not acceptable), and 415 (Unsupported Media Type).</td>
</tr>
</tbody>
</table>
Related reference:

“HTTP response codes” on page 29
The common set of HTTP response codes for an HTTP method from the ObjectServer HTTP interface.

“Success JSON message payload” on page 32
The ObjectServer HTTP interface returns a JSON success message in the payload.
The HTTP response code informs the requester whether the request was successful, and the JSON message gives additional information. This information includes the number of rows in a table collection that were affected by the request.

“Error JSON message payload” on page 32
The ObjectServer HTTP interface might return a JSON error message payload in any nonsuccess response code, such as 500. This error message gives information about the internal ObjectServer return code failure that is related to the request.

“Example row element DELETE response” on page 49

SQL command factory

Use the SQL command factory of the ObjectServer HTTP interface to execute arbitrary SQL commands via HTTP.

To execute SQL through this interface, users need the AllowISQL user permission for read SQL commands, and the AllowISQLWrite user permission for write SQL commands.

For more information about user permissions, see the IBM Tivoli Netcool/OMNibus Administration Guide.

To execute a SQL command, post a JSON SQL command message to the SQL command factory URI, as shown in the following example.
http://host:port/objectserver/restapi/sql/factory

The SQL command factory supports the following HTTP methods: POST.

**SQL command factory POST request**

The requirements for a POST request to SQL command factory to execute arbitrary SQL commands in the ObjectServer.

*Table 20. SQL command factory: POST request*

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP method</td>
<td>POST</td>
</tr>
<tr>
<td>Query parameters</td>
<td>Not applicable</td>
</tr>
<tr>
<td>Request headers</td>
<td></td>
</tr>
<tr>
<td>Authorization</td>
<td>Required</td>
</tr>
<tr>
<td>Host</td>
<td>Required</td>
</tr>
<tr>
<td>Accept</td>
<td>application/json</td>
</tr>
<tr>
<td>Content-Type</td>
<td>application/json</td>
</tr>
<tr>
<td>Request body</td>
<td>JSON SQL command message</td>
</tr>
</tbody>
</table>
Any connection to the ObjectServer HTTP interface needs a set of Tivoli Netcool/OMNIbus user credentials for authentication. Only basic HTTP authentication is supported.

This example SQL command factory message issues a drop user command for execution.

### SQL command factory POST response

The elements of an HTTP POST response for the execution of arbitrary SQL commands in the ObjectServer.

**Table 21. SQL command factory: POST response**

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response headers</td>
<td></td>
</tr>
<tr>
<td><strong>Server</strong></td>
<td>The name of the HTTPd engine.</td>
</tr>
<tr>
<td><strong>Date</strong></td>
<td>The date or time of the response.</td>
</tr>
<tr>
<td><strong>Connection</strong></td>
<td>The state of the connection. Possible states are Close or Keep-Alive.</td>
</tr>
<tr>
<td>Content-Type</td>
<td>application/json</td>
</tr>
<tr>
<td>Normal HTTP response codes</td>
<td>200 (Created): The response body contains a JSON success message.</td>
</tr>
<tr>
<td>Error HTTP response codes</td>
<td>400 (Bad Request): The JSON row set definition of the row to insert is invalid. The row is invalid because it contains missing, unknown or incorrect columns, or because insufficient or incorrect column values are provided. 500 (Internal Server Error): The server failed to complete the request due to an unexpected internal problem. The response body contains the JSON error message with more information. Other common HTTP error response codes are 401 (Unauthorized), 403 (Forbidden), 406 (Not acceptable), and 415 (Unsupported Media Type).</td>
</tr>
</tbody>
</table>
Related reference:

"HTTP response codes" on page 29
The common set of HTTP response codes for an HTTP method from the ObjectServer HTTP interface.

"Success JSON message payload" on page 32
The ObjectServer HTTP interface returns a JSON success message in the payload. The HTTP response code informs the requester whether the request was successful, and the JSON message gives additional information. This information includes the number of rows in a table collection that were affected by the request.

"Error JSON message payload" on page 32
The ObjectServer HTTP interface might return a JSON error message payload in any nonsuccess response code, such as 500. This error message gives information about the internal ObjectServer return code failure that is related to the request.

"Example SQL command factory POST response" on page 49

System information services

Use this URI to query and determine the compilation details and versions of the HTTP interface and the OSLC interface in the ObjectServer.

All available system information elements can be obtained by issuing an HTTP GET request to the system information services URI, as shown as the following example.

http://host:port/objectserver/restapi/sysinfo

The system information elements can be fetched individually by referring to the required system information element in the URI, as shown in the following example.

http://host:port/objectserver/restapi/sysinfo/element

Where element can be one of the following system information elements.

compile
   Gets the detailed build information of the HTTP interfaces that are hosted in the ObjectServer.

rest
   Gets version information about the HTTP interface of the ObjectServer.

oslc
   Gets version information about the OSLC interface of the ObjectServer.

The system information service URI supports the following HTTP methods: GET.

GET collection request

The elements of an HTTP GET request to the system information collection URI to retrieve all system information from an ObjectServer.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP method</td>
<td>GET</td>
</tr>
<tr>
<td>Query parameters</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>
Table 22. System information collection services: GET request (continued)

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request headers</td>
<td>Authorization</td>
</tr>
<tr>
<td></td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td>Host</td>
</tr>
<tr>
<td></td>
<td>Required</td>
</tr>
<tr>
<td>Accept</td>
<td>application/json</td>
</tr>
<tr>
<td>Request body</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

Related reference:
“Authentication mechanisms” on page 31
Any connection to the ObjectServer HTTP interface needs a set of Tivoli Netcool/OMNIbus user credentials for authentication. Only basic HTTP authentication is supported.
“Example system information GET request” on page 49

GET collection response
The elements of an HTTP GET response for the retrieval of all ObjectServer system information.

Table 23. System information collection services: GET response

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
</table>
| Response headers         | Server
|                          | The name of the HTTPd engine.                    |
|                          | Date
|                          | The date or time of the response.                |
|                          | Connection
|                          | The state of the connection. Possible states are Close or Keep-Alive. |
| Content-Type             | application/json                                |
| Normal HTTP response codes | 200 (Created): The response body contains a JSON success message. |
| Error HTTP response codes | 500 (Internal Server Error): The server failed to complete the request due to an unexpected internal problem. The response body contains the JSON error message with more information. |
|                          | Other common HTTP error response codes are 401 (Unauthorised), 403 (Forbidden), and 406 (Not acceptable). |
Related reference:
“HTTP response codes” on page 29
The common set of HTTP response codes for an HTTP method from the ObjectServer HTTP interface.
“Error JSON message payload” on page 32
The ObjectServer HTTP interface might return a JSON error message payload in any nonsuccess response code, such as 500. This error message gives information about the internal ObjectServer return code failure that is related to the request.
“Example system information GET response” on page 50

GET element request
The elements of an HTTP GET request to retrieve a specific system information element from an ObjectServer.

Table 24. System element information services: GET request

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP method</td>
<td>GET</td>
</tr>
<tr>
<td>Query parameters</td>
<td>Not applicable</td>
</tr>
<tr>
<td>Request headers</td>
<td>Authorization Required</td>
</tr>
<tr>
<td></td>
<td>Host Required</td>
</tr>
<tr>
<td>Accept</td>
<td>application/json</td>
</tr>
<tr>
<td>Request body</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

Related reference:
“Authentication mechanisms” on page 31
Any connection to the ObjectServer HTTP interface needs a set of Tivoli Netcool/OMNIbus user credentials for authentication. Only basic HTTP authentication is supported.
“Example system information element GET request” on page 50

GET element response
The elements of an HTTP GET response for the retrieval of a specific system information element from an ObjectServer.

Table 25. System element information services: GET response

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response headers</td>
<td>Server The name of the HTTPd engine.</td>
</tr>
<tr>
<td></td>
<td>Date The date or time of the response.</td>
</tr>
<tr>
<td></td>
<td>Connection The state of the connection. Possible states are Close or Keep-Alive.</td>
</tr>
<tr>
<td>Content-Type</td>
<td>application/json</td>
</tr>
<tr>
<td>Normal HTTP response codes</td>
<td>200 (Created): The response body contains a JSON success message.</td>
</tr>
</tbody>
</table>
Table 25. System element information services: GET response (continued)

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error HTTP response codes</td>
<td>500 (Internal Server Error): The server failed to complete the request due to an unexpected internal problem. The response body contains the JSON error message with more information.</td>
</tr>
<tr>
<td></td>
<td>Other common HTTP error response codes are 401 (Unauthorized), 403 (Forbidden), and 406 (Not acceptable).</td>
</tr>
</tbody>
</table>

Related reference:

“HTTP response codes” on page 29
The common set of HTTP response codes for an HTTP method from the ObjectServer HTTP interface.

“Error JSON message payload” on page 32
The ObjectServer HTTP interface might return a JSON error message payload in any nonsuccess response code, such as 500. This error message gives information about the internal ObjectServer return code failure that is related to the request.

“Example system information element GET response” on page 50
Chapter 3. Common behaviors

Characteristics that are common to all requests from, and all responses to, the ObjectServer HTTP interface.

HTTP and HTTPS support

The ObjectServer HTTP interface supports HTTP or HTTPS connectivity at HTTP 1.0 or HTTP 1.1.

HTTP response codes

The common set of HTTP response codes for an HTTP method from the ObjectServer HTTP interface.

Success message codes

The following table shows the common HTTP success message codes.

Table 26. Common HTTP success message codes

<table>
<thead>
<tr>
<th>HTTP method</th>
<th>HTTP response code</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>200 (OK)</td>
<td></td>
</tr>
<tr>
<td>POST</td>
<td>201 (Created)</td>
<td>Location contains the URI for the newly created resource.</td>
</tr>
<tr>
<td>PATCH</td>
<td>200 (OK)</td>
<td></td>
</tr>
<tr>
<td>DELETE</td>
<td>200 (OK)</td>
<td></td>
</tr>
</tbody>
</table>

Error message codes

The following table shows the common HTTP error message codes.

Table 27. Common HTTP error message codes

<table>
<thead>
<tr>
<th>HTTP response code</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>Bad Request. Check the request payload and query parameters.</td>
</tr>
<tr>
<td>401</td>
<td>Not Authorized. The request does not contain valid authentication credentials.</td>
</tr>
<tr>
<td>403</td>
<td>Access to the defined resource is denied. The authentication credentials that were used to make the connection are denied access to the resources that are specified in the request.</td>
</tr>
<tr>
<td>404</td>
<td>The requested resource was not found. The request might be deleted.</td>
</tr>
<tr>
<td>406</td>
<td>The requested accept MIME type is not supported.</td>
</tr>
<tr>
<td>409</td>
<td>Conflict. An attempt was made to insert a row that already exists.</td>
</tr>
</tbody>
</table>
Table 27. Common HTTP error message codes (continued)

<table>
<thead>
<tr>
<th>HTTP response code</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>415</td>
<td>Specified content MIME type is not supported.</td>
</tr>
<tr>
<td>500</td>
<td>Internal server error. For more information, check the RDF/XML error message payload.</td>
</tr>
</tbody>
</table>

Related reference:
“Error JSON message payload” on page 32

The ObjectServer HTTP interface might return a JSON error message payload in any nonsuccess response code, such as 500. This error message gives information about the internal ObjectServer return code failure that is related to the request.

Query parameters

Syntax information about the OSLC query parameters that are supported by the HTTP interface.

Table 28. Description of the query parameters of the HTTP interface

<table>
<thead>
<tr>
<th>Query parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filter</td>
<td>This query parameter defines which rows the request acts on, when the request is made to the ObjectServer. Use the same format for this parameter as for an SQL WHERE clause. You can use this parameter only with GET, PATCH, and DELETE requests to an ObjectServer table collection URI. The following example shows a sample filter parameter. Node='hostname.domain' The following example shows the same parameter after it is encoded for use in a URL. Node%3D%27hostname.domain%27 The following example shows a sample URI that contains this parameter. <a href="http://localhost/objectserver/restapi/alerts/status?filter=Node%3D%27hostname.domain%27">http://localhost/objectserver/restapi/alerts/status?filter=Node%3D%27hostname.domain%27</a></td>
</tr>
<tr>
<td>collist</td>
<td>This query parameter is valid only on an HTTP GET request. The parameter defines which columns of the table to return in the JSON row set message. To define the value of the parameter, use the same syntax as the column list component of a SQL SELECT statement. The following example shows a sample collist parameter. Serial,Node,Summary The following example shows the same parameter after it is encoded for use in a URL. Serial%2CNode%2CSummary The following example shows a sample URI that contains this parameter. <a href="http://localhost/objectserver/restapi/alerts/status?collist=Serial%2CNode%2CSummary">http://localhost/objectserver/restapi/alerts/status?collist=Serial%2CNode%2CSummary</a></td>
</tr>
</tbody>
</table>
Table 28. Description of the query parameters of the HTTP interface (continued)

<table>
<thead>
<tr>
<th>Query parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| **orderby**     | This query parameter is valid only on an HTTP GET request to an ObjectServer table collection URI. The parameter defines the order of rows in the JSON row set message that is returned. To define the value of the parameter, use the same syntax as the ORDER BY clause of a SQL SELECT statement.  
  The following example shows a sample `orderby` parameter.  
  `Serial ASC`
  
  The following example shows the same parameter after it is encoded for use in a URI.  
  `Serial%20ASC`
  
  The following example shows a sample URI that contains this parameter.  
  `http://localhost/objectserver/restapi/alerts/status/?orderby=Serial%20ASC` |

The following example shows a URI that uses all the query parameters in [Table 28 on page 30](#).

```
http://localhost/objectserver/restapi/alerts/status?filter=Node%3D%27hostname.domain%27&collist=Serial%2CNode%2CSummary&Serial%20ASC&orderby=Serial%20ASC
```

**Authentication mechanisms**

Any connection to the ObjectServer HTTP interface needs a set of Tivoli Netcool/OMNIbus user credentials for authentication. Only basic HTTP authentication is supported.

If no basic HTTP credentials are provided in the HTTP header **Authorization**, a 401 (Not Authorized) HTTP response is returned.

Because basic HTTP credentials are insecure, use HTTPS to ensure that the socket communication is encrypted.

For more information about using SSL to encrypt communications, see the *IBM Tivoli Netcool/OMNIbus Installation and Deployment Guide*.

**Related reference**: 
"SQL command factory POST request" on page 22

The requirements for a POST request to SQL command factory to execute arbitrary SQL commands in the ObjectServer.
Success JSON message payload

The ObjectServer HTTP interface returns a JSON success message in the payload. The HTTP response code informs the requester whether the request was successful, and the JSON message gives additional information. This information includes the number of rows in a table collection that were affected by the request.

Table 29. Description of RDF/XML success message payload

<table>
<thead>
<tr>
<th>Name</th>
<th>Value type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>affectedRows</td>
<td>Integer</td>
<td>The number of rows affected by the request. For requests that are applied to a table collection, the value can range from 0 to many.</td>
</tr>
<tr>
<td>keyField</td>
<td>String</td>
<td>The key field ID of the a single affected resource. The key field ID is present only in successful row insertions.</td>
</tr>
<tr>
<td>uri</td>
<td>String</td>
<td>The URI of the successfully inserted row or the URI of the successful request. For a successful row insertion, this URI will match the URI in the HTTP header Location.</td>
</tr>
</tbody>
</table>

Related reference:
"Example SQL command factory: POST" on page 38
This example SQL command factory message issues a drop user command for execution.

Error JSON message payload

The ObjectServer HTTP interface might return a JSON error message payload in any nonsuccess response code, such as 500. This error message gives information about the internal ObjectServer return code failure that is related to the request.

Table 30. Description of RDF/XML error message payload

<table>
<thead>
<tr>
<th>Name</th>
<th>Value type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>statusCode</td>
<td>Integer</td>
<td>The HTTP status code that is reported with the error.</td>
</tr>
<tr>
<td>message</td>
<td>String</td>
<td>An informative message that describes the error.</td>
</tr>
<tr>
<td>moreInfo</td>
<td>String</td>
<td>More information, if available. If no more information is available, this element of the payload is not returned.</td>
</tr>
</tbody>
</table>

Related reference:
"Example JSON error message" on page 38
An example JSON error message indicating that the cause of the 400 (Bad Request) HTTP response was caused by a referenced object, such as a column not being found in the table.
Message encryption

Encryption of the message payload is not supported.

Accept MIME types

The accept MIME types supported by the ObjectServer HTTP interface.

The supported MIME types are as follows.

application/json (JSON)

Content MIME types

The content MIME types supported by the ObjectServer HTTP interface.

The supported MIME types are as follows.

application/json (JSON)

Response caching

Rows in the ObjectServer, especially event data, change constantly due to user or programmatic actions. Because the ObjectServer HTTP interface is hosted directly with the table data, there is no penalty to access the data. The interface therefore does not cache any data, such as responses, at any level. Each ObjectServer HTTP request is resolved separately each time, in the same way as any request upon the ObjectServer data from any of the interfaces of the ObjectServer.
Appendix A. Example JSON payloads

Examples of JSON message payloads.

Example JSON row set: GET

This example JSON row set is from a HTTP GET to the alerts.status table collection URI.

```json
{
    "rowset": {
        "osname": "NCOMS",
        "dbname": "alerts",
        "tblname": "status",
        "coldesc": [{
            "name": "Identifier",
            "type": "string",
            "size": 255
        }, {
            "name": "Serial",
            "type": "integer",
            "size": 4
        }, {
            "name": "Node",
            "type": "string",
            "size": 64
        }, {
            "name": "NodeAlias",
            "type": "string",
            "size": 64
        }, {
            "name": "AlertKey",
            "type": "string",
            "size": 255
        }, {
            "name": "Severity",
            "type": "integer",
            "size": 4
        }, {
            "name": "Summary",
            "type": "string",
            "size": 255
        }, {
            "name": "StateChange",
            "type": "utc",
            "size": 4
        }, {
            "name": "FirstOccurrence",
            "type": "utc",
            "size": 4
        }, {
            "name": "LastOccurrence",
            "type": "utc",
            "size": 4
        }, {
            "name": "RowSerial",
            "type": "integer",
            "size": 4
        }],
        "rows": [{
            "Identifier": "Startup@sol9-build1",
            "Serial": 12469
        }]
    }
}
```
Example JSON row set: PATCH

This example updates the Location, LastOccurrence, Acknowledged, OwnerUID and OwnerGID columns of the matched rows in the alerts.status table.

```json
{
  "rowset": {
    "coldesc": [{
      "type": "integer",
      "name": "Acknowledged"
    }, {
      "type": "string",
      "name": "Location"
    }, {
      "type": "integer",
      "name": "OwnerUID"
    }, {
      "type": "integer",
      "name": "OwnerGID"
    }, {
      "type": "utc",
      "name": "LastOccurrence"
    }],
    "rows": [{
      "Location": "UPDATED",
      "LastOccurrence": 1341412223,
      "Acknowledged": 1,
      "OwnerUID": 65534,
      "OwnerGID": 1
    }],
    "affectedRows": 0
  }
}
```
**Example JSON row set: POST**

This example JSON row set defines a row to be inserted into the alerts.status table of the ObjectServer.

```json
{
  "rowset": {
    "coldesc": [
      {"type":"string", "name":"Identifier"},
      {"type":"string", "name":"Node"},
      {"type":"string", "name":"Manager"},
      {"type":"string", "name":"Agent"},
      {"type":"string", "name":"AlertKey"},
      {"type":"integer", "name":"Severity"},
      {"type":"integer", "name":"Type"},
      {"type":"string", "name":"Summary"},
      {"type":"integer", "name":"Acknowledged"},
      {"type":"string", "name":"Location"},
      {"type":"utc", "name":"FirstOccurrence"},
      {"type":"utc", "name":"LastOccurrence"},
      {"type":"integer", "name":"OwnerUID"},
      {"type":"integer", "name":"OwnerGID"}
    ],
    "rows": [ {
      "FirstOccurrence":1341412087,
      "Node":"localhost",
      "AlertKey":"JUnitEventInstance",
      "Agent":"createEventNew()",
      "Summary":"This is a test event generated by the JUnit REST Event Tests.(0)",
      "LastOccurrence":1341412087,
      "Acknowledged":0,
      "Identifier":"JUnitEventTestInstance####0",
      "Manager":"com.ibm.netcool.ommibus.ws.junit.rest.schema.utils.TableRowEvent",
      "OwnerGID":0,
      "Location":"NOT UPDATED",
      "Type":1,
      "Severity":4,
      "OwnerUID":0
    } ]
  }
}
```

Appendix A. Example JSON payloads
Example SQL command factory: POST

This example SQL command factory message issues a drop user command for execution:

```json
{
  "sqlcmd": "drop user 'testuser01';"
}
```

Related reference:
“SQL command factory POST request” on page 22
The requirements for a POST request to SQL command factory to execute arbitrary SQL commands in the ObjectServer.

Example JSON success message

An example JSON success message for a row insertion into the alerts.status table of the ObjectServer:

```json
{
  "entry": {
    "affectedRows": 1,
    "keyField": "14382%3ANCOMS",
    "uri": "http://localhost/objectserver/restapi/alerts/status/kf/14382%3ANCOMS"
  }
}
```

Example JSON error message

An example JSON error message indicating that the cause of the 400 (Bad Request) HTTP response was caused by a referenced object, such as a column not being found in the table:

```json
{
  "exception": {
    "statusCode": 400,
    "message": "Object not found"
  }
}
```

Related reference:
“Error JSON message payload” on page 32
The ObjectServer HTTP interface might return a JSON error message payload in any nonsuccess response code, such as 500. This error message gives information about the internal ObjectServer return code failure that is related to the request.
Appendix B. HTTP request and response examples

Examples of HTTP requests and responses.

Example table collection POST request

Insertion of a row into the alerts.status table.

```
Accept: application/json
Authorization: Basic dGVzdHVzZXlXewMTpuZXRjb29s
Content-Type: application/json
Host: localhost
Connection: keep-alive
Content-Length: 984

{
  "rowset": {
    "coldesc": [ {
      "type": "string",
      "name": "Identifier"
    }, {
      "type": "string",
      "name": "Node"
    }, {
      "type": "string",
      "name": "AlertKey"
    }, {
      "type": "integer",
      "name": "Severity"
    }, {
      "type": "string",
      "name": "Summary"
    }, {
      "type": "utc",
      "name": "FirstOccurrence"
    }, {
      "type": "utc",
      "name": "LastOccurrence"
    }, {
      "type": "integer",
      "name": "OwnerUID"
    }, {
      "type": "integer",
      "name": "OwnerGID"
    }],
  "rows": [ {
    "FirstOccurrence": 1341412087,
    "Node": "localhost",
    "AlertKey": "JUnitEventInstance",
    "Summary": "This is a test event generated by the JUnit REST Event Tests.(1)",
    "LastOccurrence": 1341412087,
    "Identifier": "JUnitEventTestInstance####1",
    "OwnerGID": 0,
    "Severity": 4,
    "OwnerUID": 0
  }]
}
```
Example table collection POST response

HTTP/1.1 201 Created
Location: http://localhost/objectserver/restapi/alerts/status/kf/12481%3ANCOMS
Cache-Control: no-cache
Server: libnhttpd
Date: Wed Jul 4 15:31:53 2012
Connection: Keep-Alive
Content-Type: application/json;charset=UTF-8
Content-Length: 304

{  
  "entry": {  
    "affectedRows": 1,  
    "keyField": "12481%3ANCOMS",  
    "uri": "http://localhost/objectserver/restapi/alerts/status/kf/12481%3ANCOMS"  
  }  
}

Example table collection GET request

Select all rows from the alerts.status table.

GET /objectserver/restapi/alerts/status HTTP/1.1
Accept: application/json
Authorization: Basic dGVzdHVzZXIwMTpuZXRjb29s
Host: localhost
Connection: keep-alive

Example table collection GET response

HTTP/1.1 200 OK
Cache-Control: no-cache
Server: libnhttpd
Date: Wed Jul 4 15:32:03 2012
Connection: Keep-Alive
Content-Type: application/rdf+xml
Content-Length: 24860

{  
  "rowset": {  
    "osname": "NCOMS",  
    "dbname": "alerts",  
    "tblname": "status",  
    "coldesc": []  
  }  
}
Appendix B. HTTP request and response examples
Example table collection PATCH request

Update the Location, LastOccurrence, Acknowledged, OwnerUID and OwnerGID columns of all rows in the alerts.status table.

PATCH /objectserver/restapi/alerts/status HTTP/1.1
Accept: application/json
Authorization: Basic dGVzdHVzZXIwMTpuZXRjb29s
Content-Type: application/json
Host: localhost
Connection: keep-alive
Content-Length: 1092

```json
{
    "rowset": {
        "coldesc": [
            {
                "type": "integer",
                "name": "Acknowledged"
            },
            {
                "type": "string",
                "name": "Location"
            },
            {
                "type": "integer",
                "name": "OwnerUID"
            },
            {
                "type": "integer",
                "name": "OwnerGID"
            },
            {
                "type": "utc",
                "name": "LastOccurrence"
            }
        ],
        "rows": [
            {
                "Location": "UPDATED",
                "LastOccurrence": 1341412235,
                "Acknowledged": 1,
                "OwnerUID": 65534,
                "OwnerGID": 1
            }
        ]
    }
}
```

Example table collection PATCH response

HTTP/1.1 200 OK
Cache-Control: no-cache
Server: libnhttpd
Date: Wed Jul 4 15:32:03 2012
Connection: Keep-Alive:
Content-Type: application/json;charset=UTF-8
Content-Length: 158

```json
{
    "entry": {
```
Example table collection DELETE request
Delete all rows in the alerts.status table.
DELETE /objectserver/restapi/alerts/status HTTP/1.1
Accept: application/json
Authorization: Basic dGVzdHVzZXIwMTpuZXRjb29s
Host: localhost
Connection: keep-alive

Example table collection DELETE response
HTTP/1.1 200 OK
Cache-Control: no-cache
Server: libnhttpd
Date: Wed Jul 4 15:38:53 2012
Connection: Keep-Alive:
Content-Type: application/json;charset=UTF-8
Content-Length: 157
{
  "entry": {
    "affectedRows": 10,
    "uri": "http://localhost/objectserver/restapi/alerts/status"
  }
}

Example row element GET request via RowSerial
Accept: application/json
Authorization: Basic dGVzdHVzZXIwMTpuZXRjb29s
Host: localhost
Connection: keep-alive

Example row element GET via RowSerial response
Cache-Control: no-cache
Server: libnhttpd
Date: Wed Jul 4 15:32:03 2012
Connection: Keep-Alive:
Content-Type: application/json;charset=UTF-8
Content-Length: 5964
{
  "rowset": {
    "osname": "NCOMS",
    "dbname": "alerts",
    "tblname": "status",
    "coldesc": [
      {
        "name": "Identifier",
        "type": "string",
        "size": 255
      },
      {
        "name": "Serial",
        "type": "integer",
        "size": 4
      },
      {
        "name": "Node",
        "type": "string",
        "size": 64
      },
      {
        "name": "NodeAlias",

Appendix B. HTTP request and response examples
"type": "string",
"size": 64
},
"name": "Manager",
"type": "string",
"size": 64
},
"name": "Agent",
"type": "string",
"size": 64
},
"name": "AlertGroup",
"type": "string",
"size": 255
},
"name": "AlertKey",
"type": "string",
"size": 255
},
"name": "Severity",
"type": "integer",
"size": 4
},
"name": "Summary",
"type": "string",
"size": 255
},
"name": "StateChange",
"type": "utc",
"size": 4
},
"name": "FirstOccurrence",
"type": "utc",
"size": 4
},
"name": "LastOccurrence",
"type": "utc",
"size": 4
],
***** TRUNCATED *****
]}]
"rows": [{
"Identifier": "JUnitEventTestInstance####0",
"Serial": 12510,
"Node": "localhost",
"NodeAlias": "",
"Manager": "com.ibm.netcool.omnibus.ws.junit.rest.schema.utils.TableRowEvent",
"Agent": "createEventNew()",
"AlertGroup": "",
"AlertKey": "JUnitEventInstance",
"Severity": 4,
"Summary": "This is a test event generated by the JUnit REST Event Tests. (0)",
"StateChange": 1341412207,
"FirstOccurrence": 1341412087,
"LastOccurrence": 1341412087,
"InternalLast": 1341412207,
"Poll": 0,
"Type": 1,
"Tally": 1,
"Class": 0,
"Grade": 0,
"Location": "NOT UPDATED",
"OwnerUID": 0,
"OwnerGID": 0,
"Acknowledged": 0,
"Flash": 0,
"EventId": "",}
Example row element GET request via KeyField

Select a specific row from the alerts.status table via the key field.

GET /objectserver/restapi/alerts/status/kf/12510%3ANCOMS HTTP/1.1
Accept: application/json
Authorization: Basic dGVzdHVzZXIwMTpuZXRjb29s
Host: localhost
Connection: keep-alive

Example row element GET response via key field

HTTP/1.1 200 OK
Cache-Control: no-cache
Server: libnhttpd
Date: Wed Jul 4 15:32:03 2012
Connection: Keep-Alive:
Content-Type: application/json;charset=UTF-8
Content-Length: 5964

{ "rowset": [ { "osname": "NCOMS", "dbname": "alerts", "tblname": "status", "cldesc": [ { "name": "Identifier", "value": "12510\%3ANCOMS" } ] }, { } ], "affectedRows": 1 }
<table>
<thead>
<tr>
<th>Identifier</th>
<th>Serial</th>
<th>Node</th>
<th>NodeAlias</th>
<th>Manager</th>
<th>Agent</th>
<th>AlertGroup</th>
<th>AlertKey</th>
<th>Severity</th>
<th>Summary</th>
<th>StateChange</th>
<th>FirstOccurrence</th>
<th>LastOccurrence</th>
</tr>
</thead>
<tbody>
<tr>
<td>JUnitEventTestInstance####0</td>
<td>12510</td>
<td>localhost</td>
<td></td>
<td>com.ibm.netcool.ommibus.ws.junit.rest.schema.utils.TableRowEvent</td>
<td>createEventNew</td>
<td></td>
<td>JUnitEventInstance</td>
<td>4</td>
<td>This is a test event generated by the JUnit REST Event Tests. (0)</td>
<td>1341412207</td>
<td>1341412087</td>
<td>1341412087</td>
</tr>
</tbody>
</table>

***** TRUNCATED *****
Example row element PATCH request

Update the Location, LastOccurrence, Acknowledged, OwnerUID and OwnerGID columns of a specific row in the alerts.status table.

PATCH /objectserver/restapi/alerts/status/kf/12510%3ANCOMS HTTP/1.1
Accept: application/json
Authorization: Basic dGVzdHVzZXIwMTpuZXRjb29s
Content-Type: application/json
Host: localhost
Connection: keep-alive
Content-Length: 1092

{ "rowset": [ "coldesc": [ } ]
}

affectedRows": 1
}
Example row element PATCH response

HTTP/1.1 200 OK
Cache-Control: no-cache
Server: libnhttpd
Date: Wed Jul 4 15:32:03 2012
Connection: Keep-Alive:
Content-Type: application/json;charset=UTF-8
Content-Length: 215

{
"entry": {
"affectedRows": 1,
"uri": "/objectserver/restapi/alerts/status/kf/12510%3ANCOMS"
}
}

Example row element DELETE request

Delete a specific row in the alerts.status table.

DELETE /objectserver/restapi/alerts/status/kf/12621%3ANCOMS HTTP/1.1
Accept: application/json
Authorization: Basic dGVzdHlvdXIwMTkuZXRjb29s
Host: localhost
Connection: keep-alive
Example row element DELETE response

HTTP/1.1 200 OK
Cache-Control: no-cache
Server: libnhttpd
Date: Wed Jul 4 15:38:53 2012
Connection: Keep-Alive:
Content-Type: application/json;charset=UTF-8
Content-Length: 207
{
  "entry": {
    "affectedRows": 1,
    "uri": "/objectserver/restapi/alerts/status/kf/12621%3ANCOMS"
  }
}

Example SQL command factory POST request

POST /objectserver/restapi/sql/factory HTTP/1.1
Accept: application/json
Authorization: Basic dGVzdHVzZXIwMTpuZXRjb29s
Content-Type: application/json
Content-Length: 64
Host: localhost
Connection: keep-alive
{
  "sqlcmd":"drop user 'testuser01';"
}

Related reference:
[“SQL command factory POST request” on page 22](#)

The requirements for a POST request to SQL command factory to execute arbitrary SQL commands in the ObjectServer.

Example SQL command factory POST response

HTTP/1.1 200 OK
Cache-Control: no-cache
Server: libnhttpd
Date: Wed Jul 4 15:38:53 2012
Connection: Keep-Alive
Content-Type: application/json;charset=UTF-8
Content-Length: 124
{
  "rowset": {
    "osname": "NCOMS",
    "affectedRows": 0
  }
}

Example system information GET request

GET /objectserver/sysinfo HTTP/1.1
Accept: application/json
Authorization: Basic dGVzdHVzZXIwMTpuZXRjb29s
Host: localhost
Connection: keep-alive
Example system information GET response

GET /objectserver/sysinfo/compile HTTP/1.1
Accept: application/json
Authorization: Basic dGVzdHVzZXIwMTpuZXRjb29s
Host: localhost
Connection: keep-alive

HTTP/1.1 200 OK
Cache-Control: no-cache
Server: libnhttpd
Date: Wed Jul 4 15:38:53 2012
Connection: Keep-Alive
Content-Type: application/json;charset=UTF-8
Content-Length: 286
{
  "compile": {
    "full_details": "Tuesday June 26 17:12:01 BST 2012 on hurccsol.hursley.ibm.com (SunOS 5.9 Generic_118558-30)",
    "date": "Tuesday June 26 17:12:02 BST 2012",
    "machine": "hurccsol.hursley.ibm.com",
    "system": "SunOS 5.9 sparc",
    "build_version": "750.CAPPL.01"
  },
  "rest": {
    "version": "v1.0",
    "major": 1,
    "minor": 0
  },
  "oslc": {
    "version": "v1.0",
    "major": 1,
    "minor": 0
  }
}

Example system information element GET request

GET /objectserver/sysinfo/compile HTTP/1.1
Accept: application/json
Authorization: Basic dGVzdHVzZXIwMTpuZXRjb29s
Host: localhost
Connection: keep-alive

Example system information element GET response

HTTP/1.1 200 OK
Cache-Control: no-cache
Server: libnhttpd
Date: Wed Jul 4 15:38:53 2012
Connection: Keep-Alive
Content-Type: application/json;charset=UTF-8
Content-Length: 412
{
  "compile": {
    "full_details": "Tuesday June 26 17:12:01 BST 2012 on hurccsol.hursley.ibm.com (SunOS 5.9 Generic_118558-30)",
    "date": "Tuesday June 26 17:12:02 BST 2012",
    "machine": "hurccsol.hursley.ibm.com",
    "system": "SunOS 5.9 sparc",
    "build_version": "750.CAPPL.01"
  }
}
This example shows a $OMNIHOME/etc/libnhttpd/json configuration file, which is edited to define MIME type settings and HTTP headers in HTTP responses that are returned by the HTTP interface and OSLC interface. To enable MIME type settings and HTTP headers, enable the NHttpd.ConfigFile property.

The sections that enable MIME type settings and HTTP headers are as follows:

**httpResponse**
Defines the HTTP headers that are in the HTTP responses that are returned by the HTTP interface and OSLC interface. It has the following subsections:

**corsHeaders**
Overrides Cross-Origin Resource Sharing (CORS) HTTP headers. By default, the default headers are overridden to indicate that the Location HTTP header are be allowed and exposed. This setting is required for HTTP 201 Create response messages.

**httpHeaders**
For user-defined HTTP headers. These headers are added to all HTTP responses. Use this section to add static values for clients. A sample header is provided in the example.

**mimeTypes**
This section assigns a file extension, for example .html, to a MIME type. When file-serving is enabled, these definitions are used to determine the MIME type for the file. They also set the Content-Type HTTP header so that browsers can handle the file correctly. The $OMNIHOME/etc/libnhttpd/json file has a default set of MIME type settings that you can add to.

**Example**

```json
{
   "_comment": "This file provides additional configuration data to the embedded HTTP socket library (libnhttpd).",
   "httpResponse": {
      "_comment": "This section defines a set of user defined static elements that should be returned in an HTTP response, such as HTTP headers.",
      "corsHeaders": [
         {
            "name": "Access-Control-Allow-Headers",
            "value": "Location"
         },
         {
            "name": "Access-Control-Expose-Headers",
            "value": "Location"
         }
      ],
      "httpHeaders": [
      ]
   },
   "mimeTypes": {
      "_comment": "This section maps MIME types to file extensions. It is used by libnhttpd to determine the MIME type for a file that is to be served from its file serving URI.",
      "application/json": [
         "json"
      ],
   }
}
```
"application/rdf+xml" : [
  "rdf"
],
"application/xslt+xml" : [
  "xsl", "xslt"
],
"image/jpeg" : [
  "jpg", "jpeg"
],
"image/gif" : [
  "gif"
],
"image/png" : [
  "png"
],
"text/css" : [
  "css"
],
"text/javascript" : [
  "js"
],
"text/HTML" : [
  "htm", "html"
],
"text/plain" : [
  "txt", "log"
],
"text/xml" : [
  "xml"
]
}

Related reference:

"ObjectServer properties that control the HTTP interface and OSLC interface" on page 2

ObjectServer properties that control the HTTP and OSLC interfaces.
Appendix C. List of abbreviations

The API documentation for the ObjectServer HTTP interface and the ObjectServer OSLC interface use the following abbreviations and terms.

**HTTP**  Hyper Text Transfer Protocol. HTTP version 1.1 is defined in RFC2616. Unless otherwise noted, the term HTTP is used in this document to mean both HTTP and HTTPS.

**HTTPS**  Hyper Text Transfer Protocol Secure, as defined in RFC2818.

**JazzSM**  Jazz for Service Management, which is available from https://www.ibm.com/developerworks/mydeveloperworks/groups/service/html/communityview?communityUuid=69ec672c-dd6b-443d-add8-bb9a9a490eba.

**JSON**  JavaScript Object Notation, as defined in ECMA-262.

**MIME**  Multipurpose Internet Mail Extensions. MIME media types are defined in IANA MIME Media Types.

**OSLC**  Open Services for Lifecycle Collaboration, as defined at http://open-services.net.

**REST**  Representational State Transfer, as originally and informally described in Architectural Styles and the Design of Network-based Software Architectures.

**URI**  Uniform Resource Identifier, as defined in RFC3986.

**XML**  eXtensible Markup Language, as defined by W3C.
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:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, 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 wish 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
958/NH04
IBM Centre, St Leonards
601 Pacific Hwy
St Leonards, NSW, 2069
Australia

IBM Corporation
896471/H128B
76 Upper Ground
London SE1 9PZ
United Kingdom

IBM Corporation
JBF1/SOM1
294 Route 100
Somers, NY, 10589-0100
United States of America

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 document 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.

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.

 Portions of this product include software developed by Daniel Veillard.
• libxml2-2.7.8
The libxml2-2.7.8 software is distributed according to the following license agreement:
All Rights Reserved. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:
The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.
The SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE DANIEL VEILLARD BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
Except as contained in this notice, the name of Daniel Veillard shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization from him.

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

Trademarks

AIX, IBM, the IBM logo, ibm.com®, Informix, Netcool, System z, Tivoli®, and Tivoli Enterprise Console® are trademarks or registered 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.