Foglight® 6.0.0

Data Model Guide

their respective owners.

Legend

■   **WARNING: A WARNING icon indicates a potential for property damage, personal injury, or death.**

❗   **CAUTION: A CAUTION icon indicates potential damage to hardware or loss of data if instructions are not followed.**

ℹ   **IMPORTANT NOTE**, **NOTE**, **TIP**, **MOBILE**, or **VIDEO:** An information icon indicates supporting information.

Contents

The Foglight data model

This topic provides an introduction to models and discusses the Foglight data model.

- What are models?

- Who needs to know about models?

- Where can I see the models?

- How are models organized?

- What are domains?

- What is a data source?

- What are "KnowledgeItems"?

What are models?

In general, models are abstractions that capture the essence of the objects they are supposed to represent. A good model looks and behaves like the real thing, at least in certain ways. If a model were perfect in every respect, it would be indistinguishable from the real thing. Thus, we could pose questions, submit these in some way to the model, and obtain the same, or almost the same, results as we would by doing those things to the real object. If the object under consideration undergoes a change, the model would have to change accordingly in order to faithfully represent that object.

The data model used in the Management Server is constructed to do just that. The data sent to the Management Server changes with time, not only because the measurements on properties change, but because the objects themselves may come and go. So, a data model for use with the Management Server must be designed to accommodate the creation of objects, by placing them in a well-designed model hierarchy. Objects have relationships among themselves, and a good model accounts for those relationships.

To the Management Server, models are collections of related data objects. The totality of data objects in existence at any one time is referred to as the "data model".

- Objects are created by transforming the raw data collected by agents (collection models) or when services are created, deleted, or modified (service models).

- Objects have properties, such as lists and metrics (time series values). Properties may be simple values, but often they are other objects. Being objects, they can have properties that are objects, and these objects may have the starting object as a property. Thus, the relationships form a graph, not a tree.

  > **i** | **NOTE:** Data objects having other data objects as properties can be a source of confusion when you attempt to drill down into an object's properties using a data browser. It is possible to encounter a loop of related properties that the data viewer (a tree of nodes) unwinds into a seemingly endless chain of repeating properties. When using the data browser, it is seldom productive to go more than five levels deep. After that, it is likely that you are in an unwound loop.

- Data models can be organized into sub-modules, for instance:

  - HostModel represents a collection of agents on a host.

  - Windows_System represents a Windows® System agent.

  - Physical_Disk represents a disk.

- Raw data can be modeled in different ways.

- You can examine the data model's static skeleton (the defined data types and their inter-relationships) in the Schema Browser.

- You can examine the dynamic data objects in the data browser (**Dashboards > Configuration > Data > Management Server > All Data**).

> **i** | **IMPORTANT:** Attempting to diagram the data model using these views can entail considerable work. Models should be regarded as being internal to the Management Server.

Who needs to know about models?

Knowledge of the data model is beneficial if you are performing one of the tasks listed in the following table.

Table 1. Who needs to know about models?

Extending the model	You might want to add an extra metric to the system, extend an existing collection, or monitor a new and currently unsupported application.
Using the query language to build a new UI query	You need to understand the data model to construct a path to the data objects of interest. If you want to write a query to return all the instances of an object of a certain type about which the Management Server currently knows, you can locate them in the data model. Some objects are deeply nested; construct the most efficient path to them.
Building your own dashboards	If you intend to design and build your own dashboards, you need to know what the existing data model can supply. Because the Management Server is an open-ended data modeler, it is possible with sufficient effort to build a new data collection framework from the ground up. In brief, the process of building a new data collection framework is as follows: • If the data is not already available from existing agents, create an agent to collect the data and install it on target systems. One way is to create a formatted script that writes to `STDOUT`, upload the script, build a script agent, and then deploy and activate the agent. Consult Quest's Professional Services organization. **NOTE:** You can use any executable on the client system that writes to `STDOUT`. All you need is a script to launch the application. • Configure the transformation layer that creates typed objects to hold the data and define the interrelationships between data types. Tools: XML and Groovy. Design and build the dashboards to present the data in the Management Server using WCF. Tool: the component editors in Definitions. Resources: the *Web Component Tutorial* and the *Web Component Reference* pages.

Modeling process overview

In this guide, modeling refers to the process of creating new in-memory models. This process can include the creation of:

- A static framework of types and their relationships

- New types when they become needed

- Transformations to create the types as a result of data arrival or model changes

- Instances of those types

- Dashboards and monitoring policies to work on those types

The modeling process is illustrated in the following diagram.

Figure 1. Modelling process
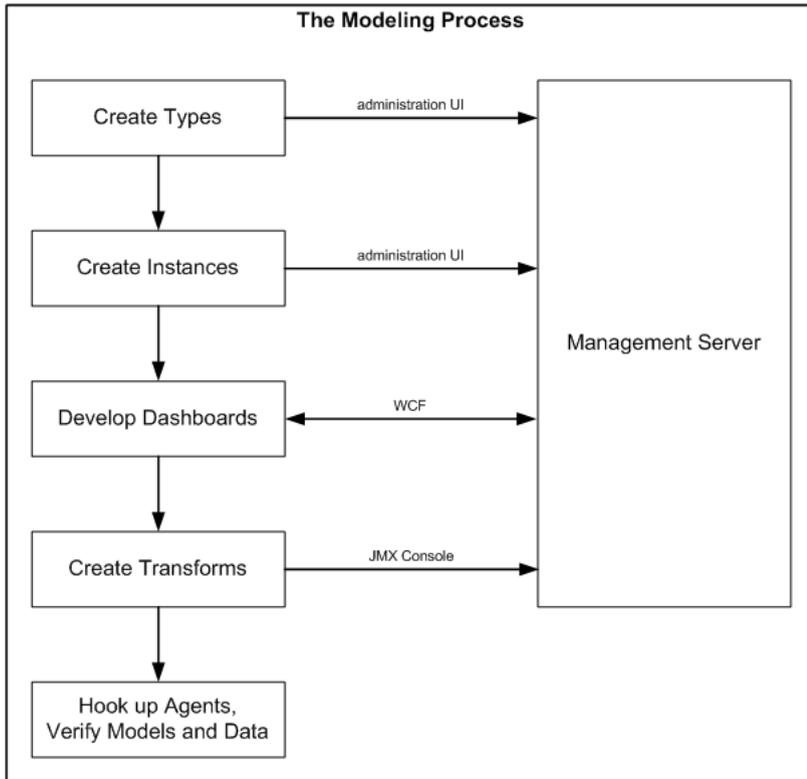
You develop models to organize and convey the relationships that exist among the pieces of monitoring data, so that they can be presented visually in dashboards.

The first step in the modeling process is to define the static types that are used to represent the data. These form the core data types of the data model. Once these types are defined, it is possible to represent them in a schema, and these types can be seen in the Schema Browser dashboard.

You define types by creating a topology type XML file. You can deploy the XML file to the Management Server using the administration UI. The Management Server handles versioning of the types.

The second step in the modeling process is to define instances of those types. Currently, you can only create instances of types using Groovy scripts.

The third step in the modeling process is to create dashboards using WCF. You can perform dashboard creation with just the types deployed, as long as the type definitions include metric definitions. However, it is difficult to test the dashboards without instances.

Once you have built dashboards and they are operating on test data, it is time to start using agents and enable transformations (fourth step in the modeling process). The transformations translate collected data into the topology object instances specified in the second step.

Ideally, the final-form agent is feature complete and collecting data when you reach this step.

Where can I see the models?

The following sections present additional model-related information.

- Topology type definitions in XML

- Schema browser
- Data dashboard

Topology type definitions in XML

Most core model type definitions are available as XML files in the Management Server's *config* directory.

- The *topology-types.xml* file contains the core types.

- The *host-topology-types.xml* file contains the host model topology types.

- The *forge-topology-types.xml* file contains types for the server's self-monitoring.

- The *foglight4-topologytypes.xml* file contains types related to the conversion from Foglight 4 to Foglight 5, and related to the Foglight4Model.

Schema browser

Use the Schema Browser dashboard to view information about the available data types, their relationships in the data model, properties, and object instances. This dashboard can help you to better understand the data model structure and learn about existing object dependencies.

For more information about the Schema Browser, see the *Foglight Dashboard Support Guide*.

> **i** | **NOTE:** The Schema Browser does not show how models are organized inside the Management Server. Use the Data Browser to see that organization.

Data dashboard

The Data dashboard shows the models' organization, with related sub-models grouped together. For example, all Host instances are available in a *Hosts* model at the top level; all Foglight 4 converted agents are available in a *Foglight4Model*; and all services are available together in the *Services* model.

The Data dashboard:

- Shows the raw underlying set of objects.

- Is organized around a root, like a file system.

- Indicates the path to objects so that you can use the paths to define WCF queries.

- Shows objects and views for objects of that type.

- Includes a property viewer for looking at the details.

- Contains data browser views for F4 tables.

Use the Data dashboard to see models as they are organized in the Management Server.

To access the Data dashboard:

- In the navigation panel, under Dashboards, navigate to **Configuration > Data**.

    The Data dashboard appears.

The Data dashboard consists of a model/object selector on the left and a view on the right. The view on the right is based on the type of object you select. There are default viewers for looking at objects (Property Viewer), lists (List Viewer), and metrics (Metric Viewer). If a user creates a view that takes a particular type as input (for example, Host), then that view is available for use in the Data browser when that type is chosen. The most useful view for inspecting an object is the Property Viewer, which shows the properties and values for the currently selected object. The Property Viewer shows the raw object details (property values and metrics).

How are models organized?

In the Data dashboard, the top-level nodes are based on root queries. The entries that appear in the Data tree are instances of the types defined in the data model. Thus, not all defined types appear, but only those types for which Agents have created at least one instance.

These queries are defined in modules and they use the module name as the node name. There are several nodes that are always present because they are root queries defined in the Management Server. Other nodes may be present. They originate from root queries defined in cartridges that have been installed on the Management Server.

Some examples are:

- Alarms — *Current Alarms* and *Outstanding Alarms*

- Hosts — *All Hosts*

- Management Server — *All Agents*, *All Data*, *Schema*, and *Servers*

- Services — *All Model Roots*, *All Object Groups*, *All Service Categories*, *All Services*, and *All System Services*

- Other nodes, depending on the cartridges that have been installed

For example, the common Host model is a topology model defined in the Management Server to describe the host objects being monitored. The use of a common model allows the Management Server to provide out-of-the-box configuration for visualizing hosts that could be monitored by any type of agent.

For more information, see:

- Where can I see the data model hierarchy?

- What internal models are created?

- What other models are created?

- What is in "Model Roots"?

- How are Foglight 4 agents represented?

- Why might I need to navigate models?

- How do I create a model?

- How do I delete a model?

Where can I see the data model hierarchy?

All data models have a root. In the Management Server's user interface, *Dashboards > Configuration > Data > Management Server > All Data* is considered the root ("/") in the dashboard framework. The *All Data* node contains references to the ModelRoot topology instances. A data object shows up under *Management Server > All Data* if it is defined in *topology-types.xml* file.

While it is possible for a cartridge to add a root property by explicitly defining the property in the cartridge's *topology-types.xml* file, changing types in this way is not encouraged. It is better for the cartridge to provide a new model root, so that a property is added automatically.

The cartridge can do this by defining a topology type in its schema that extends the core `"ModelRoot"` type. For example:

```
<type name="MyModelRoot" extends="ModelRoot">
```

When the new root type is created, the server automatically adds a corresponding property to the `Root` type that returns instances of that model root. The cartridge then needs to create a single instance of that type and reference the top-level domain objects. The topology types used for top-level domain objects should be annotated with the core `"DomainRoot"` annotation, so that they can be configured with the admin grouping functionality. For more information about domains, see the What are domains? topic.

For example:

```
<type name="MyModelRoot" extends="ModelRoot">
  <property name="topLevelObjects" type="MyTopLevelObject" is-many="true"/>
</type>
<type name="MyTopLevelObject" extends="TopologyObject">
  <annotation name="DomainRoot"/>
</type>
```

What internal models are created?

The Management Server builds internal models for its own activities.

Table 2. Management Server internal models

Model Name	Description
Alarms	Contains all alarms.
All Type Instances	Contains a hierarchy of types with instances; similar to schema browser.
FSMServers, FSMDatabases	Contains the internal Management Server and database models for self-monitoring.
Model Summary	Collects metrics on all models in Model Roots.

What other models are created?

Table 3. Other models created

Model Name	Description
AgentMap	Contains all agents.
FSMServiceRoot	Contains the service model.
ServiceLevelPolicies	Contains all service level policy objects.

What is in "Model Roots"?

The `ModelRoots` property is essentially a collection of all the `ModelRoot` instances on the system. This is a subset of the other root properties, because some model roots are not instantiated if the cartridge is not being actively used. The `ModelRoots` and `ModelInstances` properties should not be used (especially when developing dashboards) because it is more efficient to use the root property that was added for specific `ModelRoot` types.

Cartridge developers can add their own element to `ModelRoots` by extending a special class called `CollectionModelRoot`.

How are Foglight 4 agents represented?

Foglight 4 agents are represented as follows:

- Foglight 4 organizes data as tables inside agents running on hosts in an IPMap.

- Foglight 5 creates objects that mimic this: F4Host, F4Agent, and F4Table.

- Tooling converts the F4 DCMs into these classes.

- Data shows up under All Collections > IPMap.

- One object instance per sample, for example, one object for `C:`, another for `D:`.

Why might I need to navigate models?

You need to understand the model when building a query to return data. Alternatively, you might need to construct a path to a particular property of an object. The object lives in data model space; typically, there is more than one pathway to access it. You will want to choose the most efficient of these.

How do I create a model?

The simplest way to create a model is as follows:

1 Diagram the model and analyze it for flaws.

2 Create the *cartridgename-topology-types.xml* file for all the data types and their relationships in the model.

> **i** | **NOTE:** You can name the XML file as best fit for your model. The type definitions are merged into the server, but the XML file is not retained on the server, so there is no possibility of a name clash with XML files existing on the server. Once the cartridge development gets to the stage where there are several artifacts (for example, monitoring policy files, WCF modules, etc.) the *cartridgename-topology-types.xml* file is added to a cartridge along with the other components.

3 Populate the model using a Groovy script.

How do I delete a model?

- The Data Management dashboard allows objects to be deleted.

- Only items from *ModelRoots* can be deleted.

- If data is being collected, delete is not destructive, and

  - The object is recreated

  - Metrics are reconnected

  - Agent instances are more difficult to recreate. Avoid deleting them.

What are domains?

A domain is a specific technology or part of your environment that you are interested in monitoring and for which data is collected and model instances are built.

The Domains dashboard presents an overview of all domains for which you can collect data. It summarizes the state of all monitored domains and allows you to drill down on a specific domain and investigate problems related to it. For more information about monitoring domains, see the *Foglight User Guide*.

Example: the Host Model structure

Besides its central importance, the Host Model serves as the prime example for understanding parts of the overall data model.

Host Model structure

The following diagram illustrates the topology types that are used in the construction of the host model and the arrangements of those types.

Figure 2. Host Model diagram
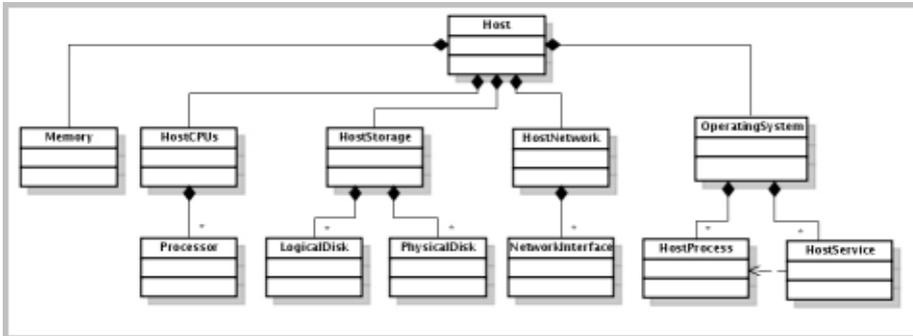
The `Host` type is the root of the model. Instances of this type are identified by the `name` property which is typically set with the fully qualified domain name of the host.

The Host object contains an aggregate node to group related components. There is a single aggregate node to summarize the state and performance of the host's memory, processor, storage and network components. The aggregate nodes may then contain other objects to provide a breakdown of the aggregate information.

Agents running on a host are referenced from the corresponding Host object, but do not form part of the host model and do not contribute to the state of the host model itself. That is, there may exist a host that is running an agent that encounters a fatal error. In this case, the Host object in the topology remains in the normal state and references the Agent object that encountered the error. The Host objects only change state because of host-specific rules (for example, CPU utilization).

Host Model creation

The common host model can be created from many different sources. One common source is the operating system cartridge, which contains canonical data transformations (CDTs) to create the host model from the data submitted by their agents. This information is transformed via CDTs to produce the physical host model objects and to populate the metrics. Script agents must use CDTs to convert their data into a host model. Another common source is the infrastructure cartridge, which uses the agent manager API to construct the host model directly in Java® code, without requiring a CDT. The host model is also created for virtual machines by the VMware® and Hyper-V® cartridges.

Extending the Host Model

Agents can capture host-related data that cannot be stored on the types defined in the common host model. The host model types **must not** be sub-typed to add support for this data because the types must be known in order for multiple cartridges to share the model. The host model may be extended through composition by adding instances of cartridge-specific types to the detail collection that is exposed on all host model objects.

Common Host Model types

The host model defines the following topology types: Host, Memory, HostCPUs, Processor, HostStorage, PhysicalDisk, LogicalDisk, HostNetwork, NetworkInterface, OperatingSystem, HostProcess, and HostService.

- There is a single `Memory` object attached to a host that is identified by that `Host` instance. The `name` property of a `Memory` object is set with the constant string "Memory".

- There is a single `HostCPUs` instance attached to a Host that provides host level summary metrics for the processors on a host. The `HostCPUs` instance is identified by the reference to the associated `Host` object and has its name set with the constant string "CPUs".

    - The `Processor` type is used to represent a logical CPU that is available on the Host.

- There is a single `HostStorage` instance attached to a host that provides host level summary metrics for the logical and physical disks on the host. The `HostStorage` instance is identified by the reference to the associated `Host` object and has its name set with the constant string "Storage".

    Some metrics are produced by derived metrics. The disk metrics are calculated based on the values provided for the associated `PhysicalDisk` objects. The space metrics are calculated based on the values provided for the `LogicalDisk` objects.

    - The `PhysicalDisk` represents a disk that is installed in the machine or configured for a virtual machine.

    - The `LogicalDisk` type is used to represent a Windows® partition or Unix® filesystem. These types have the same set of observations.

- There is a single `HostNetwork` instance attached to a host that provides host level summary metrics for the network interfaces on the host. The `HostNetwork` instance is identified by the reference to the associated Host object and has its name set with the constant string "Network".

    The observations on the `HostNetwork` type are all produced by derived metrics. These derived metrics calculate the value of a metric from the associated `NetworkInterface` objects.

    - The `NetworkInterface` type is used to represent a network interface that is installed in the machine or configured for a virtual machine.

- There is a single instance of the `OperatingSystem` type attached to a host. The instance is identified by the reference to that Host, and has the following child object types that provide additional details about the operating system (OS):

    - The `HostProcess` type captures aggregate metrics for all processes of a given type on the host. The `instances` complex observation captures the detailed per-process statistics, but that observation may not be produced at all times.

    - The `HostService` type captures information on the services configured to run on a Host and their state.

What is a data source?

A data source is a container for a data model. For instance, the Monitoring data source is for modeling monitored objects. It contains the data model. The Dashboards Meta Data source models the Web Component Framework artifacts. Thus, the various data sources model different top-level data collections. Domains such as WebLogic, which are closely bound to the monitoring model, are incorporated in the Monitoring data source.

If you are building dashboards for monitoring purposes, use the Monitoring data source.

The Data Sources dashboard is where you choose a data source. The default (and only current) option is the *foglight-5* data source. Click a data source to display its ID, name, and topology and UI query service names.

To display the Data Sources dashboard:

- From the navigation panel, under **Dashboards**, click **Configuration** > **Data Sources.**

The Data Sources dashboard appears.

Figure 3. Data Sources dashboard
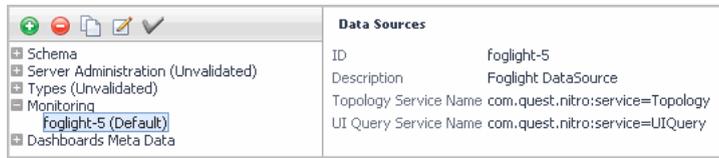
Use the icons above the data sources to add ⊕ , delete ⊖ , copy ▢ , and edit ✎ them, and set a default ✓ data source.

> ⚠ **CAUTION: If you click the Delete icon, the data source is removed without a confirmation dialog.**

What are "KnowledgeItems"?

`KnowledgeItem` is a concept that allows users to attribute business-relevant information to their monitored infrastructure. This information may include (but is not limited to) cost-center attributes, business unit associations, or configuration and application management metadata.

The Management Server pre-defines an abstract type `KnowledgeItem` that contains basic information:

```
<type name='KnowledgeItem' extends='TopologyObject'>
  <annotation name='Abstract'/>
  <property name='asset' type='TopologyObject' is-identity='true'/>
  <property name='name' type='String'/>
  <property name='description' type='String'/>
</type>
```

Concrete instances of sub-types can be used to extend these knowledge items for a specific purpose (for example, business unit information), and are linked to the corresponding monitored topology information via the reference `'asset'`.

The Management Server will offer management of these knowledge items in the UI in a later release. At the moment, users should extend the `KnowledgeItem` type as desired, and create business-related knowledge content and references in scripts.

The following is an example of a simple business-unit related knowledge item:

```
<types>
  <type name='AcmeBUKnowledgeItem' extends='KnowledgeItem'>
    <property name='lineOfBusiness' type='String'/>
    <property name='businessUnit' type='String'/>
  </type>
</types>
```

To add real value to this basic knowledge item, extend this type, by renaming and customizing it appropriately — see *<foglight_home>/extension/knowledge-items/acme-types.xml*

The types-file can be imported into the model by copy and paste, or as a file upload in the **Administration > Data > Add Topology Type** dashboard.

After users define various KnowledgeItems, other knowledge items instances can be created programmatically. For example:

```
// run in script editor, assuming a target scope (the asset) is set
if (scope==null)
  return "No scope/asset to attach KI to"
ts = server.TopologyService;
t = ts.getType("AcmeBUKnowledgeItem");
ki = ts.getObjectShell(t);
```

```
ki.set("asset", scope);
ki.set("name", "R&D Business Unit Asset #1");
ki.set("description", "Host hosting business unit R&D");
ki.set("lineOfBusiness", "Research");
ki.set("businessUnit", "R&D");
ts.mergeData([ki]);
```

The Management Server shows the `KnowledgeItems` in the model under **Services > All Model Roots > Knowledge Items** and **Root of Monitoring > KnowledgeItemModel >knowledgeItems**. To see the data model tree, in the navigation panel, under **Dashboards**, click **Configuration > Data**.

The server comes with an example that shows how to surface the knowledge items in the UI. The example dashboard depends on a Acme example KnowledgeItem type (see *<foglight_home>/extension/knowledge-items/acme-types.xml*) that first needs to be imported through the Add Topology Type dashboard. The example dashboards can then be imported by using the following command:

```
fglcmd -cmd util:uiimport -f acme-dashboards.zip
```

When accessing the Service Operations Console (in the navigation panel, under **Dashboards**, click **Services > Service Operation Console**), all `Acme` knowledge items for elements inside the selected service are then shown in an extra tab, named **ACME BU Knowledge Items**.

Data modeling tutorials

This chapter provides a tutorial that walks you through the process of adding new functionality to Foglight, by building new models and without changing the Foglight core code.

The technique for extending Foglight is shown in the following illustration.

Figure 1. Extending the Foglight model

The process consists of the following steps:

1 Define your new types.

2 Create your types through data — Groovy functions or agents.

3 Create dashboards based on your type instances.

4 Re-iterate (repeat Step 1 through Step 3) until you get the result you want.

> **NOTE:** If you make an error in your type definitions, you do not have to throw away your server. You can change your definition on the fly, and Foglight adapts. There are also functions available to remove the data. This means that it is possible to iterate over a model definition and work towards the result you want. You do not have to sit down with a Unified Modeling Language (UML) tool for a day or two.

Modeling example: The org chart

Foglight is not just a monitor, it is an object-oriented application server. Using Foglight's modeling capabilities, you can extend Foglight to model data from pretty much any domain.

This example walks you through an exercise that extends Foglight to represent a company's Org structure. We create the model, load it into the server, populate it using a Groovy script, and create some dashboards to visualize the result:

- Defining the org structure types
- Populating the model using a script
- Connecting "Employees" to Host objects
- Visualizing the data

A company's org chart shows all the different departments in the company, along with their relationships.

Figure 2. The company org chart
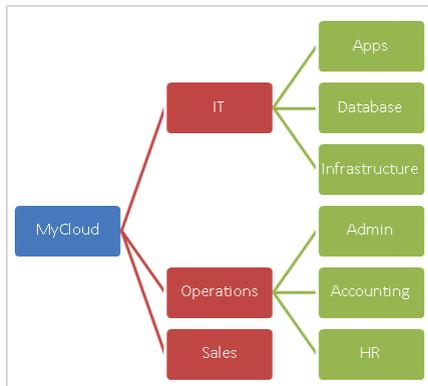
To make this exercise interesting, we are going to model the org chart and the employee reporting hierarchy. Each organization has a leader, and that leader has direct reports. Then, we are going to connect the IT department to the actual IT infrastructure. The result is a meaningful org chart that shows the state of the IT and the root cause of potential outages from an employee perspective.

Figure 3. The company org chart structure
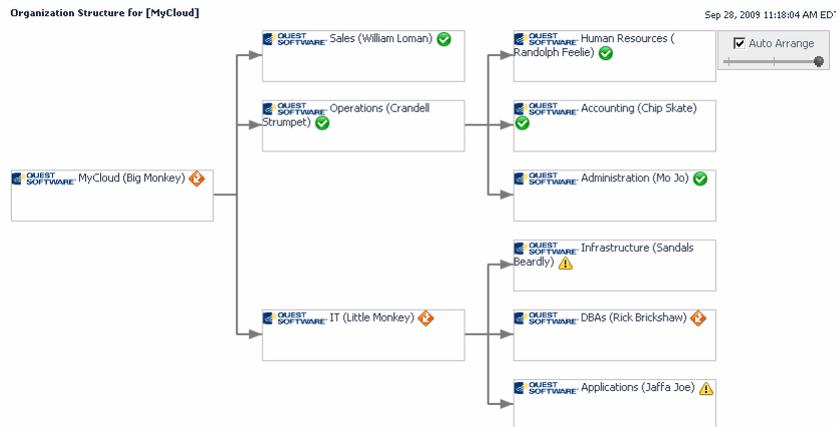
Defining the org structure types

To demonstrate the power of models, we are going to create a company org structure inside Foglight.

The org structure consists of organizations that contain other organizations. The organizations are lead by managers. The managers have direct reports.

We will create a model that represents these relationships, populate the model, then visualize it using dashboards. Then, we will create a relationship between this Org Structure model and Host objects, to show how models can be connected.

Org structure: Type hierarchy and relationship

There are three key types in our Org Structure example: Organization, Manager, and Employee. It seems natural that a Manager should be a specialization of an Employee. It also seems natural that there is a relationship between the organizations, and also between the employees.

As always with anything object-oriented, there are multiple ways to solve the problem of relationships between objects. Here are the key relationships we want to maintain:

- We want to know what sub-organizations make up different organizations.

- We want to know who owns each organization.

- We want to know who works in each organization.

- We want to know who reports directly and indirectly to each manager.

- We want to be able to have managers who do not own an organization.

- We want the contact information for every employee.

To do that, we are going to create three objects: `Organization`, `Manager`, and `Employee`. `Manager` is a subclass of `Employee`.

> **i** | **NOTE:** You could also create an extra object, Department, as a subtype for Organization. But for this
> example, we will keep things simple and create just the previously mentioned objects.

All the relationships we need are going to be modeled using three collections:

- Each `Organization` has a `Manager` defined by `Organization.orgLeader`.

- Each `Organization` has a list of child `Organizations` defined by `Organization.childOrgs`.

- Each `Manager` has a list of direct reports (`Employee` objects) defined by `Manager.directReports`.

Before we get into the type definitions for each of our three types, we need to figure out what defines the identity of each type. In Foglight, identity is critical, because it determines when a new object is created.

In a real-world case, the identity of each employee is defined by an employee ID. For this example, we are going to define the identity of an employee based on his or her first and last name. That feels like a unique value for an individual, at least in a small organization. For `Organization`, we are going to define the identity based on the name of the organization. The reasons behind choosing this option are explained in the following scenarios:

- If the identity of an organization included the name of the manager of the organization, what would happen if we changed the organization's leader? We would have a new organization, and that is not right. The organization should remain unique, even if the manager changes.

- If the identity of an organization included the child organizations, then we would not be able to create departments without changing the parent organization into a new instance.

- If the identity of a manager included the organization that he or she manages, then we would not be able to handle promotions cleanly.

Therefore, we are going to use the following identity information for our types (as illustrated in the following diagram):

- `Employee`: identity fields `lastName` and `firstName`.

- `Manager`: same identity as Employee, has a `directReports` collection of Employees.

- `Organization`: identity field `orgName`. Has an `orgLeader` link to the organization's manager, and has a `childOrgs` list of departments that make up the organization.

Figure 4. Organization structure
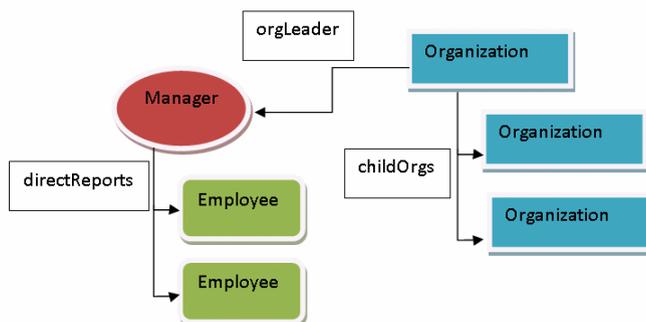
Now we can finally get started on creating some types inside Foglight. After designing the object/type hierarchy, getting the types into Foglight is relatively easy.

Foglight model definition for org chart

Rather than examining the syntax for topology type definitions, we are going to study real examples.

In Foglight types, a type can extend another type. Most types extend `TopologyObject` in order to inherit alarm propagation and roll-up features. Each type contains a set of properties. A property can be:

- A metric (that is, time-series data like "CPU Utilization").

- A Java® type, like "String" or "Integer".

- Another object.

- A collection of other objects.

The "Employee" type

We can get started by looking at the simplified definition for the `Employee` type. It contains two properties, `lastName` and `firstName`:

```
<type name='Employee' extends='TopologyObject'>
  <property name='lastName' type='String' is-identity='true'/>
  <property name='firstName' type='String' is-identity='true'/>
</type>
```

To load this type definition:

1 On the navigation panel, under Dashboards, click **Administration > Data > Add Topology Type**.

The **Add Topology Types** dashboard appears.

2 In the Import From Text box, keep the pre-populated text in the dialog, and insert your type definition between the `<types></types>` tags.

Figure 5. Add Topology Type dashboard
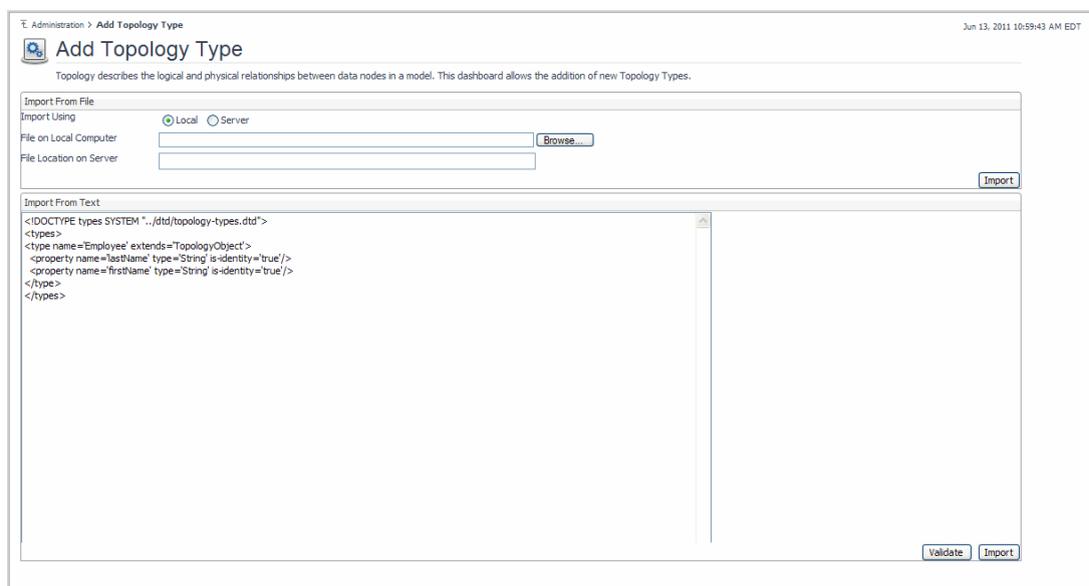
3 To validate the type definition, click **Validate**.

If the code passes the validation, the **Successful** dialog box appears, confirming the validation result.

4 Click **Close**.

5 To import the new type definition, click **Import**.

The **Successful** dialog box appears, confirming the result of importing the new type definition.

6 Click **Close**.

> **i** | **IMPORTANT:** You can repeat this procedure as many times as you want. As you modify your type, Foglight adapts. Later on you can add new fields, and they are added to the type definition. This is important to understand. It means you can iterate over a type in a Foglight server without worrying about backing out the older versions.

> **i** | **NOTE:** To review the definitions for all the important types, including TopologyObject and Host, see FGLHOME/config/topology-types.xml. To review the DTD definition for the topology type XML file format, see FGLHOME/dtd/topology-types.dtd.

The `Employee` type definition is now part of the Foglight topology types. We should highlight the following characteristics:

- `Employee` extends `TopologyObject`. This means that it inherits all the alarm roll-up capabilities that most Foglight objects use. We want this capability because we want to be able to show the state of the systems managed by an `Employee`.

- Both `lastName` and `firstName` are String properties.

- Both `lastName` and `firstName` are identity properties. This means that the combination of first and last name define a unique `Employee` object instance. You can have two `Employee` objects with the same last name, or the same first name. But the combination is considered unique.

Next, we are going to look at a slightly extended version of the `Employee` object:

```
<type name='Employee' extends='TopologyObject'>
  <property name='lastName' type='String' is-identity='true'/>
  <property name='firstName' type='String' is-identity='true'/>
  <property name='jobTitle' type='String'/>
  <property name='phoneNumber' type='String'/>
  <property name='emailAddress' type='String'/>
  <property name='organization' type='Organization'/>
  <property name='ownedItems' type='TopologyObject' is-many='true' is-
containment='true'/>
  <property name='interestingItems' type='TopologyObject' is-many='true' is-
containment='false'/>
</type>
```

This (partial) definition includes new properties, which hold basic contact information for each employee. We have added `jobTitle`, `phoneNumber`, and `emailAddress`. These properties are a little different than `lastName` and `firstName`. Specifically, they are not identity properties. What that means is:

- An `Employee` is not uniquely defined by his or her phone number, for example. A change to a phoneNumber property does not create an `Employee` instance.

- Foglight automatically tracks changes to non-identity properties as topology changes. This means Foglight notes the time of the change, and the old-new value pair. Therefore, we get the object change tracking for free (for example, we can find out when an employee's phone number changed).

The next interesting definitions are for `ownedItems` and `interestingItems`, which are used to model an employee's system responsibilities. If Bob in IT is responsible for three machines, we want to be able to represent that relationship. We have chosen to add two collections to represent this relationship:

- `ownedItems`: represents those items for which an employee is responsible. The state of these items impact the state of the employee. If Bob in IT has three machines, and one of them is in a `Fatal` state, then the `Employee` object representing Bob is in a `Fatal` state. This is made to happen by setting `is-containment` to `true`.

- **interestingItems**: represents things that an employee finds interesting, but for which they are not responsible. This allows us to maintain a relationship between an employee and other things that does not result in alarm state rolling up. This is made to happen by setting `is-containment` to `false`.

> **i** | **NOTE:** Both `ownedItems` and `interestingItems` are set to type TopologyObject. Remember that TopologyObject is the base type for most objects in Foglight. By using this as the type for a collection, we are allowing nearly any object to be added to the collection. In this case, it makes sense, as it allows for an Employee to include services, hosts, application servers, databases, and agents in their `ownedItems` list.

There is one last technique to show related to model definitions. The value of a property can be set at run-time by agent or script, but we can also supply code to create a property. As a simple example, consider the fact that we have stored `lastName` and `firstName` separately. It is likely that we want to put the two names together when we build dashboards. To do that, we could add a `fullName` property that has a copy of the data in `lastName` and `firstName`, but they might get out of sync. It makes much more sense for us to make `fullName` a derived property, and to provide a Groovy code to calculate its value:

```
<property name='fullName' type='String'>
  <annotation name='DerivationExprType' value='Script'/>
  <annotation name='DerivationExpression'>
  <value>
    StringBuilder fullName = new StringBuilder();
    fullName.append(scope.get("firstName")).append(" ");
    fullName.append(scope.get("lastName"));
    return fullName.toString();
  </value>
  </annotation>
</property>
```

The code is straight forward and should be familiar to everyone who knows either Groovy or Java®. We are building a string that puts the first and last names together, and returns that value. When we access `fullName`, this script is run.

> **i** | **NOTE:** We need to emphasize that the value of an object's Property can be produced by a Groovy expression. This is a powerful capability. For example, you can create a property that finds all SQL Server® database instances running on hosts that are also running WebLogic.

The "Manager" type

The `Manager` type extends `Employee`; its simplified definition contains one property, `directReports`.

```
<type name='Manager' extends='Employee'>
  <property name='directReports' type='Employee' is-many='true'/>
</type>
```

This adds the ability to have a list of direct reports. By setting the `is-many` attribute to `true`, we allow a `Manager` to have zero or more `Employees`.

The "Organization" type

The `Organization` type (simplified) definition contains several properties:

```
<type name='Organization' extends='TopologyObject'>
  <property name='orgName' type='String'/>
  <property name='orgLeader' type='Manager' is-containment='true'/>
  <property name='childOrgs' type='Organization' is-containment='true' is-
many='true'/>
</type>
```

An `Organization` inherits `TopologyObject`. Each `Organization` has an `orgName` that uniquely defines its identity. All `Organization` instances must have different values for `orgName`.

Each `Organization` has a relationship with a `Manager` through the `orgLeader` property. Each `Organization` can only have one manager. This might be a flaw in our model, but we will keep it as-is, for now. Also note that `is-`

`containment` is set to `true`. This means that an `Organization` inherits the state of its `Manager`, who in turn inherits the state of his or her `Employees`.

Each `Organization` also has a relationship with zero or more child `Organization` objects through the `childOrgs` property. This property has `is-many` attribute set to `true`, in order to allow an Organization to have zero or more child Organizations.

How models hang together

Objects can be created easily in Foglight. However, unless some thought is given to their structure, objects can be easily lost or become orphans.

To find orphan objects:

1   On the navigation panel, under Dashboards, click **Configuration > Data**.

   The Data Browser dashboard appears.

2   Expand the topology tree to show the **Management Server > All Data > All Type Instances > TopologyObject** collection.

   Figure 6. Expanded topology tree

   Underneath this collection you can find the list of all instances organized by type.

3   Open **TopologyObject > Sub Types**, then (for example) open **Organization** to see all `Organization` object instances.

But this is the hard way to find things. And if your objects are orphaned, you cannot write dashboards. What you want to do is to anchor all your new objects into a collection somewhere. To do that, you need a type definition as follows:

```
<type name='OrganizationModel' extends='ModelRoot'>
  <property name='organizations' type='Organization' is-many='true' is-
containment='true'/>
</type>
```

This creates a model that contains all `Organization` instances. As we create our objects (see Full definitions for the org structure types), we need to make sure that we add them to `OrganizationModel.organizations`. But how does this help? The key is the `ModelRoot`. This is a special marker class that tells Foglight to include this model in the list of `ModelRoots`. This means it displays as shown in the following illustration.

Figure 7. OrganizationModel property in the topology tree

As a result of including something in `ModelRoots`, you can now refer to your model in a query like this:

```
/OrganizationModel/organizations
```

Full definitions for the org structure types

This section presents the full type definitions for the new topology types discussed in the previous sections.

```
<!DOCTYPE types SYSTEM "../dtd/topology-types.dtd">
<types>

<type name='Employee' extends='TopologyObject'>
  <property name='name' type='String' is-many='false' is-containment='false'>
    <annotation name='DerivationExprType' value='Script'/>
    <annotation name='DerivationExpression'>
    <value>
      StringBuilder name = new StringBuilder();
      name.append(scope.get("fullName")).append(" (");
      name.append(scope.get("jobTitle")).append(" )");
      return name.toString();
    </value>
    </annotation>
  </property>
  <property name='lastName' type='String' is-identity='true'/>
  <property name='firstName' type='String' is-identity='true'/>
  <property name='fullName' type='String'>
    <annotation name='DerivationExprType' value='Script'/>
    <annotation name='DerivationExpression'>
    <value>
      StringBuilder fullName = new StringBuilder();
      fullName.append(scope.get("firstName")).append(" ");
      fullName.append(scope.get("lastName"));
      return fullName.toString();
    </value>
    </annotation>
  </property>
  <property name='jobTitle' type='String'/>
  <property name='phoneNumber' type='String'/>
  <property name='emailAddress' type='String'/>
  <property name='organization' type='Organization'/>
```

```xml
    <property name='ownedItems' type='TopologyObject' is-many='true' is-
containment='true'/>
    <property name='interestingItems' type='TopologyObject' is-many='true' is-
containment='false'/>
</type>

<type name='Manager' extends='Employee'>
  <property name='directReports' type='Employee' is-many='true'/>
  <property name='name' type='String' is-many='false' is-containment='false'>
    <annotation name='DerivationExprType' value='Script'/>
    <annotation name='DerivationExpression'>
    <value>
      StringBuilder name = new StringBuilder();
      name.append(scope.get("fullName")).append(": ");
      name.append(scope.get("jobTitle")).append(" of ");
      name.append(scope.get("organization").get("orgName"));
      return name.toString();
    </value>
    </annotation>
  </property>
</type>

<type name='Organization' extends='TopologyObject'>
  <property name='name' type='String' is-many='false' is-containment='false'>
    <annotation name='DerivationExprType' value='Script'/>
    <annotation name='DerivationExpression'>
    <value>
      StringBuilder name = new StringBuilder();
      name.append(scope.get("orgName")).append(": ");
      name.append(scope.get("orgLeader").get("fullName")).append(" ");
      return name.toString();
    </value>
    </annotation>
  </property>
  <property name='orgName' type='String' is-identity='true'/>
  <property name='orgLeader' type='Manager' is-containment='true'/>
  <property name='childOrgs' type='Organization' is-containment='true' is-
many='true'/>
</type>

<type name='OrganizationModel' extends='ModelRoot'>
  <property name='organizations' type='Organization' is-many='true' is-
containment='true'/>
</type>

</types>
```

Populating the model using a script

So far we have built some sophisticated types. To create instances, we have the following options:

- Create an agent.

- Create a script to load some data.

For this exercise, we will choose the second option (which allows us to get started much faster), and run the createOrg Groovy script in the script editor. This script populates our Org Chart model with interesting data.

The most important part of that script is the following function, which allows you to create an object and set properties on the object:

```
def createOrUpdateObject(typeName, propertyValues) {
  topSvc = server.TopologyService;

  def type = topSvc.getType(typeName);
  objShell = topSvc.getObjectShell(type);
  propertyValues.each ({propertyName, propertyValue ->
    def prop = type.getProperty(propertyName);
    if (!prop.isMany()) {
      objShell.set(prop, propertyValue);
    }
    else if (propertyValue instanceof Collection) {
      objShell.getList(prop).addAll(propertyValue);
    }
    else {
      objShell.getList(prop).add(propertyValue);
    }
  });
  return topSvc.mergeData(objShell);
}
```

This function uses the topology service to create or update an object instance. To do that, it requires a `typeName` and a collection of `propertyValues`. The `typeName` is used to create a potential instance of an object of that type. The `TopologyService` is used to create the instance. The `propertyValues` are a map of (name, value) pairs. The script iterates over the properties, inspects each one and tries to set that property on the object using the appropriate API.

What happens at the end is critical. Using the `topSvc.mergeData()` call, we send a potential instance of an object with several properties set. The `TopologyService` decides whether we need to create an object or simply update the definition for an existing one.

The following is an example call to this function:

```
def appsDev1 = createOrUpdateObject("Employee", [lastName:"Nichy",
firstName:"Frederick"]);
```

This creates a new `Employee` object. When using this function, you must include all the identity properties at least. You can include additional properties if you want.

Here is another example:

```
def appsDev1 = createOrUpdateObject("Employee", [lastName:"Nichy",
firstName:"Frederick", jobTitle:"Senior App Support", phoneNumber:"416-555-1354",
emailAddress:"frederick.nichy@mycloud.com"]);
```

With this function we almost have everything we need to create models using scripts. In order to enable iterative development, we need a way to delete old object instances. In this way, we can create a re-entrant Groovy script that cleans up obsolete objects.

The delete function is defined as follows:

```
def deleteObject(typeName, objName) {
  obj = createOrUpdateObject(typeName, objName);
  def args = [];
  args.add(obj);
  topSvc.deleteObjects(new HashSet(args));
}
```

Note that we use the `createOrUpdateObject` call to return the original object. Then we make a `TopologyService` call to delete it.

Connecting "Employees" to Host objects

At the end of the script, there is a bit of code which allows you to connect the employees to Host systems. This code simply finds some Host instances and manually attaches them to the employees. This code was included so that we can verify whether the alarm propagation works properly.

In a real example, you can add a dashboard that would allow an employee to express interest in objects. Behind the scenes, this dashboard would call code that looks something like this:

```
createOrUpdateObject("Employee", [lastName:"Nichy", firstName:"Frederick",
ownedItems:[topologyObject]]);
```

...where `topologyObject` is the object that the user has chosen as interesting. This is pretty much what is done in the Services Operations Console. It is all done using Dashboards and Groovy.

Visualizing the data

The last step is to build dashboards to visualize the data. The createOrg Groovy script provided in the appendix creates a nice organization.

Appendix: Groovy scripts

This appendix includes Groovy scripts used in the Data modeling tutorials.

createOrg Groovy script

Use the following script to populate with interesting data the Org Chart model presented in section Modeling example: The org chart.

```
def createOrUpdateObject(typeName, propertyValues) {
  topSvc = server.TopologyService;

  def type = topSvc.getType(typeName);
  objShell = topSvc.getObjectShell(type);
  propertyValues.each ({propertyName, propertyValue ->
    def prop = type.getProperty(propertyName);
    if (!prop.isMany()) {
      objShell.set(prop, propertyValue);
    }
    else if (propertyValue instanceof Collection) {
      objShell.getList(prop).addAll(propertyValue);
    }
    else {
      objShell.getList(prop).add(propertyValue);
    }
  });
  return topSvc.mergeData(objShell);
}

def getObject(typeName, objectName) {
  return createOrUpdateObject(typeName, [name:objectName]);
}

def deleteObject(typeName, objName) {
  obj = createOrUpdateObject(typeName, objName);
  def args = [];
  args.add(obj);
  topSvc.deleteObjects(new HashSet(args));
}

// Delete old employees
deleteObject("Manager", [lastName:"Monkey", firstName:"Big"]);
deleteObject("Manager", [lastName:"Monkey", firstName:"Little"]);
deleteObject("Employee", [lastName:"Chimp", firstName:"Pamela"]);
deleteObject("Manager", [lastName:"Strumpet", firstName:"Crandell"]);
deleteObject("Manager", [lastName:"Jo", firstName:"Mo"]);
deleteObject("Manager", [lastName:"Skate", firstName:"Chip"]);
deleteObject("Manager", [lastName:"Randolph", firstName:"Feelie"]);
deleteObject("Manager", [lastName:"Loman", firstName:"William"]);
deleteObject("Manager", [lastName:"Joe", firstName:"Jaffa"]);
```

```
deleteObject("Manager", [lastName:"Brickshaw", firstName:"Rick"]);
deleteObject("Manager", [lastName:"Beardly", firstName:"Sandals"]);

deleteObject("Employee", [lastName:"Nichy", firstName:"Frederick"]);
deleteObject("Employee", [lastName:"Froyd", firstName:"Ziggy"]);
deleteObject("Employee", [lastName:"Bonds", firstName:"Bartholomew"]);
deleteObject("Employee", [lastName:"Ruth", firstName:"Barbie"]);
deleteObject("Employee", [lastName:"Mathewson", firstName:"Christine"]);
deleteObject("Employee", [lastName:"Gently", firstName:"Dirck"]);
deleteObject("Employee", [lastName:"Dent", firstName:"Arthur"]);

// Delete old org structure
deleteObject("Organization", [orgName:"MyCloud"]);
deleteObject("Organization", [orgName:"IT"]);
deleteObject("Organization", [orgName:"Administration"]);
deleteObject("Organization", [orgName:"Accounting"]);
deleteObject("Organization", [orgName:"Human Resources"]);
deleteObject("Organization", [orgName:"Operations"]);
deleteObject("Organization", [orgName:"Sales"]);
deleteObject("Organization", [orgName:"Applications"]);
deleteObject("Organization", [orgName:"DBAs"]);
deleteObject("Organization", [orgName:"Infrastructure"]);

// Build up the model root and the main organization: MyCloud
def orgModel = createOrUpdateObject("OrganizationModel",
[name:"OrganizationModel"]);
def cloudOrg = createOrUpdateObject("Organization", [orgName:"MyCloud"]);
orgModel = createOrUpdateObject("OrganizationModel", [name:"OrganizationModel",
organizations:[cloudOrg]]);

// Big Monkey runs MyCloud
def bigMonkey = createOrUpdateObject("Manager", [lastName:"Monkey", firstName:"Big",
jobTitle:"CEO", phoneNumber:"416-555-1212", emailAddress:"big.monkey@mycloud.com",
organization:cloudOrg]);
cloudOrg = createOrUpdateObject("Organization", [orgName:"MyCloud",
orgLeader:bigMonkey]);

//  Big Monkey needs a personal assistant
def personalAssistant = createOrUpdateObject("Employee", [lastName:"Chimp",
firstName:"Pamela", jobTitle:"Personal Assistant", phoneNumber:"416-555-1200",
emailAddress:"pamela.chimp@mycloud.com", organization:cloudOrg]);

// MyCloud is a tech company, so IT is an important player
def cloudIT = createOrUpdateObject("Organization", [orgName:"IT"]);
def littleMonkey = createOrUpdateObject("Manager", [lastName:"Monkey",
firstName:"Little", jobTitle:"IT Manager", phoneNumber:"416-555-1213",
emailAddress:"little.monkey@mycloud.com", organization:cloudIT]);
cloudIT = createOrUpdateObject("Organization", [orgName:"IT",
orgLeader:littleMonkey]);
cloudOrg = createOrUpdateObject("Organization", [orgName:"MyCloud",
childOrgs:[cloudIT]]);

bigMonkey = createOrUpdateObject("Manager", [lastName:"Monkey", firstName:"Big",
directReports:[personalAssistant, littleMonkey]]);

// The rest of MyCloud is called "Operations".
def ops = createOrUpdateObject("Organization", [orgName:"Operations"]);
def opsManager = createOrUpdateObject("Manager", [lastName:"Strumpet",
firstName:"Crandell", jobTitle:"VP of Operations", phoneNumber:"416-555-1202",
emailAddress:"crandell.strumpet@mycloud.com", organization:ops]);
```

```groovy
ops = createOrUpdateObject("Organization", [orgName:"Operations",
orgLeader:opsManager]);

// Administration
def admin = createOrUpdateObject("Organization", [orgName:"Administration"]);
def adminManager = createOrUpdateObject("Manager", [lastName:"Jo", firstName:"Mo",
jobTitle:"Office Manager", phoneNumber:"416-555-1203",
emailAddress:"mo.jo@mycloud.com", organization:admin]);
admin = createOrUpdateObject("Organization", [orgName:"Administration",
orgLeader:adminManager]);

// Accounting
def accounting = createOrUpdateObject("Organization", [orgName:"Accounting"]);
def accountingManager = createOrUpdateObject("Manager", [lastName:"Skate",
firstName:"Chip", jobTitle:"VP Finance", phoneNumber:"416-555-1250",
emailAddress:"chip.skate@mycloud.com", organization:accounting]);
accounting = createOrUpdateObject("Organization", [orgName:"Accounting",
orgLeader:accountingManager]);

// Human Resources
def hr = createOrUpdateObject("Organization", [orgName:"Human Resources"]);
def hrManager = createOrUpdateObject("Manager", [lastName:"Feelie",
firstName:"Randolph", jobTitle:"Director of Human Resources", phoneNumber:"416-555-
1201", emailAddress:"randolph.feelie@mycloud.com", organization:hr]);
hr = createOrUpdateObject("Organization", [orgName:"Human Resources",
orgLeader:hrManager]);

// Sales
def sales = createOrUpdateObject("Organization", [orgName:"Sales"]);
def salesManager = createOrUpdateObject("Manager", [lastName:"Loman",
firstName:"William", jobTitle:"VP Sales", phoneNumber:"416-555-1400",
emailAddress:"william.loman@mycloud.com", organization:sales]);
sales = createOrUpdateObject("Organization", [orgName:"Sales",
orgLeader:salesManager]);

// Now for the IT subdepartments
// Applications group
def cloudApps = createOrUpdateObject("Organization", [orgName:"Applications"]);
def appManager = createOrUpdateObject("Manager", [lastName:"Joe", firstName:"Jaffa",
jobTitle:"Applications Team Lead", phoneNumber:"416-555-1300",
emailAddress:"jaffa.joe@mycloud.com", organization:cloudApps]);
cloudApps = createOrUpdateObject("Organization", [orgName:"Applications",
orgLeader:appManager]);

// DBA group
def cloudDB = createOrUpdateObject("Organization", [orgName:"DBAs"]);
def dbManager = createOrUpdateObject("Manager", [lastName:"Brickshaw",
firstName:"Rick", jobTitle:"DB Team Lead", phoneNumber:"416-555-1301",
emailAddress:"rick.bricksahw@mycloud.com", organization:cloudDB]);
cloudDB = createOrUpdateObject("Organization", [orgName:"DBAs",
orgLeader:dbManager]);

// SysAdmin group
def cloudInfra = createOrUpdateObject("Organization", [orgName:"Infrastructure"]);
def infraManager = createOrUpdateObject("Manager", [lastName:"Beardly",
firstName:"Sandals", jobTitle:"Infrastructure Team Lead", phoneNumber:"416-555-
1302", emailAddress:"sandals.beardly@mycloud.com", organization:cloudInfra]);
cloudInfra = createOrUpdateObject("Organization", [orgName:"Infrastructure",
orgLeader:infraManager]);

// Attach all the groups together in the org chart
```

```
cloudIT = createOrUpdateObject("Organization", [orgName:"IT", childOrgs:[cloudApps,
cloudDB, cloudInfra]]);
cloudOrg = createOrUpdateObject("Organization", [orgName:"MyCloud", childOrgs:[ops,
sales]]);
ops = createOrUpdateObject("Organization", [orgName:"Operations", childOrgs:[admin,
accounting, hr]]);

// Now let's add some reports to the IT groups.
// Apps group has two people
def appsDev1 = createOrUpdateObject("Employee", [lastName:"Nichy",
firstName:"Frederick", jobTitle:"Senior App Support", phoneNumber:"416-555-1354",
emailAddress:"frederick.nichy@mycloud.com", organization:cloudApps]);
def appsDev2 = createOrUpdateObject("Employee", [lastName:"Froyd",
firstName:"Ziggy", jobTitle:"Junior App Support", phoneNumber:"416-555-1355",
emailAddress:"ziggy.froyd@mycloud.com", organization:cloudApps]);
appManager = createOrUpdateObject("Manager", [lastName:"Joe",
firstName:"Jaffa",directReports:[appsDev1, appsDev2]]);


// There are three DBAs
def dbDev1 = createOrUpdateObject("Employee", [lastName:"Bonds",
firstName:"Bartholomew", jobTitle:"Senior DBA", phoneNumber:"416-555-1360",
emailAddress:"bartholomew.bonds@mycloud.com", organization:cloudDB]);
def dbDev2 = createOrUpdateObject("Employee", [lastName:"Ruth", firstName:"Barbie",
jobTitle:"Senior DBA", phoneNumber:"416-555-1361",
emailAddress:"barbie.ruth@mycloud.com", organization:cloudDB]);
def dbDev3 = createOrUpdateObject("Employee", [lastName:"Mathewson",
firstName:"Christine", jobTitle:"Junior DBA", phoneNumber:"416-555-1362",
emailAddress:"christine.mathewson@mycloud.com", organization:cloudDB]);
dbManager = createOrUpdateObject("Manager", [lastName:"Brickshaw",
firstName:"Rick",directReports:[dbDev1, dbDev2, dbDev3]]);

// There are two sysadmins
def infraDev1 = createOrUpdateObject("Employee", [lastName:"Gently",
firstName:"Dirck", jobTitle:"Windows Sysadmin", phoneNumber:"416-555-1303",
emailAddress:"dirck.gently@mycloud.com", organization:cloudInfra]);
def infraDev2 = createOrUpdateObject("Employee", [lastName:"Dent",
firstName:"Arthur", jobTitle:"Unix Sysadmin", phoneNumber:"416-555-3141",
emailAddress:"arthur.dent@mycloud.com", organization:cloudInfra]);
infraManager = createOrUpdateObject("Manager", [lastName:"Beardly",
firstName:"Sandals",directReports:[infraDev1, infraDev2]]);

// Now let's add some random hosts
// Now add some stuff to the services to make them meaningful
// Requires that you have some Host objects
allHosts = #!Host#.getTopologyObjects();
if (allHosts.size() >= 20) {
  createOrUpdateObject("Employee", [lastName:"Nichy", firstName:"Frederick",
ownedItems:[allHosts[0], allHosts[1]]]);
  createOrUpdateObject("Employee", [lastName:"Froyd", firstName:"Ziggy",
ownedItems:[allHosts[2], allHosts[3]]]);
  createOrUpdateObject("Employee", [lastName:"Bonds", firstName:"Bartholomew",
ownedItems:[allHosts[4], allHosts[5]]]);
  createOrUpdateObject("Employee", [lastName:"Ruth", firstName:"Barbie",
ownedItems:[allHosts[6], allHosts[7]]]);
  createOrUpdateObject("Employee", [lastName:"Mathewson", firstName:"Christine",
ownedItems:[allHosts[8], allHosts[9]]]);
  createOrUpdateObject("Employee", [lastName:"Gently", firstName:"Dirck",
ownedItems:[allHosts[10], allHosts[11]]]);
  createOrUpdateObject("Employee", [lastName:"Dent", firstName:"Arthur",
ownedItems:[allHosts[12], allHosts[13]]]);
```

```
   createOrUpdateObject("Manager", [lastName:"Joe", firstName:"Jaffa",
ownedItems:[allHosts[14], allHosts[15]]]);
   createOrUpdateObject("Manager", [lastName:"Brickshaw", firstName:"Rick",
ownedItems:[allHosts[16], allHosts[17]]]);
   createOrUpdateObject("Manager", [lastName:"Beardly", firstName:"Sandals",
ownedItems:[allHosts[18], allHosts[19]]]);

}
```

Appendix: Internal database schema

This appendix provides basic information on the internal database used by Foglight.

> **CAUTION: We do not recommend directly accessing the database in your own code or customizations. The Foglight database schema may change at any time. Upgrading Foglight will not migrate any customizations you make that reference the database directly.**

- acl_class Table
- acl_entry Table
- acl_object_identity Table
- acl_sid Table
- agent_client_defaults Table
- agent_config_binder Table
- agent_dc_manager_schedule_ids Table
- agent_dc_manager_state Table
- agent_manager_state Table
- alarm_alarm Table
- alarm_annotations Table
- alarm_loc_msg Table
- auditing_log Table
- baseline_config Table
- baseline_config_properties Table
- baseline_engine_profile Table
- baseline_observation_profile Table
- cartridge_cartridge_relation Table
- cartridge_components Table
- cartridge_installed_cartridges Table
- cartridge_items Table
- credential_data Table
- credential_lockbox Table
- credential_mapping Table
- credential_mapping_entry Table
- credential_order Table
- credential_policy Table
- current_version Table
- database_instance_id Table

- database_version Table
- derivation_calculation Table
- derivation_complex_definition Table
- derivation_definition Table
- fgl4_migration_agent Table
- fgl4_migration_data_span Table
- fgl4_migration_dcm Table
- fgl4_migration_host Table
- fgl4_migration_host_mapping Table
- fgl4_migration_log Table
- fgl4_migration_server Table
- incident_affected_objects Table
- incident_incident Table
- incident_linked_alarms Table
- incident_problem_ticket Table
- incident_problem_tickets Table
- licensing_licenses Table
- mgmt_object_size Table
- mgmt_observation_size Table
- mgmt_timeslice Table
- mgmt_timeslice_data_avail Table
- model_association Table
- model_property_formula Table
- model_query_criteria Table
- obs_binary_* Tables
- obs_metric_aggregate_* Tables
- obs_metric_scalar_* Tables
- obs_string_* Tables
- pcm_encoded_data Table
- persistable_config_model Table
- persistable_script Table
- persistence_column_mapping Table
- persistence_db_column Table
- persistence_db_schema Table
- persistence_db_table Table
- persistence_grouping_policy Table
- persistence_lifecycle Table
- persistence_lifecycle_period Table
- persistence_obs_key_purge_age Table
- persistence_obs_purge Table

- sec_group Table
- sec_group_nesting Table
- sec_group_role_match Table
- sec_grouprole Table
- sec_jaas_source Table
- sec_object Table
- sec_object_mask Table
- sec_object_permission Table
- sec_object_type Table
- sec_permission Table
- sec_permission_def Table
- sec_policy Table
- sec_resource Table
- sec_role Table
- sec_user_alias Table
- sec_user_obj_permission Table
- sec_user_res_permission Table
- sec_usergroup Table
- sec_userrole Table
- sec_x_attribute Table
- sec_x_attribute_value Table
- tagging_service_mapping Table
- threshold_bound Table
- threshold_config Table
- topology_activity_calendar Table
- topology_activity_upgrade Table
- topology_object Table
- topology_object_history Table
- topology_property Table
- topology_property_annotation Table
- topology_property_history Table
- topology_property_name Table
- topology_property_value Table
- topology_service_state Table
- topology_type Table
- topology_type_annotation Table
- topology_type_history Table
- upgrade_pending_operations Table
- wcf_groups_by_cartridges Table
- wcf_resources Table

acl_class Table

Spring security ACL Table: defines the domain object types to which ACLs apply.

Table 1. acl_class Table

Fields	
id	The primary key
class	Java® class name of the object

acl_entry Table

Spring security ACL Table: stores the ACL permissions which apply to a specific object identity and security identity.

Table 2. acl_entry Table

Fields	
id	The primary key
sid	Security identity
acl_object_identity	ACL object identity
ace_order	Ace order
mask	Mask
granting	A flag to indicate whether permission is granted
audit_success	A flag to indicate whether audit success
audit_failure	A flag to indicate whether audit failure

acl_object_identity Table

Spring security ACL Table: stores the object identity definitions of specific domain object.

Table 3. acl_object_identity Table

Fields	
id	The primary key
entries_inheriting	A flag to indicate whether parent ACL entries inherit into the current ACL
object_id_class	Object class ID
object_id_identity	Object identity
parent_object	Parent object
owner_sid	ACL object owner security identity

acl_sid Table

Spring security ACL Table: stores the security identities recognized by the ACL system.

Table 4. acl_sid Table

Fields	
id	The primary key
sid	Security identity
principal	A flag to indicate whether security identity is principal

agent_client_defaults Table

Holds information about the default agent management client.

Table 5. agent_client_defaults Table

Fields	
acd_holder_id	Identifies the kind of default agent management client
acd_host_name	The host name of the agent management client
acd_local_id	The ID of the agent management client (unique within the host)

agent_config_binder Table

Records the binding between agents and agent configurations.

Table 6. agent_config_binder Table

Fields	
config_id	Unique ID of agent configuration binding
adapter	Namespace of the agent
agent_type	Type of the agent
agent_name	Name of the agent
known_config_name	Known name of the agent configuration
resolved_config_uuid	ID of the agent configuration bond to the agent

agent_dc_manager_schedule_ids Table

Links agent data collection (a.k.a. blackout) schedules to agents (many-to-one).

Table 7. agent_dc_manager_schedule_ids Table

Fields	
magentid	Agent ID
scheduleid	Schedule ID

agent_dc_manager_state Table

Holds information about the pre-configured agent data collection states.

Table 8. agent_dc_manager_state Table

Fields	
magentid	Agent ID
mdatacollectiondisabled	A flag indicating whether data collection should be disabled (1) or enabled (0) for the associated agent

agent_manager_state Table

Holds various information about agents (one agent per row).

Table 9. agent_manager_state Table

Fields	
mid	Agent ID
mnamespace	Agent's namespace
magentname	Agent's name
moriginalname	Whether the name is original (1) or it was modified (0) to bring it into compliance with agent name restrictions
madapterjmxname	The JMX name of the agent adapter that manages this agent
madaptername	Agent adapter's short name
magenttype	Agent type
mactivated	Whether the agent has been activated (1) or not (0)
mactivatable	Whether the agent can be activated (1) or not (0) in principle
mfromtemplate	Not used
mhostname	The last known host name of this agent
mobsolete	Whether the agent is obsolete (1) or not (0)
mhidden	Whether the agent is hidden (1) or not (0)
magentdisplayname	Agent's display name
magentversion	The last known version of this agent
magentbuildid	The last known build number of this agent
mremotehomehostname	The last known host name of the agent manager that controls this agent
mremotehomelocalid	The last known ID of the agent manager that controls this agent
mremotehomedisplayname	The last known display name of the agent manager that controls this agent

alarm_alarm Table

Holds alarm data.

Table 10. alarm_alarm Table

Fields	
alarm_id	Alarm ID (sequential)
id	Alarm UUID
source_id	The ID of the entity (e.g. a rule) that produced the alarm
message	The alarm's default message
topology_object_id	The ID of the topology object associated with this alarm (UUID)
is_cleared	Whether the alarm has been cleared (1) or not (0)
is_acknowledged	Whether the alarm has been acknowledged (1) or not (0)
cleared_time	Alarm clearing time
created_time	Creation time
severity	Alarm's severity code
cleared_by	Description of the entity that cleared the alarm (e.g. user name)
ack_time	Acknowledgement time
ack_by	Description of the entity that acknowledged the alarm (e.g. user name)
source_name	Name of the entity that created the alarm (e.g. rule name)
rule_id	The ID of the related rule, if the alarm was created by a rule
user_defined_data	Extra alarms data specified by the creator of the alarm
auto_ack	Indicates whether a user has instructed the server to acknowledge this alarm and certain related alarms automatically

alarm_annotations Table

Holds alarm annotations.

Table 11. alarm_annotations Table

Fields	
annotation_id	Alarm annotation ID (UUID)
text	Annotation text
username	The name of the user that created this annotation
created_time	Annotation creations time
alarm_id	The UUID of the annotated alarm

alarm_loc_msg Table

Holds localized alarm messages.

Table 12. alarm_loc_msg Table

Fields	
alarm_id	UUID of an alarm
message	Localized message
locale	Locale ID (according to Java® conventions)

auditing_log Table

Holds auditing information.

Table 13. auditing_log Table

Fields	
id	Auditing Record ID
name	Name of the related entity
service	Internal name of the related service
method	Internal name of the related method
entity_id	ID of the related entity
user_name	Name of the user whose credentials were in effect during the operation
time	Operation invocation time
data	Extra auditing data
old_version	Old version of the related entity (before the operation)
new_version	New version of the related entity (after the operation)
service_id	Internal ID of the related service

baseline_config Table

Holds baseline computation configuration (one per row).

Table 14. baseline_config Table

Fields	
bc_version_id	The version ID of this baseline configuration
bc_id	The ID of this baseline configuration
bc_cartridge_name	The name of the cartridge from which this baseline configuration was deployed
bc_cartridge_version	The version of the cartridge from which this baseline configuration was deployed

Table 14. baseline_config Table

Fields	
bc_created_timestamp	Baseline configuration creation/modification timestamp
bc_user_created	Whether this configuration was created/edited by a user (1) or whether it is still the same as it was in the cartridge (0)
bc_name	Baseline configuration name
bc_type	Baseline configuration name
bc_type_version	The version of this baseline configuration type
bc_topology_query	The topology query used to select objects for baselining using this configuration
bc_observation_name	The name of the observation/metric to be baselined

baseline_config_properties Table

Holds extra properties for baseline configurations (many-to-one), references baseline_config.

Table 15. baseline_config_properties Table

Fields	
bcp_version_id	The version ID of this baseline configuration
bcp_name	Property name
bcp_value	Property value
bcp_index	Property position withing the whole property list (zero-based)

baseline_engine_profile Table

Holds general engine-specific computation data for baseline engines.

Table 16. baseline_engine_profile Table

Fields	
bep_engine_id	The name of the baseline engine using the table
bep_tag	A tag provided by the engine to look up individual pieces of profile information
bep_data	The data stored by the engine

baseline_observation_profile Table

Holds observation-specific computation data for baseline engines.

Table 17. baseline_observation_profile Table

Fields	
bop_object_id	The ID of the topology object for which this data was computed
bop_observation_name	The name of the observation for which this data was computed
bop_config_id	The identifier of the baseline configuration for which this data was computed
bop_tag	A tag provided by the baseline engine to identify individual pieces of profile information
bop_data	An opaque blob of data stored by the baseline engine

cartridge_cartridge_relation Table

Records relationships (dependency, incompatibility etc.) between cartridges.

Table 18. cartridge_cartridge_relation Table

Fields	
relationship_id	Unique ID of the relationship
relationship_type	Type of the relationship
related_cartridge_name	Name of the related cartridge
related_cartridge_version	Version of the related cartridge
dependency_match	The method for the relationship to match different version of related cartridges
cartridge_id	ID of the cartridge that has the relationship
relationship_index	The index of the relation in the list of relationships that the cartridge has

cartridge_components Table

Records components in cartridges.

Table 19. cartridge_components Table

Fields	
component_id	Unique ID of the component
component_id_name	Name of the component
cartridge_version	Version of the component
cartridge_build_id	Build ID of the component
component_id_creation_date	Creation date of the component
component_id_author	Author of the component
component_type	Type of the component

Table 19. cartridge_components Table

Fields	
component_deployed	If the component has been deployed
federate_deploy_type	How the component should be deployed in federation setup
component_domain	Domain of the component
component_deployment_item_name	The name of the item for deployment in the component
attributes	Attributes of the component
cartridge_id	ID of the cartridge that the component belongs to
component_index	The index of the component in the list of components that the cartridge has

cartridge_installed_cartridges Table

Records cartridges installed in the server.

Table 20. cartridge_installed_cartridges Table

Fields	
cartridge_id	Unique ID of the cartridge
cartridge_name	Name of the cartridge
cartridge_version	Version of the cartridge
cartridge_build_id	Build ID of the cartridge
cartridge_creation_date	Creation date of the cartridge
cartridge_author	Author of the cartridge
cartridge_foglight_version	Minimal version of Forge server required by the cartridge
cartridge_final_flag	If the cartridge is still in development
cartridge_hash_algorithm	Hashing algorithm used when constructing the cartridge
cartridge_deployed	If the cartridge has been deployed
cartridge_pending_dependency	If the cartridge is waiting for a cartridge that it depends on to be enabled
cartridge_type	Type of the cartridge
cartridge_domain	Domain of the cartridge
classloading_style	Class loading style to use for the cartridge

cartridge_items Table

Records items in cartridge components.

Table 21. cartridge_items Table

Fields	
item_id	Unique ID of the item
item_type	Type of the item
item_name	Name of the item
item_size	Size in byte of the item
item_hash	Hashcode of the time
item_source	Source location for the item
item_directory	Directory in which the item resides
component_id	ID of component that the item belongs to
item_index	Index of the item in the list of items that the component has

credential_data Table

Holds encrypted credentials (one per row).

Table 22. credential_data Table

Fields	
cd_lockbox_id	The ID of the lockbox that contains the credential varchar(36) not null
cd_id	Credential ID
cd_type	Credential type
cd_purpose	Not used
cd_name	Credential name
cd_direct_access	Whether direct access to the secret part of this credential is allowed to agents (this flag is duplicated in the encrypted properties)
cd_properties	Secret credential properties (encrypted)
cd_index	Credential position within the lockbox (zero-based)

credential_lockbox Table

Holds information about credential lockboxes.

Table 23. credential_lockbox Table

Fields	
cl_id	Lockbox ID
cl_name	Lockbox name

Table 23. credential_lockbox Table

Fields	
cl_open_pwd_enc	The encrypted version of the lockbox password if the lockbox is "open"
cl_enc_algorithm	Encryption algorithm (for public/private key cryptography)
cl_key_format	The format of public and private key byte arrays
cl_public_key	The public key for this lockbox
cl_private_key_lb	The private key for this lockbox (encrypted with the lockbox password)

credential_mapping Table

Holds information about which credentials have resource mappings.

Table 24. credential_mapping Table

Fields	
cme_credential_id	Credential ID
cme_lockbox_id	Lockbox ID

credential_mapping_entry Table

Holds credential resource mapping data.

Table 25. credential_mapping_entry Table

Fields	
cme_credential_id	Credential ID
cme_record_id	Resource mapping record ID
cme_field_id	Resource mapping field ID
cme_condition_type	Resource mapping condition type
cme_value	Resource mapping model/test value
cme_negated	Whether this condition is negated (1) or not (0)
cme_null_matches	Whether null values provided in credential queries should match this condition (1) or not (0)
cme_index	Position of this condition within the record (zero-based)

credential_order Table

Defines a global partial order across all credentials.

Table 26. credential_order Table

Fields	
co_credential_id	Credential ID
co_index	Position (zero-based); smaller values mean higher position/priority

credential_policy Table

Holds credential policy data.

Table 27. credential_policy Table

Fields	
cp_lockbox_id	Lockbox ID
cp_id	Policy ID
cp_credential_id	Credential ID
cp_type	Policy type
cp_critical	Whether the policy is critical (1) or not (0)
cp_properties	Policy properties
cp_index	Policy position within the list of all policies for the associated credential (zero-based)

current_version Table

Holds information about the current version of various server configuration items.

Table 28. current_version Table

Fields	
id	UUID of a configuration item
version_id	UUID of a particular version of the configuration item
is_user_created	Whether this version was created/modified by a user (1) or not (0)
service	Internal ID of the service that uses the configuration item

database_instance_id Table

Holds internal ID of the Management Server database.

Table 29. database_instance_id Table

Fields	
col_index	Primary key (internal)
instance_id	Database instance UUID

database_version Table

Holds information about the current version of this database schema, and contains the history of database schema versions (history of upgrades) since first install.

Table 30. database_version Table

Fields	
dbv_version	The database schema version number
dbv_date_applied	The date that the server was upgraded to this version

derivation_calculation Table

Records calculations in complex derivation definition.

Table 31. derivation_calculation Table

Fields	
calculation_id	Unique ID of the calculation
description	Description of the calculation
domain_query	Query that determines all the topology objects that the calculation applies to
expression	Expression used for calculation
trigger_type	Way the trigger the calculation
firing_interval	Interval in seconds for a time-driven calculation to be triggered
trigger_without_data	If a time-driven or schedule-driven calculation should be triggered without new metric data that affects the calculation
schedule_trigger_timing	How a schedule-driven calculation should be triggered based on the starting and ending of the schedule
triggering_schedule	The schedule that triggers a schedule-driven calculation
derivation_version_id	The version ID of the complex derivation definition that contains the calculation
calculation_index	Index of the calculation in the list of calculations that the complex derivation definition

derivation_complex_definition Table

Records complex derivation definitions.

Table 32. derivation_complex_definition Table

Fields	
version_id	Unique version ID of derivation definition
id	ID of derivation definition
created	Creation time of the derivation definition
user_created	If this version of derivation definition is modified by user
cartridge_name	Name of cartridge that defines the derivation definition
cartridge_version	Version of cartridge that defines the derivation definition
property_name	Name of the observation the derivation is defined for
comments	Description of the derivation definition
value_type	Type of return value of the derivation definition
unit_name	Unit of the return value of the derivation definition

derivation_definition Table

Records simple derivation definitions, which is used only in older versions of Management Server and is now deprecated.

fgl4_migration_agent Table

This table is related to Foglight 4 migration functionality, and is no longer in use.

fgl4_migration_data_span Table

This table is related to Foglight 4 migration functionality, and is no longer in use.

fgl4_migration_dcm Table

This table is related to Foglight 4 migration functionality, and is no longer in use.

fgl4_migration_host Table

This table is related to Foglight 4 migration functionality, and is no longer in use.

fgl4_migration_host_mapping Table

This table is related to Foglight 4 migration functionality, and is no longer in use.

fgl4_migration_log Table

This table is related to Foglight 4 migration functionality, and is no longer in use.

fgl4_migration_server Table

This table is related to Foglight 4 migration functionality, and is no longer in use.

incident_affected_objects Table

Holds UUIDs of topology objects related a particular incident.

Table 33. incident_affected_objects Table

Fields	
incident_id	Incident ID
object_id	Topology Object UUID

incident_incident Table

Holds incident data.

Table 34. incident_incident Table

Fields	
incident_id	Incident ID
reported_start	The start time of the incident
reported_end	The end time of the incident
severity	Incident's severity
message	Incident's message
start_time	The time when the incident was reported to FMS
end_time	The time when the incident was closed in FMS
created_by	Description of the entity that reported the incident (e.g. user name)
acknowledged_by	Description of the entity that acknowledged the incident (e.g. user name)
acknowledged_time	Acknowledgement time
ended_by	Description of the entity that closed the incident (e.g. user name)
last_updated	The time when this incident report was last updated
last_updated_by	Description of the entity that updated the incident last (e.g. user name)

Table 34. incident_incident Table

Fields	
type	Incident type
parent	ID if the parent incident (if applicable)

incident_linked_alarms Table

Holds IDs of alarms related to a particular incident.

Table 35. incident_linked_alarms Table

Fields	
incident_id	Incident ID
alarm_id	Alarm ID

incident_problem_ticket Table

Holds problem ticket data.

Table 36. incident_problem_ticket Table

Fields	
id	Problem ticket ID
source_id	ID of the source of this problem ticket
ticket_number	Problem ticket number
status	Problem ticket status
updated_by	Description of the entity that updated the problem ticket (e.g. user name)
ticket_url	URL associated with the problem ticket
created_time	Problem ticket creation time

incident_problem_tickets Table

Holds IDs of problem tickets related to a particular incident.

Table 37. incident_problem_tickets Table

Fields	
incident_id	Incident ID
ticket_id	Problem ticket ID

licensing_licenses Table

Holds installed licenses' data.

Table 38. licensing_licenses Table

Fields	
serial	Serial number of a licence
license	The licence file data
license_installed	The time when the license was installed
is_deleted	Whether the license has been deleted (1) or not (0)

mgmt_object_size Table

Records size information for topology objects.

Table 39. mgmt_object_size Table

Fields	
mobj_object_id	The topology object ID
mobj_topology_rows	The number of topology rows used by the object
mobj_topology_total_rows	The total number of rows used by the object and its dependents
mobj_topology_agg_rows	The total number of rows used by the object and its contained objects
mobj_memory_size	The size of the object in-memory
mobj_memory_total_size	The total size in memory of the topology object and its dependents
mobj_memory_agg_size	The total size in memory of the topology object and its contained objects
mobj_observation_rows	The number of observation rows persisted for the object
mobj_observation_total_rows	The total number of observation rows used by the object and its dependents
mobj_observation_agg_rows	The total number of observation rows used by the objects and its contained objects
mobj_observation_size	The total size of the observations persisted for the object
mobj_observation_total_size	The total size of observations persisted for the object and its dependents
mobj_observation_agg_size	The total size of observations persisted for the object and its contained objects

mgmt_observation_size Table

Records the size of data persisted for objects in timeslices.

Table 40. mgmt_observation_size Table

Fields	
mobs_id	The primary key
mobs_timeslice_id	The timeslice ID
mobs_object_id	The topology object ID
mobs_number_rows	The number of observation rows stored for the object in the timeslice
mobs_total_number_rows	The number of observation rows stored for the object and its dependents in this timeslice

mgmt_timeslice Table

Stores the summary information for individual timeslices.

Table 41. mgmt_timeslice Table

Fields	
mt_id	The primary key
mt_timeslice_id	The timeslice ID
mt_generation	The timeslice generation
mt_storage_type	The storage type
mt_start_time	The timeslice start time
mt_end_time	The timeslice end time
mt_number_rows	The number of rows in the timeslice
mt_size	The size of the timeslice
mt_rollup_timestamp	The timestamp of the last rollup performed on the timeslice
mt_purge_age	The timestamp of the last purge operation performed on the timeslice

mgmt_timeslice_data_avail Table

Stores the details of which observations have data stored in particular timeslices.

Table 42. mgmt_timeslice_data_avail Table

Fields	
mda_id	The primary key
mda_timeslice_id	The timeslice ID
mda_data_availability	A BLOB containing the data availability information

model_association Table

Stores the model association configuration.

Table 43. model_association Table

Fields	
ma_version_id	The version ID
ma_id	The ID of the model association configuration
ma_name	The name of the model association configuration
ma_create_source	A flag to indicate whether the source object should be created for the association when target objects are found
ma_update_delay	The delay before processing the association after being notified of a possible change
ma_source	The ID of the query used to find source objects
ma_created_timestamp	The timestamp at which the configuration was created
ma_is_user_created	A flag that indicates whether the configuration is user edited
ma_cartridge_name	The name of the cartridge that contained the configuration
ma_cartridge_version	The version of the cartridge that contained the configuration

model_property_formula Table

Stores the details of how to produce a topology property value in a model association configuration.

Table 44. model_property_formula Table

Fields	
mpf_id	The ID of the formula
mpf_type	Indicates whether this is a constant value, topology query, criteria query or an instance-specific query
mpf_constant_value	The value used for constant property values
mpf_query_text	The query used for topology query formulas
mpf_topology_type	The topology type used for criteria queries
mpf_property	The property that holds the query for instance specific queries
mpfm_assoc_id	The model association ID
mpfm_property_name	The property that the formula applies to

model_query_criteria Table

Stores the details of a criteria query used in a model association configuration.

Table 45. model_query_criteria Table

Fields	
mqc_id	The primary key
mqc_name	The property name
mqc_comparator	The comparator used in the query
mqc_value	The value against which the property is compared
mqc_query_id	The ID of the property formula
mqc_index integer	The index position of this criteria query in the property formula

obs_binary_* Tables

Tables to store the values of ComplexObservations.

Table 46. obs_binary_* Tables

Fields	
ob_id	The primary key
ob_object_id	The topology object ID
ob_observation_id	The observation property ID
ob_start_time	The start time
ob_end_time	The end time
ob_sampled_period	The sampled period
ob_root	The summary information for the observation
ob_value	The observation value

obs_metric_aggregate_* Tables

Tables to store aggregate metric values.

Table 47. obs_metric_aggregate_* Tables

Fields	
oma_id	The primary key
oma_object_id	The topology object ID
oma_observation_id	The observation property ID
oma_start_time	The start time
oma_end_time	The end time
oma_sampled_period	The sampled period
oma_count	The number of source values in the aggregate
oma_sum	The sum of the metrics

Table 47. obs_metric_aggregate_* Tables

Fields	
oma_min	The minimum value in the aggregate
oma_max	The maximum value in the aggregate
oma_sum_squares	The sum of squares

obs_metric_scalar_* Tables

Tables to store scalar metric values.

Table 48. obs_metric_scalar_* Tables

Fields	
oms_id	The primary key
oms_object_id	The topology object ID
oms_observation_row	The observation index row
oms_end_time	The end time
oms_sampled_period	The sampled period
oms_obs_0	The column 0 value
oms_obs_1	The column 1 value
oms_obs_2	The column 2 value
oms_obs_3	The column 3 value
oms_obs_4	The column 4 value
oms_obs_5	The column 5 value
oms_obs_6	The column 6 value
oms_obs_7	The column 7 value
oms_obs_8	The column 8 value
oms_obs_9	The column 9 value

obs_string_* Tables

Tables to store string observation values.

Table 49. obs_string_* Tables

Fields	
os_id	The primary key
os_object_id	The topology object ID
os_observation_id	The observation property ID
os_start_time	The start time

Table 49. obs_string_* Tables

Fields	
os_end_time	The end time
os_sampled_period	The sampled period
os_index	The value index
os_value	The observation value

pcm_encoded_data Table

Records encoded agent configuration.

Table 50. pcm_encoded_data Table

Fields	
version_id	The version ID of the agent configuration this encoded data belongs to
encoded_data	Encoded compressed agent configuration

persistable_config_model Table

Record agent configuration.

Table 51. persistable_config_model Table

Fields	
version_id	Unique version ID of the agent configuration
id	ID of the agent configuration
created	Creation time for the agent configuration
user_created	If the agent configuration is created/modified by user
is_clone	If the agent configuration is a clone from other agent configuration
adapter	Name space of the agent configuration
encoding_system	Encoding system used by the agent configuration
config_type	Type of the agent configuration
config_qualifier	Qualifier of the agent configuration
sharing_name	Sharing name of the agent configuration
agent_type	Agent type of the agent configuration
known_name	Known name of the agent configuration
type_id	ID of the type of the agent configuration
identify_by_known_name	If the agent configuration is identified by its known name
cartridge_name	Name of the cartridge this agent configuration comes from

Table 51. persistable_config_model Table

Fields	
cartridge_version	Version of the cartridge this agent configuration comes from
is_agent_volatile	If the agent configuration is modified by agent itself

persistable_script Table

Holds scripts deployed into Management Server.

Table 52. persistable_script Table

Fields	
config_version_id	The version ID of the configuration item that contains this script
config_id	The entity ID of the configuration item that contains this script
namespace_name	Script's namespace
java_class_name	The name of the associated Java® class (if applicable)
script_name	Script name
script_descr	Script description
script_text	Actual body of the script (if not backed by a Java® class)
language_specifier	Identifies the script's programming language
tc_created_timestamp	The time when the script was deployed
tc_is_user_created	Whether the script was created/edited by a user (1) or not (0)
tc_cartridge_name	The name of the associated cartridge
tc_cartridge_version	The version of the associated cartridge
return_type	The declared return value type
return_description	Plain text description of the return value

persistence_column_mapping Table

Records the column mapping details of a grouping policy.

Table 53. persistence_column_mapping Table

Fields	
cm_id	The primary key
cm_grouping_policy_id	The grouping policy ID
cm_column_name	The column name
cm_expression	The expression used to produce the column value
cm_observation_property	The observation property used to populate the column
cm_aggregation	The aggregation function applied to the observation

Table 53. persistence_column_mapping Table

Fields	
cm_observation_alias	The alias used for the observation property
cm_units_name	The units in which the observation value is stored
cm_is_identity	Flag indicating whether the property is an identity property

persistence_db_column Table

Records the details of a column in a table used in a grouping policy.

Table 54. persistence_db_column Table

Fields	
col_id	The primary key
col_table_id	The ID of the table
col_name	The column name
col_sql_expr	The SQL expression used to produce the column value
col_aggregation_fn	The aggregation function used when rolling up the property

persistence_db_schema Table

Records the details of a schema configured for use in a grouping policy.

Table 55. persistence_db_schema Table

Fields	
schema_version_id	The version ID
schema_id	The ID of the schema configuration
schema_name	The schema name
schema_created_timestamp	The timestamp at which the configuration was created
schema_is_user_created	Flag indicating whether the configuration was user edited
schema_cartridge_name	The cartridge from which the configuration originated
schema_cartridge_version	The version of the originating cartridge

persistence_db_table Table

Records the details of a table used in a grouping policy.

Table 56. persistence_db_table Table

Fields	
tbl_id	The primary key
tbl_schema_id	The ID of the containing schema

Table 56. persistence_db_table Table

Fields	
tbl_name	The table name
tbl_max_size	The maximum size of the table
tbl_low_water_mark	The high water mark for the table size
tbl_high_water_mark	The low water mark for the table size
tbl_auto_increment_col	The auto-increment column in the table
tbl_end_time_col	The name of the column that stores the end-time
tbl_start_time_col	The name of the column that stores the start-time

persistence_grouping_policy Table

Records the configuration of a grouping policy.

Table 57. persistence_grouping_policy Table

Fields	
gp_version_id	The ID of the policy version
gp_id	The grouping policy ID
gp_name	The grouping policy name
gp_schema_id	The ID of the schema to which the policy applies
gp_table_name	The name of the table into which the policy writes data
gp_lifecycle_id	The lifecycle used to control the rollup and purging of the inserted data
gp_scoping_expr	The scoping query for objects processed by the policy
gp_period	The period (in milliseconds) with which the policy inserts data
gp_conditional_expr	The conditional expression used to determine whether data is to be inserted for an object
gp_discriminator	The SQL expression used to identify records in the target table that are processed by this policy
gp_created_timestamp	The time at which the policy was created
gp_is_user_created	A flag indicating whether the policy was user edited
gp_cartridge_name	The name of the cartridge that deployed the policy
gp_cartridge_version	The version of the cartridge that deployed the policy

persistence_lifecycle Table

Records the configuration of a lifecycle used to control the rollup and purging of data.

Table 58. persistence_lifecycle Table

Fields	
lc_version_id	The version ID
lc_id	The lifecycle ID
lc_name	The lifecycle name
lc_public	A flag indicating whether the lifecycle is "public" and available for use outside of the containing cartridge
lc_created_timestamp	The timestamp at which the configuration was created
lc_is_user_created	A flag indicating whether the configuration is user-edited
lc_cartridge_name	The name of the cartridge that deployed the configuration
lc_cartridge_version	The version of the cartridge that deployed the configuration

persistence_lifecycle_period Table

Records the configuration of a rollup/purge period in a lifecycle.

Table 59. persistence_lifecycle_period Table

Fields	
lp_id	The primary key
lp_age	The age (in milliseconds) at which the rollup should occur
lp_granularity	The granularity (in milliseconds) to which the data should be aggregated
lp_lifecycle_id	The ID of the lifecycle that contains the period

persistence_obs_key_purge_age Table

A "temporary" table used during purge operations.

Table 60. persistence_obs_key_purge_age Table

Fields	
okp_object_id	The object ID
okp_observation_id	The observation ID
okp_purge_age	The purge age

persistence_obs_purge Table

Records the details of user-initiated observation purge operations.

Table 61. persistence_obs_purge Table

Fields	
pop_id	The primary key
pop_topology_query	The topology query that selects the objects that are to be purged
pop_purge_age	The purge age (in milliseconds)
pop_created_by_user	The user that created the purge operations
pop_created_timestamp	The time at which the operation was created

persistence_obs_purge_age Table

A "temporary" table used during a purge operation to store the purge age of an observation property.

Table 62. persistence_obs_purge_age Table

Fields	
popa_id	The primary key
popa_type_id	The topology type ID
popa_observation_id	The observation property ID
popa_is_property_override	Flag indicating whether the purge age is from a property-specific retention policy
popa_purge_age	The purge age (in milliseconds)

persistence_observation_index Table

Records the details of how observations are mapped to scalar metric tables.

Table 63. persistence_observation_index Table

Fields	
poi_id	The primary key
poi_type_id	The topology type ID
poi_property_id	The observation property ID
poi_row integer	The row to which the observation is mapped
poi_column	The column to which the observation is mapped

persistence_operation Table

Records the progress of long running operations performed on observation tables.

Table 64. persistence_operation Table

Fields	
po_id	The primary key
po_version	The version
po_operation_type	The type of operation being performed
po_start_time	The time that the operation was initiated
po_details	The details of the operation

persistence_retention_policy Table

Records the details of retention policy configuration.

Table 65. persistence_retention_policy Table

Fields	
rp_version_id	The version ID
rp_id	The retention policy ID
rp_name	The retention policy name
rp_type_name	The topology type to which the policy applies
rp_object_id	The unique Id of the topology object to which the policy applies
rp_property_name	The name of the observation property to which the policy applies
rp_cache_duration	The minimum duration (in milliseconds) that values have to remain in the cache
rp_lifecycle_id	The ID of the lifecycle that controls the observations
rp_created_timestamp	The timestamp at which the policy was created
rp_is_user_created	A flag indicating whether the policy is user-edited
rp_cartridge_name	The cartridge name
rp_cartridge_version	The cartridge version

persistence_rollup_progress Table

Records the progress of a rollup operation.

Table 66. persistence_rollup_progress Table

Fields	
prp_id	The primary key
prp_handler	The handler performing the rollup
prp_start_time	The start time of the rollup time range

Table 66. persistence_rollup_progress Table

Fields	
prp_end_time	The end time of the rollup time range
prp_partition	The partition being rolled up
prp_lifecycle_id	The lifecycle being rolled up
prp_property_based	Indicates whether the progress of property-based rollups is being tracked
prp_custom_rollup	Indicates whether the progress of custom rollups is being tracked
prp_last_object_id	The ID of the last object processed by the rollup

persistence_rollup_retry Table

Records retry operations that are to be performed to rollup observations.

Table 67. persistence_rollup_retry Table

Fields	
prr_id	The primary key
prr_handler	The handler that will perform the operation
prr_start_time	The start time of the rollup
prr_end_time	The end time of the rollup
prr_partition	The partition being rolled up
prr_number_attempts	The number of attempts that have been performed
prr_batch_details	The details of the rollup operation

persistence_storage_config_xml Table

Stores the observation storage configuration.

Table 68. persistence_storage_config_xml Table

Fields	
pscx_id	The configuration ID
pscx_version	The configuration version
pscx_xml	The configuration

persistence_storage_manager Table

Records the details of the master storage manager operating in the cluster.

Table 69. persistence_storage_manager Table

Fields	
psm_id	The primary key
psm_last_updated	The last heartbeat timestamp
psm_primary_node	The name of the current master node in the cluster

persistence_timeslice_table Table

Records the details of the observation tables used by the server.

Table 70. persistence_timeslice_table Table

Fields	
ptt_id	The ID
ptt_partition	The partition to which the table maps
ptt_storage_type	The storage type
ptt_generation	The generation in which the table resides
ptt_start_time	The start time of the table's timeslice
ptt_end_time	The end time of the table's timeslice
ptt_table_name	The table name
ptt_table_suffix	The table name suffix (e.g. "0001")

persistence_topobj_purge_age Table

Records the purge age of various topology objects.

Table 71. persistence_topobj_purge_age Table

Fields	
ptpa_object_id	The object ID
ptpa_purge_age	The purge age

persistence_type_hierarchy Table

A "temporary" table that records the details of the topology type hierarchy.

Table 72. persistence_type_hierarchy Table

Fields	
pth_id	The primary key
pth_type_id	The type ID
pth_super_type	The ID of the type's supertype

registry_performance_calendar Table

Record performance calendars defined in registry values.

Table 73. registry_performance_calendar Table

Fields	
performance_calendar_id	Unique ID of the performance calendar
value	Value of the performance calendar
schedule	Schedule used for the performance calendar
value_id	ID of registry value that this performance calendar belongs to
performance_calendar_index	Index of this performance calendar in the list of performance calendars contained by the registry value

registry_registry_value Table

Records registry values defined in registry variables.

Table 74. registry_registry_value Table

Fields	
value_id	Unique ID of the registry value
value_type	Type of the registry value
user_created	If the registry value is created by user
default_value	Value of the registry value
topology_object_id	The unique ID of the topology object that the value is bond to
topology_type	The topology type that the value is bond to
registry_variable_version_id	The version ID of the registry variable that the value is for

registry_registry_variable Table

Records registry variables.

Table 75. registry_registry_variable Table

Fields	
version_id	Unique version ID of the registry variable
id	The ID of the registry variable
created	The time the registry variable is created
user_created	If the registry variable is created/modified by user
cartridge_name	The name of the cartridge that the registry variable comes from
cartridge_version	The version of the cartridge that the registry variable comes from

Table 75. registry_registry_variable Table

Fields	
domain_name	The name of the domain the registry variable belongs to
name	Name of the registry variable
class_name	The Java® class name for the value of the registry variable
comments	Description of the registry variable

report_output Table

Table to store the output of reports.

Table 76. report_output Table

Fields	
ro_id	The primary key
ro_report_schedule_id	The schedule report ID
ro_date_run	The timestamp that the report was run on
ro_error_message	Any error message that was encountered when running the report
ro_size	The report size
ro_data	The report file
ro_report_type	The report type
ro_report_id	The report ID
ro_user_name	The user that scheduled the report
ro_report_format	The report format
ro_email_recipients	The email addresses of recipients
ro_report_name	The report name
ro_man_gen	A flag indicating whether the report was manually generated
ro_report_params	The report parameters

report_schedule Table

Table to store the details of a scheduled report.

Table 77. report_schedule Table

Fields	
rs_version_id	The version ID
rs_id	The ID of the scheduled report
rs_name	The name of the scheduled report
rs_schedule_id	The ID of the schedule used to run the report

Table 77. report_schedule Table

Fields	
rs_num_results_retained	The number of report results that are to be retained
rs_is_disabled	A flag indicating whether the report has been disabled
rs_created_timestamp	The timestamp on which the report configuration was created
rs_is_user_created	Flag indicating whether the scheduled report has been user edited
rs_cartridge_name	The cartridge name
rs_cartridge_version	The cartridge version
rs_report_type	The report type
rs_report_id	The report ID
rs_user_name	The name of the user that created the schedule report
rs_report_format	The report format
rs_email_recipients	The email addresses of the report's recipients
rs_report_name	The name of the report
rs_report_params	The report parameters

rule_action_handler Table

Records actions that are associated with rules' conditions.

Table 78. rule_action_handler Table

Fields	
id	A unique identifier for each action
action	The name of the action that will be executed
description	A description of the action

rule_action_message Table

Records the messages that are associated with rules' actions.

Table 79. rule_action_message Table

Fields	
handler_id	The identifier of the action handler from rule_action_handler this message is bound to
message_name	The display name of this message
message_text	The text contents of this message

rule_action_registry_reference Table

Records a reference to a registry variable that will be used by the action handler.

Table 80. rule_action_registry_reference Table

Fields	
handler_id	The identifier of the action handler from rule_action_handler this reference is bound to
name	The display name of this reference
registry_variable_name	The name of the registry variable referred to

rule_action_variable_reference Table

Records a variable that can be used by a particular rule action.

Table 81. rule_action_variable_reference Table

Fields	
handler_id	The identifier of the action handler from rule_action_handler this reference is bound to
name	The display name of this variable
variable_name	The name of the rule expression this variable is bound to

rule_blackout_schedules Table

A table used to tie together rules with their blackout schedules.

Table 82. rule_blackout_schedules Table

Fields	
rule_version_id	The version-specific identifier of the rule included in this link
schedule_id	The identifier of a blackout schedule that should be applied to the provided rule

rule_effective_schedules Table

A table used to tie together rules with their effective schedules.

Table 83. rule_effective_schedules Table

Fields	
rule_version_id	The version-specific identifier of the rule included in this link
schedule_id	The identifier of a schedule controlling when this rule should be active

rule_expression Table

A table containing expressions that can be referred to by other parts of the rule.

Table 84. rule_expression Table

Fields	
rule_version_id	The version-specific identifier of the rule included in this link
name	The display name of this expression
text	The text contents of this expression

rule_firing_strategy Table

Contains data that describes how rules should fire.

Table 85. rule_firing_strategy Table

Fields	
id	An identifier that can be used to look up this particular firing strategy
rule_version_id	The version-specific identifier of the rule included in this link
type	The type of firing strategy represented by this row
delay_period	The amount of time that should be waited before the rule begins to fire
monitor_window_size	The number of hits that should be considered with a hit rate firing strategy
monitor_hit_threshold	The number of hits that must be detected before the rule's condition evaluates to true

rule_messages Table

Contains data that will be used in alarms generated by the rule.

Table 86. rule_messages Table

Fields	
rule_version_id	The version-specific identifier of the rule included in this link
name	The display name of this message
text	The text contents of this message

rule_rule Table

The main entry point for rules. Each row represents a particular instance of a rule. Changes to a rule result in a new entry being written so that historical values can be located.

Table 87. rule_rule Table

Fields	
rule_version_id	The version-specific identifier of this rule
type	The type of rule represented by this row
id	A global identifier used to refer to this rule (does not change with each version)
created	A timestamp indicating when this rule was created
name	The display name of this rule
comments	A description of what this rule does
is_external	Indicates whether this rule came from an external source or not
is_user_created	Indicates whether or not this rule was created or modified by the user
trigger_without_data	Indicates whether or not the rule's conditions should be tested even if new data has not arrives
trigger_type	The type of triggering strategy used by this rule
query_text	A topology query describing which objects this rule should monitor (for data-driven rules)
cartridge_name	The name of the cartridge this rule was shipped in (can be null)
cartridge_version	The version of the cartridge this rule was shipped in
domain_name	The name of the domain this containing objects this rule should be interested in
actions_suspended_till	A time (in the future) that this rule's actions will be suspended until (null if actions are not suspended)
alarms_suspended_till	A time (in the future) that this rule's alarms will be suspended until (null if alarm generation is not suspended)
help	Help text describing how this rule should be used
triggering_event	The name of the event (for event-driven rules) that will cause this rule's conditions to be evaluated
triggering_schedule	The identifier of the schedule (for schedule-driven rules) that will cause this rule's conditions to be evaluated
is_suspended	Indicates whether or not this rule is suspended (suspended rules are never evaluated)
firing_interval	How frequently this rule should fire (for time-driven rules)

rule_sev_to_clear_actn_hndlr Table

A table used to link up a rule's severities with the actions that should be executed when the severity's condition no longer holds.

Table 88. rule_sev_to_clear_actn_hndlr Table

Fields	
severity_id	The identifier of the severity linked to the clear action handler
handler_id	The identifier of the action handler that should run when the linked severity is no longer true

rule_sev_to_fire_actn_hndlr Table

A table used to link up a rule's severities with the actions that should execute when the severity becomes true.

Table 89. rule_sev_to_fire_actn_hndlr Table

Fields	
severity_id	The identifier of the severity linked to a fire action handler
handler_id	The identifier of the action handler that should run when the severity's condition becomes true

rule_severity Table

A table containing the different severity levels possible in rules.

Table 90. rule_severity Table

Fields	
id	The identifier of this severity
type	The type of severity (simple or conditional)
severity_level	The level of this severity: undefined (-1), normal (0), fire (1), warning (2), critical (3), or fatal (4)
is_disabled	Bit indicating whether or not this severity has been disabled
message_name	The display name of the message associated with this severity
message_text	The text of this severity's message
condition_name	The display name of this rule's condition
condition_text	The Groovy script that will be executed when evaluating this condition
rule_version_id	The version-specific identifier of this rule

rule_severity_expression Table

An expression linked to a severity that can be referred to by the rule's action handlers.

Table 91. rule_severity_expression Table

Fields	
severity_id	The identifier of the severity to which this expression is bound
name	The display name of this expression
text	The contents of this expression

rule_severity_messages Table

An message linked to a severity that can be referred to by the rule's action handlers.

Table 92. rule_severity_messages Table

Fields	
severity_id	The identifier of the severity to which this message is bound
name	The display name of this message
text	The contents of this message

schedule_named_schedule Table

Records schedules.

Table 93. schedule_named_schedule Table

Fields	
version_id	Unique version ID of the schedule
id	ID of the schedule
created	Creation time of the schedule
user_created	If the schedule is created/modified by user
cartridge_name	The name of the cartridge that the schedule comes from
cartridge_version	The version of the cartridge that the schedule comes from
name	Name of the schedule
description	Description of the schedule
auto_removed	If the schedule should be removed automatically when it expires
schedule	The XML definition of the schedule

script_annt Table

Holds annotations on deployed scripts.

Table 94. script_annt Table

Fields	
annt_id	Annotation ID
name	Annotation name
value	Annotation value
script_id	UUID of the annotated script

script_annt_attr Table

Holds attributes of script annotations.

Table 95. script_annt_attr Table

Fields	
annt_id	Annotation ID
name	Annotation attribute name
value	Attribute value

script_argument Table

Holds script argument declarations.

Table 96. script_argument Table

Fields	
arg_id	Argument ID
name	Argument name
type	Argument type
description	Argument description
optional	Whether the argument is optional (1) or not (0)
metric	Not used
script_id	Associated script ID
position	Argument's position

script_argument_annt Table

Holds script argument annotations.

Table 97. script_argument_annt Table

Fields	
annt_id	Annotation ID
name	Annotation name
value	Annotation value
arg_id	Argument ID

script_argument_annt_attr Table

Holds attributes of script argument annotations.

Table 98. script_argument_annt_attr Table

Fields	
annt_id	Annotation ID
name	Annotation attribute name
value	Attribute value
annt_id	Annotation ID

script_example Table

Holds script usage examples.

Table 99. script_example Table

Fields	
script_id	Script ID
example_usage	Script usage example (plain text)
description	Script description

script_return_annt Table

Holds script return value annotations.

Table 100. script_return_annt Table

Fields	
annt_id	Annotation ID
name	Annotation name
value	Annotation value
script_id	Script ID

script_return_annt_attr Table

Holds attributes of script return value annotations.

Table 101. script_return_annt_attr Table

Fields	
annt_id	Annotation ID
name	Annotation attribute name
value	Attribute value

sec_group Table

Holds information about security group.

Table 102. sec_group Table

Fields	
groupuuid	The primary key
groupname	Group name
groupdesc	Group description
jasssourceid	JAAS source ID
isbuiltingroup	A flag indicating whether group is built in
ishidden	A flag indicating whether group is hidden

sec_group_nesting Table

Holds information about security group nesting relation.

Table 103. sec_group_nesting Table

Fields	
groupuuid	Parent group UUID
childgroupuuid	Child group UUID

sec_group_role_match Table

The table was used to create a relation map of external group name and internal Foglight role name, and is no longer in use.

sec_grouprole Table

Security group and security role many-to-many association table.

Table 104. sec_grouprole Table

Fields	
groupuuid	Security group UUID
roleuuid	Security role UUID

sec_jaas_source Table

Holds information about JAAS source.

Table 105. sec_jaas_source Table

Fields	
jaassourceuuid	JAAS source ID
jaassourcename	JAAS source name
jaasauthtype	JAAS source authentication type
jaassourcedesc	JAAS source description
isservicesource	A flag indicating whether it is service source
isdefaultextsource	A flag indicating whether it is default external source

sec_object Table

Holds information about security object.

Table 106. sec_object Table

Fields	
objuuid	Security object UUID
objname	Security object name
objdesc	Security object description
createdby	The user created this security object
modifiedby	The user modified this security object recently
deletedby	The user deleted this security object
objtypeuuid	Security object type UUID
createdts	Security object create time stamp
modifiedts	Security object modify time stamp
deletedts	Security object delete time stamp
owneruuid	Owner UUID
groupuuid	Group UUID
ownerperm	Owner permission
groupperm	Group permission

Table 106. sec_object Table

Fields	
otherperm	Other permission
isdeleted	A flag indicating whether the security object is deleted
useruuid	User UUID

sec_object_mask Table

Holds information about security object mask.

Table 107. sec_object_mask Table

Fields	
groupuuid	Group UUID
objtypeuuid	Object type UUID
ownerperm	Owner permission
groupperm	Group permission
otherperm	Other permission

sec_object_permission Table

Holds information about security object permission.

Table 108. sec_object_permission Table

Fields	
roleuuid	Security role UUID
objuuid	Security object UUID
permdesc	Permission description
accessperm	Access permission
accesstype	Access type

sec_object_type Table

Holds information about security object type.

Table 109. sec_object_type Table

Fields	
objtypeuuid	Security object type UUID
objtypename	Security object type name
objtypedesc	Security object type description
objclassname	Security object class name

Table 109. sec_object_type Table

Fields	
objfactoryclassname	Security object factory class name
objcreationmethodname	Security object creation method name

sec_permission Table

Holds information about security permission.

Table 110. sec_permission Table

Fields	
roleuuid	Security role UUID
resuuid	Security resource UUID
permdesc	Permission description
accessperm	Access permission
accesstype	Access type

sec_permission_def Table

Holds information about spring security ACL permission.

Table 111. sec_permission_def Table

Fields	
id	The primary key
name	Permission name
displayName	Permission display name
description	Permission description
domain	Permission domain (for example, topology domain name)
cartridgeName	The name of cartridge where permission is defined
cartridgeVersion	The name of cartridge where permission is defined
defaultValue	Default value
hidden	A flag indicating whether the permission is hidden

sec_policy Table

Holds information about security policy.

Table 112. sec_policy Table

Fields	
policykey	Security policy key
policyvalue	Security policy value

sec_resource Table

Holds information about security resource.

Table 113. sec_resource Table

Fields	
resuuid	Security resource UUID
resname	Security resource name
resdesc	Security resource description
restypeid	Security resource type

sec_role Table

Holds information about security role.

Table 114. sec_role Table

Fields	
roleuuid	The primary key
rolename	Security role name
roledesc	Security role description
isbuiltinrole	A flag indicating whether the security role is built in

sec_user_alias Table

Holds information about security user alias information.

Table 115. sec_user_alias Table

Fields	
useralias	User alias
username	User name
isdeleted	A flag indicating whether user is deleted

sec_user_obj_permission Table

Holds information about security user object permission information.

Table 116. sec_user_obj_permission Table

Fields	
useruuid	Security user UUID
objuuid	Security object UUID
permdesc	Permission description
accessperm	Access permission
accesstype	Access type

sec_user_res_permission Table

Holds information about security user resource permission information.

Table 117. sec_user_res_permission Table

Fields	
useruuid	Security user UUID
resuuid	Security resource UUID
permdesc	Permission description
accessperm	Access permission
accesstype	Access type

sec_usergroup Table

Security user and security group many-to-many association table.

Table 118. sec_usergroup Table

Fields	
groupuuid	Security group UUID
useruuid	Security user UUID

sec_userrole Table

Security user and security role many-to-many association table.

Table 119. sec_userrole Table

Fields	
roleuuid	Role UUID
useruuid	User UUID

sec_x_attribute Table

The table is related to security attribute, and is no longer in use.

sec_x_attribute_value Table

The table is related to security attribute value, and is no longer in use.

tagging_service_mapping Table

Holds various object tags.

Table 120. tagging_service_mapping Table

Fields	
mobjectid	Object ID
mtype	Object type
mtag	Tag

threshold_bound Table

A table containing the boundary conditions used in thresholds.

Table 121. threshold_bound Table

Fields	
tb_id	A unique identifier for each threshold bound instance
tb_type	Describes the type of threshold bound contained in this row (affects which columns are expected to be present)
tb_state	The enumeration object instance used to represent this threshold's state
tb_is_inclusive	A flag indicating whether the boundary value is included in the threshold itself
tb_is_override	A flag indicating whether this threshold should override any previously exceeded thresholds
tb_constant_value	The value incoming metrics will be compared against in constant thresholds
tb_registry_variable	The name of the registry variable that incoming metrics will be compared against (for registry threshold bounds)
tb_metric	The name of the metric that will be compared against this threshold boundary
tb_num_std_deviations	The number of standard deviations the incoming metric must remain within before this boundary is exceeded (for standard deviation boundaries)

Table 121. threshold_bound Table

Fields	
tb_baseline_config_id	The identifier of the baseline configuration incoming metrics will be compared to
tb_bound_id	The identifier of this bound
tb_threshold_id	The identifier of the threshold this bound belongs to
tb_index	An index used to sort threshold boundaries during comparisons

threshold_config Table

A table containing threshold configurations (each of which contains zero or more bounds).

Table 122. threshold_config Table

Fields	
tc_version_id	The version-specific identifier of this threshold configuration, which changes each time the threshold is saved
tc_id	The version-agnostic identifier of this threshold configuration
tc_topology_type	The name of the topology type this threshold will apply to
tc_observation_property	The name of the property this threshold represents
tc_enum_type	The type of enumeration that describes this threshold's state
tc_created_timestamp	A timestamp representing when this threshold was created
tc_is_user_created	A flag indicating whether this threshold was created by the user or shipped in a cartridge
tc_cartridge_name	The name of the cartridge containing this threshold
tc_cartridge_version	The version of the cartridge this threshold configuration was shipped in

topology_activity_calendar Table

Records the activity calendar used to track the activity of topology objects.

Table 123. topology_activity_calendar Table

Fields	
tac_id	The calendar ID
tac_start_time	The start time
tac_end_time	The end time
tac_activity_current_day	Bitmask for the current day
tac_activity_current_day_excl	Exclusion bitmask
tac_activity_prev_day_0	Bitmask for previous day
tac_activity_prev_day_0_excl	Exclusion bitmask

Table 123. topology_activity_calendar Table

Fields	
tac_activity_prev_day_1	Bitmask for previous day
tac_activity_prev_day_1_excl	Exclusion bitmask
tac_activity_daily	Bitmask for daily activity
tac_activity_daily_excl	Exclusion bitmask
tac_activity_weekly	Bitmask for weekly activity
tac_activity_weekly_excl	Exclusion bitmask

topology_activity_upgrade Table

Table to store activity details of an object being upgraded.

Table 124. topology_activity_upgrade Table

Fields	
tau_object_id	The object ID
tau_activity_current_day	Activity for current day
tau_activity_prev_day_0	Activity for a previous day
tau_activity_prev_day_1	Activity for a previous day
tau_activity_daily	Daily activity
tau_activity_weekly	Weekly activity

topology_object Table

Stores the details of topology objects.

Table 125. topology_object Table

Fields	
to_id	The object ID
to_type	The object type
to_unique_id	The unique ID
to_object_name	The object's name
to_object_name_hash	A hash of the object's name
to_last_updated	Last updated timestamp
to_activity_day_number	The day relative to which activity details are stored
to_activity_current_day	Activity bitmask for the current day
to_activity_prev_day_0	Activity bitmask for the previous day
to_activity_prev_day_1	Activity bitmask for the calendar day two days ago

Table 125. topology_object Table

Fields	
to_activity_daily	Daily activity bitmask
to_activity_weekly	Weekly activity bitmask

topology_object_history Table

Stores the history information for topology objects.

Table 126. topology_object_history Table

Fields	
toh_id	The history record ID
toh_type	The object's type
toh_object_id	The object's ID
toh_unique_id	The object's unique ID
toh_version	The version
toh_effective_start_date	The date at which the version came into effect
toh_effective_end_date	The date at which the version is no longer effective
toh_num_changes	The number of changes that were made to the object in this version

topology_property Table

Stores the details of a property defined on a topology type.

Table 127. topology_property Table

Fields	
tp_id	The property ID
tp_containing_type	The ID of the containing type
tp_name_id	The property name ID
tp_name	The property name

topology_property_annotation Table

Stores the details of annotations defined on topology properties.

Table 128. topology_property_annotation Table

Fields	
tpa_id	The primary key
tpa_type	The type

Table 128. topology_property_annotation Table

Fields	
tpa_property	The property ID
tpa_annotation	The annotation type
tpa_index	The value index
tpa_value	The value

topology_property_history Table

Stores the details of changes made to a topology property.

Table 129. topology_property_history Table

Fields	
tph_id	The primary key
tph_containing_type	The ID of the containing type
tph_property	The property ID
tph_type	The property type
tph_description	The description
tph_is_identity	Flag indicating whether the property is an identity property
tph_is_containment	Flag indicating whether the property is a containment property
tph_is_many	Flag indicating whether the property is a list property
tph_default_value	The default value
tph_unit_name	The units for the property

topology_property_name Table

Table that stores unique property names.

Table 130. topology_property_name Table

Fields	
tpn_id	The property name ID
tpn_name	The property name

topology_property_value Table

Table that stores the property values of topology objects.

Table 131. topology_property_value Table

Fields	
pv_id	The primary key
pv_object	The object ID
pv_property	The property
pv_effective_start_date	The timestamp at which the property value came into effect
pv_effective_end_date	The timestamp at which the property value was no longer effective
pv_index	The property value index
pv_value	The property value
pv_reference_id	The reference ID for the value

topology_service_state Table

Stores the runtime state information for the topology service.

Table 132. topology_service_state Table

Fields	
tss_service	The service name
tss_state	The state

topology_type Table

Stores the details of topology types.

Table 133. topology_type Table

Fields	
tt_id	The type ID
tt_name	The type name

topology_type_annotation Table

Stores the details of annotations attached to topology types.

Table 134. topology_type_annotation Table

Fields	
tta_id	The primary key
tta_type	The type ID
tta_annotation	The annotation type ID

Table 134. topology_type_annotation Table

Fields	
tta_index	The value index
tta_value	The value

topology_type_history Table

Stores the history of changes made to topology types.

Table 135. topology_type_history Table

Fields	
tth_id	The history record ID
tth_type_id	The type ID
tth_name	The type name
tth_class_name	The implementation class name
tth_version	The type version
tth_effective_start_date	The history record's start date
tth_effective_end_date	The history record's end date
tth_cartridge_name	The name of the associated cartridge
tth_cartridge_version	The cartridge version
tth_domain_name	The domain to which the type belongs
tth_super_type	The ID of the super type

upgrade_pending_operations Table

Table used for upgrades to keep track of operations that need to be performed on the server.

Table 136. upgrade_pending_operations Table

Fields	
upo_name	The operation name
upo_service_name	The service that will perform this operation
upo_trigger_event	The event that will trigger the operation to run
upo_operation_class	The name of the Java® class that implements the operation

wcf_groups_by_cartridges Table

Records the association between WCF modules and cartridges that contain them.

Table 137. wcf_groups_by_cartridges Table

Fields	
wcf_res_group	Qualified name of the WCF module
cartridge_id	Identifier of the cartridge

wcf_resources Table

Records WCF resources.

Table 138. wcf_resources Table

Fields	
wcf_res_id	Unique ID of the WCF resource
wcf_res_group	Qualified name of the WCF module
wcf_res_name	Name of the WCF resource
wcf_res_data	Binary data of the WCF resource

About Us

Quest creates software solutions that make the benefits of new technology real in an increasingly complex IT landscape. From database and systems management, to Active Directory and Office 365 management, and cyber security resilience, Quest helps customers solve their next IT challenge now. Around the globe, more than 130,000 companies and 95% of the Fortune 500 count on Quest to deliver proactive management and monitoring for the next enterprise initiative, find the next solution for complex Microsoft challenges and stay ahead of the next threat. Quest Software. Where next meets now. For more information, visit https://www.quest.com/.

Technical support resources

Technical support is available to Quest customers with a valid maintenance contract and customers who have trial versions. You can access the Quest Support Portal at https://support.quest.com.

The Support Portal provides self-help tools you can use to solve problems quickly and independently, 24 hours a day, 365 days a year. The Support Portal enables you to:

- Submit and manage a Service Request.
- View Knowledge Base articles.
- Sign up for product notifications.
- Download software and technical documentation.
- View how-to-videos.
- Engage in community discussions.
- Chat with support engineers online.
- View services to assist you with your product.