CA Technologies Product References

This guide references CA Spectrum®.
This Documentation, which includes embedded help systems and electronically distributed materials, (hereinafter referred to as the “Documentation”) is for your informational purposes only and is subject to change or withdrawal by CA at any time.

This Documentation may not be copied, transferred, reproduced, disclosed, modified or duplicated, in whole or in part, without the prior written consent of CA. This Documentation is confidential and proprietary information of CA and may not be disclosed by you or used for any purpose other than as may be permitted in (i) a separate agreement between you and CA governing your use of the CA software to which the Documentation relates; or (ii) a separate confidentiality agreement between you and CA.

Notwithstanding the foregoing, if you are a licensed user of the software product(s) addressed in the Documentation, you may print or otherwise make available a reasonable number of copies of the Documentation for internal use by you and your employees in connection with that software, provided that all CA copyright notices and legends are affixed to each reproduced copy.

The right to print or otherwise make available copies of the Documentation is limited to the period during which the applicable license for such software remains in full force and effect. Should the license terminate for any reason, it is your responsibility to certify in writing to CA that all copies and partial copies of the Documentation have been returned to CA or destroyed.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CA PROVIDES THIS DOCUMENTATION “AS IS” WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT. IN NO EVENT WILL CA BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, LOST INVESTMENT, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF CA IS EXPRESSLY ADVISED IN ADVANCE OF THE POSSIBILITY OF SUCH LOSS OR DAMAGE.

The use of any software product referenced in the Documentation is governed by the applicable license agreement and such license agreement is not modified in any way by the terms of this notice.

The manufacturer of this Documentation is CA.

Provided with “Restricted Rights.” Use, duplication or disclosure by the United States Government is subject to the restrictions set forth in FAR Sections 12.212, 52.227-14, and 52.227-19(c)(1) - (2) and DFARS Section 252.227-7014(b)(3), as applicable, or their successors.

Copyright © 2014 CA. All rights reserved. All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.
Contact CA Technologies

Contact CA Support

For your convenience, CA Technologies provides one site where you can access the information that you need for your Home Office, Small Business, and Enterprise CA Technologies products. At http://ca.com/support, you can access the following resources:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

Providing Feedback About Product Documentation

If you have comments or questions about CA Technologies product documentation, you can send a message to techpubs@ca.com.

To provide feedback about CA Technologies product documentation, complete our short customer survey which is available on the CA Support website at http://ca.com/docs.
Contents

Chapter 1: CA Spectrum Modeling Concepts 9
Introduction .......................................................................................................................... 9
CA Spectrum and Model Type Editor ................................................................................. 9
Model Types and Attributes ............................................................................................. 10
Models ............................................................................................................................... 14
Relations and Meta-Rules ................................................................................................. 15
Model Type Inheritance ..................................................................................................... 17
  Attribute Descriptors ....................................................................................................... 17
  Standard Hierarchy .......................................................................................................... 19
  Specialized Hierarchy ...................................................................................................... 20
  Model Type Precedence ................................................................................................. 22
  Attribute Collapsing ...................................................................................................... 25

Chapter 2: Getting Started with the Model Type Editor 27
Using a Developer ID .......................................................................................................... 27
  Access Privileges With Developer ID ............................................................................. 28
SpectroSERVER Database Protection ............................................................................... 29
Considerations on Database Migration ............................................................................. 30
About Starting the Model Type Editor ............................................................................... 31
  Start Model Type Editor from the Control Panel ......................................................... 31
  Start Model Type Editor from the Command Line ....................................................... 32
Overview of the User Interface ......................................................................................... 33
Commit Changes to the SpectroSERVER Database ......................................................... 34
Sorting and Filtering Lists ............................................................................................... 35
Add and Remove Columns from Tables ......................................................................... 36
Exit the Model Type Editor ............................................................................................. 36

Chapter 3: Creating and Modifying Model Types 37
Introduction ....................................................................................................................... 37
Attributes of Model Types ............................................................................................... 37
  Developer ID .................................................................................................................. 38
  Model Type Name ......................................................................................................... 38
  Model Type ID (Handle) ............................................................................................... 39
  Base Model Types ......................................................................................................... 39
  Derived Model Types ................................................................................................. 39
  Model Type Flags ......................................................................................................... 40
Custom Attributes ........................................................................................................... 42
Standard Attribute Descriptors ..................................................................................... 43
  Developer ID .............................................................................................................. 43
  Attribute Name ......................................................................................................... 43
  Attribute ID (Handle) ................................................................................................. 44
  Type ......................................................................................................................... 44
  Flags .......................................................................................................................... 45
  Group Name and Group ID ....................................................................................... 47
Special Attribute Descriptors .......................................................................................... 47
  Default Value ............................................................................................................. 48
  Extended Flags ......................................................................................................... 48
  OID Prefix and OID Reference ................................................................................. 50
  Polling Group ............................................................................................................ 51
Search for and Display Model Types .............................................................................. 52
Search for and Display Attributes.................................................................................. 53
Create a Model Type ...................................................................................................... 53
Delete a Model Type ...................................................................................................... 54
Working with Base Model Types .................................................................................... 56
  How to Determine the Base Model Types for a New Model Type ......................... 56
  Add a Base Model Type to a Model Type ................................................................. 58
  Remove a Base Model Type from a Model Type ...................................................... 58
Import MIBs ................................................................................................................... 60
Set Model Type Flags ................................................................................................... 62
Working with Attributes .............................................................................................. 63
  Add an Attribute to a Model Type ........................................................................... 63
  Remove an Attribute from a Model Type ................................................................. 65
  Edit an Attribute ...................................................................................................... 66
Working with Attribute Groups ..................................................................................... 69
  Create an Attribute Group ....................................................................................... 69
  Modify an Attribute Group ....................................................................................... 70
  Delete Attribute Group ............................................................................................ 71

Chapter 4: Working with Relations and Meta-Rules ..................................................... 73
About Working with Relations and Meta-Rules ............................................................ 73
Search for and Display Relations .................................................................................. 74
  Create Relations ..................................................................................................... 76
  Relation Meta—Rules ............................................................................................... 77
  Delete Relations ...................................................................................................... 78
  Delete Meta—Rules ................................................................................................. 79
Chapter 5: Importing and Exporting Model Types ................................................................. 81
About Importing and Exporting Model Types .................................................................... 81
Import Model Types Using the Model Type Editor ................................................................ 81
  Import Constraints .............................................................................................................. 82
Import Model Types ............................................................................................................. 82
Export Model Types Using the Model Type Editor ............................................................... 83
Import and Export Model Types Using dbtool ..................................................................... 85
Send an Exported Catalog to a File or Printer ..................................................................... 86

Chapter 6: Running Reports on Model Types and Relations ............................................... 87
About Running Reports on Model Types and Relations ......................................................... 87

Index ..................................................................................................................................... 89
Chapter 1: CA Spectrum Modeling Concepts

This section contains the following topics:

- Introduction (see page 9)
- CA Spectrum and Model Type Editor (see page 9)
- Model Types and Attributes (see page 10)
- Models (see page 14)
- Relations and Meta-Rules (see page 15)
- Model Type Inheritance (see page 17)

Introduction

This chapter introduces the Model Type Editor application and the database objects that you can create using it, namely, model types, attributes, relations, and meta-rules. The chapter also discusses model type inheritance, which is an important concept to understand before creating and modifying model types.

Important! Before using the Model Type Editor, read the Certification User Guide. The guide provides information on the GnSNMPDev management module. You can use GnSNMPDev to represent a SNMP-compliant network device that lacks a corresponding CA Spectrum management module. You can also use GnSNMPDev as a toolkit to create new management modules that include new, supporting device model types and application model types.

CA Spectrum and Model Type Editor

CA Spectrum is an integrated management system that runs on the Solaris, Linux, and Windows platforms. The CA Spectrum design is based on a client/server model. The server, called the SpectroSERVER, includes the CA Spectrum knowledge base, which gets and stores all network information. The client is the CA Spectrum user interface, called OneClick, which provides a graphical representation of the network environment.

The SpectroSERVER provides intelligence and contains models of the actual network devices and their relationships—models that continuously collect data about the entities they represent and collectively provide a comprehensive management perspective of the network. Through polling, the CA Spectrum database gains extensive information about network devices, their relationships, and their performance.
The OneClick client application provides a multi-dimensional picture of the SpectroSERVER database. Users can retrieve and view the information maintained in the SpectroSERVER database, and they can invoke the SpectroSERVER to control objects on the network. Network information can be presented from various perspectives, including topological or geographical views.

CA Spectrum deploys model types, models, attributes, relations, and meta-rules to monitor network infrastructure. Model types are the templates used to create models, and models are specific instances of a model type. (The terms "model type" and "template" are used interchangeably in CA Spectrum documentation.) Attributes are the characteristics of a specific model type, and relations and meta-rules define how model types interact with each other. While models and associations are defined using OneClick, the Model Type Editor lets you define model types, rules, relations, and attributes.

The Model Type Editor is the Java-based application that you use to create, modify, and delete modeling catalog objects. Because it is a thick-client application that accesses the SpectroSERVER database, you must run it from the SpectroSERVER platform.

**Model Types and Attributes**

A model type is a template that is used to create models, and it is defined by a specific set of attributes. In turn, attributes are the database variables that collectively characterize the real-world object represented by the model type. Model types range from simple with few attributes and relationships to very complex with many attributes and relationships.
Complex model types are often derived by inheriting attributes from several, simpler model types. The resulting combination constitutes a hierarchy of model types. Parent model types (model types from which one or more other model types have been derived) are called *base model types*. Child model types (model types that have been derived from one or more other types) are called *derived model types*. The following illustration shows a sample model type hierarchy that illustrates model type derivation.

In the illustration, Device is a base model type for three model types: PC Card, Hub, and Bridge. As such, all three derived model types inherit the attributes of Device.

In turn, PC Card, Hub, and Bridge are each used as a base model type for another model type, respectively, SNMP PC Card, SNMP MMAC, and SNMP Bridge. All of these derived model types inherit the attributes of their immediate parent model type, and by extension, the attributes of the Device model type. In addition, they also inherit the attributes of a model type named SNMP Protocol.
In the Model Type Editor, you can identify a model type's position in the model type hierarchy using the Hierarchy tab. The tab shows the following for the current model type:

- The base (parent) model types from which the current model type is derived
- The derived (child) model types that are derived from the current model type
The Model Type Editor also provides an Attribute tab that shows the attributes of a model type and their default values.

There are various rationales for attributes. Certain attributes, such as the name of a router or its IP address, are necessary in order to uniquely identify a resulting model within the CA Spectrum database. Other attributes relate a model to the SNMP object identifiers (OIDs) supported by the model. Still others support CA Spectrum functionality. For example, Discovery_Precedence is used by the Discovery program; Value_When_Red is used for alarm roll-up, and Polling_Interval is used to determine how often CA Spectrum polls the given device for information.
Models

A model is an instantiation of a model type by the SpectroSERVER. All models instantiated from the same model type have the same set of attributes and CA Spectrum intelligence. However, the values for the attributes are unique to each model except in the case of Shared attributes (that is, attributes for which the Shared flag is set).

As a simple example, the following illustration shows three models derived from a model type named Building, which has three attributes: Modeltype_Name, Model_Name, and Floors. Because each model is instantiated from the same model type, each model has the same value for Modeltype_Name (a Shared attribute). However, the values for Model_Name and Floors vary.

Note: A network administrator can add, edit, and delete models using OneClick. For more information, see the Modeling and Managing Your IT Infrastructure Administrator Guide.
Relations and Meta-Rules

A relation is a database construct that defines a specific type of relationship between model types. The following are examples of relations:

- Contains
- Collects
- Executes
- HASPART
- Is_Adjacent_to
- Monitors
- Pings_Through
- Schedules

Each relation is defined by one or, more typically, several rules called meta-rules. The *meta-rules* for a relation apply the relation to specific model types, thereby defining how the model types can interact with one another. As an example, the following are several of the meta-rules defined for the Contains relation:

- Building Contains Floor
- Building Contains Room
- Building Contains Section
- Country Contains Building
- Country Contains Region
- Country Contains Site

Each meta-rule must have exactly one antecedent model type (the left-hand entry) and exactly one predicate model type (the right-hand entry). Any model type can be an antecedent or a predicate for any number of meta-rules.

OneClick uses the meta-rules for model types to establish the rules for interaction between specific models instantiated from the model types. In CA Spectrum terms, instantiation of a meta-rule between two models produces an *association* between the models, and each model can react to the knowledge that it is associated with the other model. For example, consider the following meta-rule:

**Country Contains Building**

This might result in the following association between two specific models:

France contains Corporation001
There are two types of relations:

**One-To-Many**

A relation of this type defines how a single model of one model type relates to multiple models of another model type. For example, Contains is a one-to-many type of relation. As such, it can have one or more meta-rules that define one-to-many relationships between models. For example, it includes a "Country contains Building" meta-rule, which specifies that a single model of the Country model type can contain multiple models of the Building model type.

**Many-To-Many**

A relation of this type defines how multiple models of one model type relate to multiple models of another model type. For example, Is_Adjacent_to is a many-to-many type of relation. As such, it can have one or more meta-rules that define many-to-many relationships between models. For example, it includes a "Device Is_Adjacent_to Device" meta-rule, which specifies that multiple models of the Device model type can be adjacent to multiple models of the Device model type.

The Model Type Editor provides a Relation tab that shows the meta-rules defined for a specific relation.
Most relations in the modeling catalog that comes with the basic CA Spectrum package have meta-rules that specify the model types to which the relations can be applied.

Model Type Inheritance

When you derive new model types from existing model types, the specifics of model type and attribute inheritance are important to understand because the functionality of each model type depends on its inheritance.

This section provides information on important concepts related to inheritance, namely, attribute descriptors, standard versus specialized hierarchies, model type precedence, and attribute collapsing.

Attribute Descriptors

Attribute descriptors are the characteristics that define the attribute, for example, its type (boolean, integer, text string, and so on) and default value.

An attribute has two types of descriptors:

- **Standard descriptors**: The values of these attribute descriptors are inherited by all model types derived directly or indirectly from the base model type in which the attribute was first created (referred to as the originating model type).

  You can only modify the values of standard descriptors in the originating model type; you cannot modify their values in derived model types.

  Name, Attribute ID, and the Shared flag are three examples of standard descriptors.

- **Descriptors that can be specialized**: Like standard attribute descriptors, the values of these descriptors are inherited by all model types derived directly or indirectly from the base model type in which the attribute was first created. However, unlike standard descriptors, you can modify the values of these descriptors at any level in the model type hierarchy. This process is called specialization.

  When you modify a descriptor value in a derived model type, the new value overrides the inherited value for that derived model type and any of its own derived (child) model types. Other model types that are derived from the same base model type—and that are unspecialized with respect to the same attribute descriptor—continue to inherit their descriptor values; this is also the case for any of their derived (child) model types.
Important! The complete list of descriptors that you can specialize includes—notably, Default Value—as well as the Extended Flags (Memory, Database, Polled, and Logged), OID Prefix, OID Reference, and Polling Group, as identified in the following image.
Standard Hierarchy

The following illustration shows a model type hierarchy whose attributes and their default values are inherited through a standard hierarchical sequence.

In the illustration, Model Type A is derived from Base Model Type, and, as a result, Model Type A inherits attribute X and its default value of 1 from Base Model Type. In a similar manner, Model Type B is derived from Model Type A, and, as a result, Model Type B inherits attribute X and its default value of 1 from Model Type A.

This hierarchical relationship of inheritance is applied to all other model types derived from Model Type A, Model Type B, or any of their descendants. Moreover, the relationship is maintained by the database so that any change made to the value of the Default Value attribute descriptor in the originating model type is immediately applied to all derived model types that inherit the descriptor value.

As a simple example, assume you add a Technical-Assistance text string attribute to Base Model Type with a telephone number as the default value. Model Type A, Model Type B, and all other model types that derive from them would all inherit the Technical-Assistance attribute and, therefore, the value of the Default Value attribute descriptor (the telephone number). If you then changed the telephone number, all of the model types derived directly or indirectly from Base Model Type would immediately inherit the new telephone number.
Specialized Hierarchy

In contrast to the standard hierarchy, the following illustration shows a model type hierarchy where one derived model type inherits its default value but another does not.

In this illustration, the default value for attribute X (or, more specifically, the value of the Default Value attribute descriptor for attribute X) in Model Type A is specified, not inherited. In other words, the value inherited from Base Model Type is overridden and no longer used. As mentioned previously in this guide, this process is called **specialization**.

When you override a descriptor value that is inherited, you break the relationship to the base (parent) model type for that attribute descriptor only. The inheritance relationship with respect to the values of all other attribute descriptors is not affected. Furthermore, model types that are derived from specialized model types behave exactly as if they were derived from unspecialized model types. In the illustration, Model Type B inherits its default value for attribute X from Model Type A just as it would if Model Type A were not specialized. If you were to change the default value of attribute x in Model Type A, the default value of the same attribute in Model Type B would immediately change as well. Because the inheritance relationship is in effect, this would be the case even if you derived Model Type B before you made the change.
In reality, the attribute hierarchy often consists of a combination of inherited and specialized values, particularly when multiple model types are derived from a common base model type. Moreover, while specialization is most typical with respect to the Default Value attribute descriptor, it is possible for any other descriptor that can be specialized. The following diagram illustrates a model type hierarchy that uses both inheritance and specialization for an attribute (attribute X).

Note in the figure that Model Type C is specialized by introducing new attribute Y. Therefore, this attribute is inherited by derived Model Type D.
Model Type Precedence

The process of inheriting attributes from base model types means that a derived model type can inherit the same attribute from two or more inheritance paths that have a common originating model type. To avoid ambiguity in this situation, the base model types are ranked according to the order in which they are added as base model types for a derived model type, and this model type ranking determines the inheritance path to use when a derived model type can inherit an attribute from multiple paths. More specifically, the derived model type inherits the attribute from the base model type with the lowest ranking.

When you first derive a model type from a base model type, that base model type is given a ranking of 1. If you then add a second base model type to the derived model type, that second base model type is given a ranking of 2. Subsequent base model types are assigned rankings in a similar manner.

As an example, consider the following model type derivation workflow:

1. You derive model type A and model type B from a common base model type.

   ![Diagram](image)

   The base model type defines attribute X, which is inherited by both derived model types. The base model type is assigned a ranking of 1 with respect to both derived model types.
2. You derive model type C from model type A.

Model type C inherits attribute X from model type A, which, in turn, inherits the attribute from the base model type. Because model type C was created based on model type A, model type A is assigned a base model type ranking of 1 with respect to derived model type C.

3. You add model type B as a base model type of model type C.

Model type B is assigned a base model type ranking of 2 with respect to derived model type C. Because model type A has a lower ranking as a base model type, model type C inherits attribute X from model type A, not from model type B.
In the Model Type Editor, the base model types for a given model type are listed in ranked order, so you can identify the order of precedence for attribute inheritance.

**Contents:** GnSNMPDev

<table>
<thead>
<tr>
<th>Hierarchy</th>
<th>Attributes</th>
<th>Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model Type</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Developer ID:</td>
<td>Cron_SNMP</td>
<td></td>
</tr>
<tr>
<td>Model Type Name:</td>
<td>GnSNMPDev</td>
<td></td>
</tr>
<tr>
<td>Model Type ID:</td>
<td>0x003d0002</td>
<td></td>
</tr>
</tbody>
</table>

**Base Model Types**

- Listed according to model type ranking

**Derived Model Types**

- Listed alphabetically

**Note:** You can change a base model type's ranking by removing it and then re-adding it as a base model type in a specific location in the list.

**More information:**

[Add a Base Model Type to a Model Type](#) (see page 58)
Attribute Collapsing

To maintain an attribute hierarchy that is as simple as possible, the Model Type Editor includes a feature that eliminates unnecessary specialization. That is, if you modify the Default Value descriptor (or any other descriptor that can be specialized) in a derived model type so that its value matches the value in the corresponding descriptor in the base model type from which it immediately inherits the attribute (that is, from an immediate parent model type), the database discards the modification and instead inherits the descriptor value from the base model type. This process is called "attribute collapsing;" its goal is to simplify a specialized hierarchy as much as possible by removing unnecessary customizations.

Once the descriptor value reverts to the inherited value, if you subsequently change the descriptor value in the base model type, the derived model type also receives the change (which is normal inheritance behavior).

You can identify the model type from which the current model type obtains its default value in the Value Defined In field in the Component Detail panel, as shown in the following image.

Note: Attribute collapsing does not take place for default values if the attribute is Shared (that is, the Shared flag is set).
Chapter 2: Getting Started with the Model Type Editor

Before using the Model Type Editor, we recommend that you also read the Certification User Guide. The guide provides information about the GnSNMPDev management module provided with CA Spectrum. You can use GnSNMPDev to represent an SNMP-compliant network device that does not have a corresponding CA Spectrum management module. You can also customize GnSNMPDev to extend its support, or you can use it as a toolkit to create new management modules that include new, supporting device model types and application model types.

This section contains the following topics:

- Using a Developer ID (see page 27)
- SpectroSERVER Database Protection (see page 29)
- Considerations on Database Migration (see page 30)
- About Starting the Model Type Editor (see page 31)
- Overview of the User Interface (see page 33)
- Commit Changes to the SpectroSERVER Database (see page 34)
- Sorting and Filtering Lists (see page 35)
- Add and Remove Columns from Tables (see page 36)
- Exit the Model Type Editor (see page 36)

Using a Developer ID

CA Spectrum uses developer IDs to help ensure that objects such as model types that are created by users and application integrators have unique identifiers and, therefore, can be distributed to other users without conflict.

There are two types of developer IDs:

**Registered developer IDs**

This is a unique, CA-assigned developer ID that protects a developer’s model types, attributes, relations, and meta-rules from being edited by other users.

**The default developer ID**

This is the developer ID applied when the current user is not registered with CA as a CA Spectrum developer; the code designation is DF, with a Developer ID value of 0xffff.

Whenever you create a model type, attribute, or relation, the Model Type Editor uses the developer ID that is currently active (loaded in the database) to create the ID or handle for the new object. This ID is either your unique, registered developer ID or the default ID.
The Model Type Editor also uses developer IDs to determine the access privileges of users with respect to creating, modifying, and destroying modeling catalog objects (model types, attributes, relations, and meta-rules).

- Any developer can use the Model Type Editor as the default developer. However, the modeling catalog objects created using the default developer ID are not protected.
- To obtain a developer ID from CA, contact CA Spectrum Support.
- To be issued a developer ID, you must have purchased the Level 1 toolkit.

**Note:** For more information about the toolkit, see the Integrator Guide.

You can activate your developer ID by loading the developer information file that contains the ID into the SpectroSERVER database. This is a one-time operation after you have initialized the database. However, if you reinitialize the database, you must reload the information.

**Note:** Activate your developer ID using SSdbload with the -d option. For more information, see the discussion about loading developer information in the Database Management Guide.

**More information:**

[Access Privileges With Developer ID](#) (see page 28)

### Access Privileges With Developer ID

Your developer ID determines your access privileges with respect to model types, attributes, relations, and meta-rules.

If you work in the Model Type Editor using the default developer ID, you can:

- View, create, modify, delete, and export model types and attributes that were created with the default developer ID. This includes modifying the derivation of model types by adding and removing base model types.
- View model types that were created by other registered developers.
- Create, modify and delete relations and meta-rules that were created with the default developer ID.
- Export model types, attributes, relations, and meta-rules that were created with the default developer ID.
- Import model types, attributes, relations, and meta-rules from another compatible database.
- Import a Management Information Base (MIB) text file.
If you work in the Model Type Editor using a registered developer ID, you can:

- View, create, modify, or delete model types and attributes that were created with your registered developer ID. This includes modifying the derivation of model types by adding and removing base model types.
- Create, modify, and delete relations and meta-rules that were created with your registered developer ID.
- Export model types, attributes, relations, and meta-rules that were created with your registered developer ID.
- Import model types, attributes, relations, and meta-rules from another compatible database.
- Import a MIB text file.

**Important!** We recommend all users register with CA as a CA Spectrum developer to obtain a registered developer ID.

A registered developer can export model types, attributes, relations, and meta-rules that were created using a registered developer ID. This not only removes ID conflicts, but also protects the objects from accidental or deliberate modification. As indicated previously, if you use the default developer ID, you can export only the model types, attributes, relations, and meta-rules that were created with the default developer ID. It is probable that the IDs (handles) of these objects will conflict with those on another system to which you might want to export the objects. In addition, the exchange of such objects between systems using the default developer ID means that the objects are susceptible to modification or corruption on the receiving system.

**SpectroSERVER Database Protection**

When a program or process such as the Model Type Editor accesses the SpectroSERVER database, a soft lock file named .VNMD.BLOCK is created. The lock file is a safety feature that protects the database by restricting access to one CA-developed application or process at a time. Because non-CA applications or tools may not check for this lock, exercise care when using these applications to prevent concurrent access to the SpectroSERVER database; this can result in corruption of the database.

While lock files are removed automatically during normal shutdown of a CA Spectrum application, an abnormal shutdown can leave behind a lock file erroneously. In rare situations like this, you can manually remove the database lock. For information about how to do this, see the *Database Management Guide*. 
Use of the Model Type Editor involves several risks to the SpectroSERVER database, such as the accidental destruction of necessary model types, the inappropriate setting of attribute flags, and the creation of more than one database with different model type derivations. For this reason, it is recommended that you adopt the following strategies to help preserve the database:

- Avoid editing the database that you use to model your network until absolutely necessary. First test your model type changes using a test database.
- Restrict the use of the Model Type Editor to individuals who are familiar with the long-term plans for your model type derivation scheme. This can help to prevent unnecessary modifications to the database.
- Do not permit editing across multiple databases by more than one user using the same developer ID. This practice creates conflicts between the IDs of modeling catalog objects, which can only be corrected by manually recreating the affected objects. If two separate databases are being used, verify that the database files are being modified with different developer IDs.
- Use the database management utilities provided with CA Spectrum namely SSdbload and SSdbsave to initialize and copy the database. You may get unpredictable results if you use another method.

Note: For information about using these utilities, see the Database Management Guide.

Considerations on Database Migration

It is recommended that you record all changes that you make with the Model Type Editor because some changes are not migrated when the database is updated to a later version of CA Spectrum. Specifically, if you change attributes (for example, flag settings), the model type hierarchy, or relations and associated meta-rules in the model types supplied by CA or another developer (vendor), most likely the changes will not be migrated when the database is upgraded, and you will need to reapply them manually.

To preserve the default values of attributes, you can enable the Preserve Value attribute descriptor flag on the relevant attributes. This flag prevents changed default values from being overwritten by subsequent database updates. However, be aware that enabling this flag may prevent you from receiving intended changes pertaining to the same attributes.

More information:

Flags (see page 45)
About Starting the Model Type Editor

To start the Model Type Editor for the first time, obtain a registered developer ID. You can activate the ID by loading the information into the SpectroSERVER database.

**Note:** For more information, see the *Database Management Guide*.

**Important!** Only one application or process can access the SpectroSERVER database at a time. As a result, after you start the Model Type Editor, all other CA Spectrum applications, including the SpectroSERVER, are denied access. While CA-developed applications automatically deny access to other CA applications as needed, be aware that some third-party applications do not. Database corruption can result.

In rare situations, the SpectroSERVER database is not closed properly by a process, for example, during a power failure. In these situations, the database lock erroneously remains in effect and prevents you from starting the Model Type Editor. To use the Model Type Editor, you must have read and write permissions to the files in the `<$SPECROOT>\SS` directory.

**Note:** For information about removing a database lock, see the *Database Management Guide*.

Start Model Type Editor from the Control Panel

You can start the Model Type Editor from the Control Panel.

**Follow these steps:**

1. Stop the SpectroSERVER if it is running.
2. Verify that no other programs that can access the SpectroSERVER database are running.
3. Open the CA Spectrum Control Panel and click Configure, Model Type Editor.

   The Model Type Editor opens. The Root model type is set as the current model type. The Root model type is the model type at the highest point in the model type hierarchy.
Start Model Type Editor from the Command Line

You can also start the Model Type Editor from the command line.

**Follow these steps:**

1. Stop the SpectroSERVER if it is running.
2. Verify that no other programs that can access the SpectroSERVER database are running.
3. Log in to a shell environment if you are running on Unix or Linux, or open a command prompt if you are running on Windows.
4. Change to the directory that contains the SpectroSERVER database that you want to modify using the Model Type Editor.
   
   **Note:** The executable file for the Model Type Editor is installed in `<$SPECROOT>/SS-Tools`, but it must be called from the directory that contains the SpectroSERVER database that you want to modify. Typically, this directory is CA Spectrum/SS. The directory should contain the database files, which consist of paired *.db and *.ix files, miscellaneous files, and supporting subdirectories.

5. Enter the following command:

   ../SS-Tools/mte

   The Model Type Editor opens. The Root model type is set as the current model type. The Root model type is the model type at the highest point in the model type hierarchy.
Overview of the User Interface

The following image identifies the main work areas, or panels, in the Model Type Editor user interface.

Use the Navigation panel on the left to search for and select a model type, attribute, or relation. Information about a selected object appears in the Contents panel on the right. Use the Contents panel to modify the object, delete the object, or create related objects (for example, to derive a new model type from the current model type).
When you work with attributes, a Component Detail panel lets you view and modify the descriptor values of the current attribute.

**Commit Changes to the SpectroSERVER Database**

While you work in the Model Type Editor, you work with a temporary cache of database objects called the *working catalog*. The Model Type Editor adds only the database objects that are required for your activities to the working catalog.

The Model Type Editor automatically saves any changes that you make to model types, attributes, relations, and meta-rules to the working catalog. You can add them to the permanent catalog in the SpectroSERVER database on demand, or you can add them when you exit the application. You are prompted to add them when you exit the application to avoid losing your work.

You can commit changes to the SpectroSERVER database on demand.

**Follow these steps:**

1. Click File, Commit to Database.  
   A confirmation dialog opens.
2. Click OK. 
   Your changes are saved in the permanent catalog in the SpectroSERVER database. As a result, subsequent calls to the database for the affected objects retrieve the updated versions of those objects.
Sorting and Filtering Lists

To make it easier to work with large lists of modeling catalog objects (for example, model types or attributes), you can sort the lists in tables in ascending or descending order by clicking any column header. You can also sort lists using multiple column headers. For example, the following image shows a list of attributes first sorted in ascending order by Type and then sorted in ascending order by Attribute Name, as indicated by the icons in the column headers.

You can also use the Filter and Search text boxes provided on the various tabs in the Model Type Editor to display only the catalog objects whose names or IDs include a specific character string regardless of case (uppercase or lowercase). In the Filter text boxes, this limits the list of already displayed names. In the Search text box for attributes, this narrows the search criteria applied against the working catalog.

Keep in mind the following as you filter and search for modeling catalog objects:

- To filter lists by ID (for example, model type ID), the ID column must be displayed in the table.
- The Search text box for attributes always searches using both the attribute name and attribute ID as criteria.
- In most cases, a filter or search criterion remains in effect until you clear it by deleting the character string. There are a few actions that automatically clear a filter, for example, when you add a new attribute to a model type.

More information:

Add and Remove Columns from Tables (see page 36)
Add and Remove Columns from Tables

You can modify the information that is displayed in any table in the Model Type Editor by adding and removing columns from the table.

Follow these steps:
1. Right-click the table heading.
   The Table Preferences dialog opens.
2. Click the Columns tab, and select the columns you want to display.
3. (Optional) Change the table sorting and font using the controls on the Sort and Font tabs.
4. Click OK.

Exit the Model Type Editor

When you close the Model Type Editor, you are prompted to save any changes that you have made to the working catalog. Saving them propagates the changes to the SpectroSERVER database.

Note: For information about the difference between the working catalog and the permanent catalog, see Commit Changes to the SpectroSERVER Database (see page 34).

Follow these steps:
1. Click File, Exit.
   A confirmation dialog opens.
2. Click OK.
   If you have made changes to the working catalog that is stored in cache, you are prompted to save the changes to the permanent catalog in the SpectroSERVER database.
3. Do one of the following:
   ■ To save the changes to the permanent catalog, click Yes.
   ■ To discard the changes, click No.
   The changes are saved, if appropriate, and the Model Type Editor is closed.
Chapter 3: Creating and Modifying Model Types

This section contains the following topics:

Introduction (see page 37)
Attributes of Model Types (see page 37)
Standard Attribute Descriptors (see page 43)
Special Attribute Descriptors (see page 47)
Search for and Display Model Types (see page 52)
Search for and Display Attributes (see page 53)
Create a Model Type (see page 53)
Delete a Model Type (see page 54)
Working with Base Model Types (see page 56)
Import MIBs (see page 60)
Set Model Type Flags (see page 62)
Working with Attributes (see page 63)
Working with Attribute Groups (see page 69)

Introduction

This chapter provides information about how to do the following:

■ Extend and customize the default modeling catalog provided with CA Spectrum by creating and modifying model types.
■ Import MIBs.
■ Create and manage attribute groups. Attribute groups make it easier to work with logically related attributes in the Model Type Editor.

Attributes of Model Types

A model type is defined by the following attributes and classes of attributes:

■ Developer ID
■ Model type name
■ Model type ID (handle)
■ Base model types
■ Derived model types
Attributes of Model Types

- Flags
- Custom attributes

Developer ID

The developer ID that was active when the attribute was created. This can be a registered ID obtained from CA or the default ID. Once a developer ID is assigned to an attribute, it cannot be changed.

After you create an attribute, the access privileges of users for modifying, deleting, and exporting it are determined based on whether the active developer ID matches the developer ID associated with the attribute.

More information:

Access Privileges With Developer ID (see page 28)

Model Type Name

A descriptive identifier that typically describes the model type’s function. Model type names should be a maximum of 128 characters and should only consist of letters, numbers, underscore (_) characters, and dashes (-). Spaces, punctuation, or other symbols should not be used.

Note: A model type name does not need to be unique across the modeling catalog, but you should help ensure it is unique across the model types created under a given developer ID. CA Spectrum differentiates model types using both the model type name and the developer ID component in the model type ID.

While you can rename a model type, be aware that this affects the AlertMap file that is specific to the model type because the file is located in the following directory:

SS/CsVendor/<developer name>/<model type name>

If you rename a model type, you will need to manually create a new directory based on the new model type name and then move the AlertMap file.

Also be aware that it is possible—although unusual and discouraged—that an inference handler depends on the name of a model type rather than its model type ID (handle).
Model Type ID (Handle)

A unique ID that is assigned to the model type when the model type is created. The ID is generated by ORing together the value of the active developer ID and a counter value in the range from 0x0001 and 0xFFFF, as shown in the following illustration.

<table>
<thead>
<tr>
<th>Developer ID prefix</th>
<th>Unique sequence number, never repeated for same developer ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1 to 4 hexadecimal characters, starting in position 1, leading zeros not shown)</td>
<td>(4 hexadecimal characters, starting in position 5)</td>
</tr>
</tbody>
</table>

Once a model type ID is assigned to a model type, it cannot be modified. Additionally, if the model type is deleted, the ID is not reused.

Base Model Types

A ranked list of parent model types from which the current model type directly inherits attributes and SpectroSERVER intelligence.

Because a model type can inherit the same attribute from two or more inheritance paths that have a common originating model type, the ranking of the base model types is used to determine the inheritance path.

You cannot remove attributes inherited from a base model type, and you have limited capabilities to edit them. To remove an inherited attribute, or to have full editing capabilities, you must use the same developer ID that was active when the model type was created, and you must make the change in the originating model type. The originating model type is the model type in which the attribute was created.

Derived Model Types

An alphabetically sorted list of child model types that are directly derived from the current model type. That is, the derived model types inherit attributes and SpectroSERVER intelligence from the current model type—and by extension—from the base model types of the current model type.
Model Type Flags

There are several model type flags that control basic characteristics of a model type.

You can only modify flag settings for a model type if you are using the developer ID that was active when the model type was created.

The following describes each model type flag:

Visible

If enabled (checked), the model type is exposed in the output when you run a report on the modeling catalog using the reports database utility provided with CA Spectrum. If disabled (not checked), the model type is only exposed in the output if the model type was creating using the developer ID that is currently loaded in the database. The default value is enabled.

Note: The Visible flag does not affect what a user can view in the Model Type Editor or in OneClick, nor does it affect the operation of the SpectroSERVER. However, future releases of CA Spectrum may respect the flag and stops access to model types and models when the flag is disabled. For more information about the reports utility, see the Database Management Guide.

Instantiable

If enabled (checked), models of the model type can be created in the SpectroSERVER database by users and inference handlers. If disabled (not checked), models of the model type cannot be created, and the model type is not available in OneClick when creating a model by model type. The default value is disabled.

Changing the condition of the Instantiable flag for a model type does not affect existing models of that type. For example, if you were to disable the flag after creating models A and B, models A and B would be unaffected. However, you could not create additional models after disabling the flag.

Derivable

If enabled (checked), the model type can be used as a base model type from which other model types can be derived. If disabled (not checked), the model type cannot be used as a base model type. The default value is enabled.

CA Spectrum checks the Derivable flag only as new model types are being created. In other words, if you derive several model types from model type A and then disable the Derivable flag for model type A, the newly derived model types are unaffected, but you are no longer able to derive additional model types from model type A.

No Destroy

If enabled (checked), models of the model type cannot be deleted from the SpectroSERVER database by users and inference handlers. If disabled (not checked), models of the model type can be deleted. The default value is disabled.

Note: You can only enable this flag if you also enable the Instantiable flag.
**Unique**

If enabled (checked), only one model of the model type can exist in the SpectroSERVER database. If disabled (not checked), additional models of the model type can exist. The default value is disabled.

CA Spectrum checks the Unique flag only as models are being created. If you create several models of a model type and then disable the Unique flag, the previously created models are unaffected, but you are no longer able to create additional models of the model type.

**Note:** You can only enable this flag if you also enable the Instantiable flag.

**Required**

If enabled (checked), the SpectroSERVER creates a model of the model type at server startup if a model does not already exist. If disabled (not checked), a model of the model type is created only if requested by a user, inference handler, or application. The default value is disabled.

**Note:** You can only enable this flag if you also enable the Instantiable flag.
Custom Attributes

In addition to the model type attributes described earlier in this section (such as model type ID), a model type has many other attributes that are inherited from base (parent) model types or that originate in the model type itself. You can view this list of attributes on the Attributes tab in the Contents panel, as shown in the following image.
You can click the Attribute Name column header to sort the attribute list alphabetically in ascending or descending order.

The Originating Model Type column displays the model type in which each attribute was created.

You can click the Originating Model Type column header to sort the list alphabetically in ascending or descending order. This lets you group together and, therefore, identify all of the attributes in the current model type that originate in the model type itself or in a base (parent) model type.

The values for the attribute descriptors of the selected attribute are displayed in the Component Detail panel.

### Standard Attribute Descriptors

As discussed earlier in this guide, every attribute is described by a set of characteristics called **attribute descriptors**. The following are attribute descriptors that you can only modify in the originating model type (the model type in which the attribute was defined):

- Developer ID
- Attribute name
- Attribute ID (handle)
- Type
- Flags
- Group Name and Group ID

### Developer ID

The developer ID that was active when the attribute was created. This can be a registered ID obtained from CA or the default ID. Once a developer ID is assigned to an attribute, it cannot be changed.

After you create an attribute, the access privileges of users for modifying, deleting, and exporting it are determined based on whether the active developer ID matches the developer ID associated with the attribute.

**More information:**

Access Privileges With Developer ID (see page 28)
**Attribute Name**

An attribute name is a descriptive identifier. Attribute names should be a maximum of 128 characters and should only consist of letters, numbers, underscore (_) characters, and dashes (-). Spaces, punctuation, or other symbols should not be used.

**Note:** An attribute name does not need to be unique across the modeling catalog, but it should be unique across the attributes in the model type that were created using the same developer ID. In other words, within a single model type, two attributes can have the same name if they were created using different developer IDs.

**Attribute ID (Handle)**

A unique ID that is assigned to the attribute when the attribute is created. The ID is generated by ORing together the value of the active developer ID and a unique sequence number (counter value), as shown in the following illustration.

- Developer ID prefix (1 to 4 hexadecimal characters, starting in position 1, leading zeros not shown)
- Unique sequence number, never repeated for same developer ID (4 hexadecimal characters, starting in position 5)

```
```

Once an attribute ID is assigned to an attribute, it cannot be modified. Additionally, if the attribute is deleted, the ID is not reused.

**Type**

The attribute type defines the kind of data that the value of the attribute can hold. If the attribute represents a managed object that is defined in the MIB, you should set the type to correspond to the type defined in the MIB.
If an attribute requires a list of values, you must select the List check box.

**Note:** After you create an attribute, you cannot change its type or whether it stores a single value or a list of values. You must delete the attribute and create a new one of the desired type. There are two exceptions to this. First, you can change the type from Octet String to Text String, and vice versa. Second, if the type is numeric (Integer, Counter, Enumeration, Gauge, or Time Ticks), you can change from one of these numeric types to another.

### Flags

Flags—also referred to as descriptor flags—inform the SpectroSERVER of the characteristics of the attribute, for example, who can access the attribute's value. The following describes each flag.

**External**

Indicates the attribute value is maintained outside of the SpectroSERVER and that update of the value is done either at a polling interval or upon user request.

**Note:** If you set this flag, you must supply an OID prefix, which specifies the location of the managed variable in the MIB. Also note that if you set this flag, you cannot set the Shared flag.

**Readable**

Informs the SpectroSERVER that a client or other application can read the attribute value from the SpectroSERVER database.

If you set the External flag, you should set this flag in accordance with the MIB definition of the Readable variable for the attribute. Otherwise, you can set this flag as desired.

**Writable**

Informs the SpectroSERVER that a client or other application can write the attribute value to the SpectroSERVER database.

If you set the External flag, you should set this flag in accordance with the MIB definition of the Writable variable for the attribute. Otherwise, you can set this flag as desired.

**Shared**

Declares that only one value exists for the attribute and that all models of the given model type share the same value. The value is not duplicated for each model in memory or in the database.

**Note:** You can only set this flag if you also set the Database flag or the Memory flag. Also, if you set the External flag or the Polled flag, you cannot set this flag.
Guaranteed

Guarantees that the attribute will continue to exist and can be used in future model type derivations. Once set, this flag cannot be disabled except by the developer who created the attribute.

Note: Setting this flag only guarantees the presence of the attribute, not its value or values.

If this flag is disabled, any Model Type Editor user can enable (set) or disable the Extended flags of the attribute. If this flag is enabled, users having developer IDs other than the one used to create the attribute can only set the Extended flags; they cannot disable them. The user having the developer ID used to create the attribute can set or disable the Extended flags at any time.

Global

Indicates that the attribute’s value will be kept consistent across duplicate models in all landscapes in a distributed SpectroSERVER (DSS) environment.

Global attributes are only maintained for models of the User and UserGroup model types. These are duplicate model types, that is, across a distributed environment, multiple models of these types effectively represent the same user or user group. As such, a change to a model in one landscape should be propagated to all corresponding, duplicate models in all other landscapes.

Note: You can only set this flag if you also set the Memory flag or the Database flag.

Preserve Value

Indicates that imported files will not overwrite the attribute’s default value currently stored in the database.

If you customize the default values of one or more model types to meet specific requirements, and then you set this flag, your customizations (the specialized default values) will remain in place when the model types are updated by subsequent versions of CA Spectrum.

If you are the owner of the attribute (that is, the attribute was created using the developer ID that is currently active), you can modify all of the flags described in this section in the originating model type regardless of whether you are the owner of the associated model type.

If you are not the owner of the attribute, or if the attribute is inherited, you cannot modify these flags. However, you can modify the attribute’s extended flags.

More information:

OID Prefix and OID Reference (see page 50)
Extended Flags (see page 48)
Group Name and Group ID

A *group* is a logical collection of related attributes in a model type. Groups make working with related attributes easier because they let you to define and use a user-defined sorting mechanism in the Model Type Editor. You can create groups, assign attributes to them, and then add the Group Name or Group ID as a column header in the table of attributes on the Attributes tab in the Contents panel. This lets you to then click the column header to quickly group together and view together all of the attributes within a group.

*Group Name* specifies the name of the group to which the attribute is assigned, and *Group ID* specifies the ID of that group.

Like for any standard descriptor, you can change the group to which an attribute is assigned if you are modifying the model type in which the attribute was created, and if you are using the developer ID that was used when the attribute was created (that is, you own the attribute).

By default, an attribute has a Group ID value of 0x00000000, which indicates the attribute is not assigned to a group. If you assign an attribute to a group, and you subsequently decide it should not be assigned to a group, you must reassign the attribute to the Root group. You cannot restore a Group ID value of 0x00000000 in the Model Type Editor.

**Note:** If you set an attribute's Group Name to the name of another developer’s attribute group, and that developer’s attribute group is not distributed with the next version of CA Spectrum, the attribute's Group Name value is reset to <no group>, and its Group ID value is reset to 0x00000000 to indicate it is not assigned to a group.

Special Attribute Descriptors

As discussed earlier in this guide, every attribute is described by a set of characteristics called *attribute descriptors*. The following are attribute descriptors that you can specialize; you can modify them at any level of the inheritance hierarchy:

- Default value
- Extended flags
- OID Prefix and OID Reference
- Polling Group
**Default Value**

The initial value or values for the attribute. An attribute can inherit its default value from a base model type or specify its own default value (a process called specialization). In the latter case, all model types derived from the specialized model type inherit the changed attribute value.

**Note:** While you can specify a default value for an attribute using this attribute descriptor of a model type, the actual value or values of the attribute are often different in models. This is definitely the case for external attributes (attributes for which the External flag is enabled), since these attributes maintain their values by polling devices at specified intervals or making updates upon user request.

**More information:**

*Model Type Inheritance* (see page 17)

**Extended Flags**

Extended flags—also referred to as extended descriptor flags—inform the SpectroSERVER of additional characteristics of the attribute, for example, whether the attribute should be polled.

You enable and disable an attribute's extended flags using the Model Type Editor, but the flags are used by the SpectroSERVER.

If an attribute's Guaranteed flag is disabled, any Model Type Editor user can enable or disable the extended flags. If the Guaranteed flag is enabled, users having developer IDs other than the one used to create the attribute can only enable the Extended flags; they cannot disable them. The user having the developer ID used to create the attribute can enable or disable the extended flags at any time.
The following list describes each extended flag.

**Memory**
Stores a copy of the attribute's value in memory. When the SpectroSERVER is restarted, the value is reset to the default value; the value in memory is not preserved.

**Note:** You must set either this flag or the Database flag if you set either the Shared flag or the Global flag.

**Database**
Stores a copy of the attribute's value in the database so that it is preserved across SpectroSERVER restarts.

**Note:** You must set either this flag or the Memory flag if you set either the Shared flag or the Global flag.

**Polled**
Informs the SpectroSERVER that the attribute should be polled at the polling interval in order to update its value. This is only meaningful if the External flag is also set. If the Memory flag is also set for the attribute, the value retrieved by the poll is also stored in memory.

**Note:** You cannot set this flag if you set the Shared flag.

If you set this flag, you should assign the attribute to an appropriate polling group; all attributes of a polling group are polled together. If you set both this flag and the Logged flag, you must group the attribute with other attributes that also have both the Polled and Logged flags set.

**Logged**
Causes the value of the attribute to be recorded in the Distributed Data Manager (DDM) database. If you enable this flag, you should assign the attribute to an appropriate polling group; all attributes of a polling group are logged together. If you enable both this flag and the Polled flag, you must group the attribute with other attributes that also have both the Polled and Logged flags enabled.

**Note:** Logging occurs at a user-specified interval stored in the Poll_Log_Ratio attribute. By default, this attribute is set to 0 for device model types, which effectively disabled CA Spectrum’s native logging method. If you require the logging of device, attribute, and port statistics, it is recommended that you use CA Spectrum SS Logger instead of the native method, the latter of which writes the information to the Distributed Data Manager (DDM) database. CA Spectrum SS Logger is a CA Spectrum command-line application that lets you to log statistics directly to ASCII files, which reduces the load on the DDM database and eliminates the need to export the data. Of equal importance, CA Spectrum SS Logger gives you greater control over what data to log and how frequently to log it. For more information, see the CA Spectrum SS Logger User Guide.
External attributes that are set to be polled and logged may return "noSuchName" errors when a management module is based on a more current firmware version than the managed device supports. To reduce unnecessary network traffic, CA Spectrum automatically suspends normal polling and logging for attributes that return this error, and moves the attribute to the unsupported polling attribute group.

Once an attribute has been moved to the unsupported polling attribute group, CA Spectrum generates an event (0x10970). By default, this event is not logged and does not generate an alarm. However, you can change this event processing using the Event Configuration application in OneClick. CA Spectrum attempts to read the attribute at the interval specified by the unsupported_attr_poll_interval. By default, the value of the unsupported_attr_poll_interval is 12 hours. This value can be changed by manually adding the parameter and the desired interval (in seconds) to CA Spectrum's .vnmrc file.

When an attribute that had previously been reporting a "noSuchName" error reports a successful poll, CA Spectrum generates an event (0x10971). By default, this event is not logged and does not clear an alarm. However, you can also change this event processing using the Event Configuration application.

Thus, the unsupported_attr_poll_interval lets normal polling and logging for an attribute to resume automatically without requiring a SpectroSERVER restart or the destruction and recreation of the models that have the attribute.

**Note:** For more information about the unsupported_attr_poll_interval, see the Distributed SpectroSERVER Administrator Guide. For more information about the Event Configuration application, see the Event Configuration User Guide.

**More information:**

Flags (see page 45)
Polling Group (see page 51)

**OID Prefix and OID Reference**

The OID Prefix descriptor and OID Reference descriptor apply only to attributes that have the External flag enabled, that is, attributes that represent managed variables within tables in a MIB.

The OID Prefix specifies the column within the MIB table that contains the variable. You must use dotted-decimal notation when entering the OID prefix.

The OID Reference (instance ID) specifies the name of an attribute whose value serves as an index used to define the instance of the variable within the column.
The OID prefix is concatenated with the OID reference to define a complete object identifier (OID) for the variable being monitored. As an example, the following image shows the resulting OID formed using the following:

- An OID Prefix set to the OID for EnetPortColls
- An OID Reference set to the ID of an internal attribute, defining the instance ID.

**Polling Group**

The polling group to which the attribute belongs. All of the attributes in a group are polled together and logged together. The SpectroSERVER polls groups of pollable attributes by polling group, one at a time beginning with the group with the lowest number.

Specify a number between 0 (zero) and 255 inclusive. If you do not specify a value for a new or existing attribute, the attribute is automatically given a value of 0.
When attributes are polled but not logged (that is, the Polled flag is enabled, but the Logged flag is disabled), the only limit to the number of attributes within a polling group is the transmission length limits imposed by the following:

- The transmission protocol (Ethernet, FDDI, and so on)
- The management protocol (CA proprietary, SNMP, and so on)

**Search for and Display Model Types**

You can search for a model type in the working catalog by name or ID.

**Follow these steps:**

1. In the Navigation panel, click the Model Types tab.
   
   The names of all of the model types in the modeling catalog are listed.

2. In the list, select the name of the model type that you want to examine.
   
   To locate and select a specific model type, you can take the following steps:

   - Enter a text string in the Filter text box to filter the list to include only the model types whose names or IDs contain the string. To filter the list by ID, the ID column must be displayed in the table.
   - Click the Model Type bar at the top of the list to change the alphabetical sorting from ascending to descending or vice versa.

   The selected model type becomes the current model type, and information about it is displayed in the Contents panel.

3. To navigate to a base (parent) or derived (child) model type of the current model type, take the following steps:

   a. Click the Hierarchy tab in the Contents panel.
   
   b. Double-click a base or derived model type to make it the current model type.

   **Note:** You can filter the list of base or derived model types using the Filter text boxes.

   c. Repeat the preceding step as many times as needed to navigate to the desired model type in the model type hierarchy.

**More information:**

[Sorting and Filtering Lists](see page 35)
Search for and Display Attributes

You can search for an attribute in the working catalog by name or ID.

**Follow these steps:**

1. In the Navigation panel, click the Attributes tab.
2. In the Search text box, enter a text string to examine against the attribute names and IDs. (You do not need to display the ID column in the table to search by ID.)

   The attributes with names or IDs that include the string that you entered are displayed in the list. In addition, the *originating model type* for each attribute is displayed. The originating model type is the model type where the attribute was created. Use this model type to modify all of the descriptors for an attribute.
3. Select the name of the attribute that you want to examine from the list.

   The corresponding originating model type is made the current model type in the Contents panel, and information about the attribute you selected is displayed in the Component Detail panel.

Create a Model Type

When you want to represent a new device or some other entity that is not currently defined as a model type in the CA Spectrum database, you must create a model type.

**Follow these steps:**

1. Determine the attributes that are required for the new model type.
2. Identify the base model types from which the new model type can inherit its attributes or inherit as many of them as possible.
3. Set the existing model type that has *most* of the attributes that you need for the new model type as the current model type.

   This is the model type from which you will directly derive the new model type.
4. Click ![Derive a new Model Type](image)

   **Note:** If you are not able to click the button, verify that the current model type has its Derivable flag set. You cannot derive a model type from the model type unless this is the case.

   The Create Derived Model Type dialog opens.
5. For Name, enter the name of the new model type.

The name should be a maximum of 128 characters and should only consist of letters, numbers, underscore characters (_), or dash characters (-).

**Important!** A model type name is not required to be unique across the modeling catalog. However, we recommend using unique names across the model types that were created under the currently active developer ID. CA Spectrum differentiates model types using both the model type name and the developer ID component in the model type ID. In addition, reusing a model type name is not recommended.

6. Click OK.

The new model type is created and is set as the current model type.

If you click the Attributes tab, you can examine its attributes, which are those inherited from the base model type that you selected in step 1.

7. If the new model type requires additional attributes that can be inherited from other base model types, add those model types as base model types.

8. If the new model type requires additional attributes that cannot be inherited from other base model types, add those attributes directly to the new model type.

9. Set the model type flags for the new model type as appropriate.

   For example, if the new model type is a final model type (that is, it is meant to be instantiated and used by models represented in OneClick), set the Instantiable flag.

10. Restart the OneClick web server.

   You can now create a model of this new model type in the OneClick console.

**More information:**

- [Model Type Flags](#) (see page 40)
- [How to Determine the Base Model Types for a New Model Type](#) (see page 56)
- [Model Type ID (Handle)](#) (see page 39)
- [Add a Base Model Type to a Model Type](#) (see page 58)
- [Add an Attribute to a Model Type](#) (see page 63)

---

**Delete a Model Type**

Deleting a model type involves deleting all models of that model type, removing all derived model types for the model type, and finally removing all base model types for the model type. In effect, this completely removes all dependencies on the model type from the model type hierarchy so that the model type can safely be destroyed in the database.
Follow these steps:

1. In OneClick, use the Locator tab to find all of the models of the model type you intend to delete, and delete the models.
   
   **Note:** For more information, see the *Modeling and Managing Your IT Infrastructure Administrator Guide*.

2. Shut down OneClick and the SpectroSERVER, and start the Model Type Editor.

3. Set to current the model type that you want to delete.

4. Examine the hierarchy of the model type in order to identify the consequences of deleting it. For example, check whether any attributes that originate in the model type are critical to any derived model types that inherit them. Then resolve any predictable problems with inheritance factors.

5. Remove all derived model types from the model type that you want to delete:
   
   a. Make the first derived model type the current model type instead.
      
      The model type you want to delete is now displayed in the list of base model types for the new current model type.
   
   b. In the list of base model types, select the model type that you want to delete, and click - (Remove selected base Model Type).
      
      **Note:** The tooltip displayed when you hover over the button indicates whether this action will result in only the removal of the selected model type as a base model type or also the deletion of the current model type. If the current model type has no derived model types, it will also be deleted when you remove the last of its base model types because this means it is no longer a part of the model type hierarchy.
      
      A confirmation dialog opens.
   
   c. Click Yes.
   
   d. Repeat the preceding three steps as many times as needed until the model type that you want to delete is no longer being used as base model type for any model types.

6. Remove all base model types from the model type that you want to delete:
   
   a. Set current the model type that you want to delete.
   
   b. In the list of base model types, select the first model type, and click - (Remove selected base Model Type).
      
      **Note:** As previously mentioned, if the current model type has no derived model types, it will also be deleted when you remove the last of its base model types.
A confirmation dialog opens.

c. Click Yes.

d. Repeat the preceding two steps until the model type that you want to delete no longer has any base model types.

**Note:** Removing the last base model type will also delete the model type that you want to delete. You can only remove the last base model type from the current model type when the current model type has no derived model types.

**More information:**

[Search for and Display Model Types](see page 52)

---

### Working with Base Model Types

If a model type requires additional attributes above and beyond those inherited from the model type from which it is first derived, you can add additional base model types. This can add attributes and inference handlers to the model type.

While the addition of attributes typically does not cause any problem, you may add inference handlers that lock attributes, which may make old inference handlers fail. Also, be aware that if a model type has two or more base model types that share a common ancestor model type, the model type has more than one way to inherit attributes and intelligence originating in that common ancestor. As described in [Model Type Precedence](see page 22), CA Spectrum resolves this type of situation by assigning rankings to base model types. The base model type with the lower ranking (that is, the base model type that is higher in the list of base model types) is the base model type from which the derived model type inherits the shared attribute.

**Note:** You can change a base model type's ranking by removing the base model type and re-adding it in a specific location in the ranked list of base model types.

---

### How to Determine the Base Model Types for a New Model Type

To identify the model types that you want to use as base model types, use the Hierarchy tab to navigate up and down through the model type hierarchy to specific model types, and then use the Attributes tab to examine their attributes. Continue this process until you have identified the following:

- The existing model type that contains *most* of the attributes needed for the new model type. You should derive the new model type directly from this model type.
- The other model types that can provide some or all of the other attributes needed for the new model type.
As you identify the base model types for the new model type, keep the following guidelines in mind:

- You can only use derivable model types as base model types, that is, model types that have the Derivable flag set.
- Use as few base model types as possible in order to keep the hierarchy simple.
- Avoid adding base model types that do not contribute significantly to the model type being created. And avoid base model types that contain a significant number of unnecessary attributes. Because you cannot remove inherited attributes, ignoring this guideline can quickly add an excessive number of attributes, wasting storage space and perhaps affecting performance.
- You might want to add one or more base model types that provide access to specialized MIB attributes and intelligence. We recommend creating a set of MIB-specific model types, each one containing the intelligence and attributes related to a specific MIB. Create a new model type based on GnSNMPMibDerPt and give it a name that identifies associated MIB. Then import the MIB into the model type. This approach lets you add the MIB-specific model type as a base model type to multiple model types. The associated attribute IDs remain the same across all derived model types.

**Important!** For new device model types, the GnSNMPDev model type is often the best starting point. This model type contains the basic attributes and intelligence that are typically required for integration with core CA Spectrum functionality. For new application model types, select from several possible starting points, such as GnSNMPMibDerPt and GnSNMPAppDerPt. For more information, see the Certification User Guide.

**More information:**

[Import MIBs](#) (see page 60)
Add a Base Model Type to a Model Type

You can add a base model type to an existing model type.

Follow these steps:

1. Set current the model type for which you want to add a base model type.
2. Do one of the following:
   - To give the new base model type the highest ranking of all of the listed base model types (that is, place the base model type last in the list), proceed to the next step.
   - To give the new base model type a lower ranking (that is, place the base model type higher in the list), select the base model type directly beneath the location where you want to insert the new base model type. This step adds the new base model type in that location.

   Note: When a derived model type can inherit the same attribute from two or more inheritance paths that have a common originating model type, the derived model type inherits the attribute from the base model type with the lowest ranking.

3. Under Base Model Types, click + (Add a base Model Type).

   The Select New Base Model Type dialog opens.

4. Select the model type to add as a base model type.

   Note: To rapidly locate and select a specific model type, enter a text string in the Filter text box to filter the list.

5. Click OK.

   The selected model type is added as a base model type of the current model type.

More information:

Search for and Display Model Types (see page 52)
Sorting and Filtering Lists (see page 35)
How to Determine the Base Model Types for a New Model Type (see page 56)
Model Type Precedence (see page 22)

Remove a Base Model Type from a Model Type

Removing a base model type from a model type is the way in which you remove inherited attributes, inherited meta-rules, or intelligence from the model type. Essentially, this removes the hierarchical relationship between the model type and the base model type in which the undesirable attributes originate (referred to as the originating model type).
You can remove base model types from the current model type if the current model type was creating using the developer ID that is currently active.

You cannot remove the last base model type from the current model type if the current model type has derived model types. In order to break such a connection, you must first navigate to the derived model types and use the following procedure to remove the model type of interest as a base model type with respect to the derived model types. You can then navigate back to the model type of interest and remove its last remaining base model type.

**Note:** Inference handlers are code segments that define the behavior and intelligence of a model type. Problems can occur if you remove a base model type from a derived model type, and the derived model type has associated inference handlers that refer to attributes that used to be inherited from the removed base model type. This sort of dependency can be difficult to detect. Also be aware that removing a base model type also may remove inference handlers that were inherited from that base model type. This may cause anomalies if the removed inference handlers performed some vital function for the model type or for other model types derived from it.

You can remove a base model type from a model type.

**Follow these steps:**

1. Set current the model type for which you want to remove a base model type.
2. Click the Hierarchy tab, and under Base Model Types, select the model type that you want to remove, and click (Remove selected base Model Type).

   **Note:** The tooltip for the button indicates whether this action only removes the selected model type as a base model type or also deletes the current model type. If the current model type has no derived model types, it is also deleted when you remove the last of its base model types because it is no longer a part of the model type hierarchy.

   The selected model type is removed as a base model type.

**More information:**

[Search for and Display Model Types](see page 52)
Import MIBs

CA Spectrum manages devices according to the requirements and values that are specified in their MIB documents. A MIB (Management Information Base) is a database that resides on a network device and represents that device as a hierarchical collection of objects. A MIB object represents an individual element of information, such as the uptime of a device. MIBs themselves are text files with special syntax. A device MIB defines all of the objects that can be managed on the associated device. The MIB organizes this information in a tree structure with branches that organize the managed objects into logical groups.

You can use the Model Type Editor to import both SMIv1 and SMIv2 MIBs. However, when you import a SMIv2 MIB, the Model Type Editor maps the MIB data type to the corresponding data type that is defined in SMIv1. The Model Type Editor also supports most standard text conventions and associated enumerations that can be used in a MIB.

When you create a model type, you typically want to add base model types that provide access to specialized MIB attributes and intelligence. We recommend creating a set of MIB-specific model types, each one containing the intelligence and attributes of a specific MIB. Create a model type from GnSNMPMibDerPt and assign it a name that identifies the associated MIB. Then import the MIB into the model type. This approach lets you add the MIB-specific model type as a base model type to multiple model types and keep the associated attribute IDs consistent across all derived model types.

**Note:** You can only import a MIB into a model type that was created by the currently active developer ID. This rule prevents you from importing the MIB into a model type that you cannot subsequently export.

Use the Model Type Editor to import a MIB into a model type.

**Follow these steps:**

1. Select the model type from which you plan to derive a model type for the MIB import.

   Typically, the starting point is some developer-specific (vendor-specific) model type that is derived from the Manufacturer model type, as illustrated in the following figure:

   ![Diagram](image)

   Root model type
   
   Manufacturer model type
   
   `<developer name>` model type
   
   new model type
However, it can also be a model type that was derived from EntityTypes, MMDeveloper, or another model type in the modeling catalog. Another alternative is GnSNMPMibDerPt, which was designed specifically for importing MIBs into the CA Spectrum database. That model type already contains many needed attributes and relations. Regardless of your selection, you can create device model types at appropriate places in the model type hierarchy later for each device receiving the attributes of the MIB.

**Note:** For more information about designing a new model type, see the Certification User Guide.

2. Click the Hierarchy tab, and create a model type into which to import the MIB information:
   a. Click (Derive a new Model Type).
   b. Enter a model type name.
      
      The name cannot exceed 128 characters and can only consist of letters, numbers, underscore characters (_), or dash characters (-).
      
      **Important!** A model type name is not required to be unique across the modeling catalog. However, supply a name that is unique across the model types that were created under the currently active developer ID. CA Spectrum differentiates model types using both the model type name and the developer ID component in the model type ID.
   c. Click OK.
      
      The derived model type is created and is set as the current model type.

3. Click File, Import MIB.
   
   The MIB Import dialog opens.
   
   **Note:** Sometimes a single file contains multiple MIBs that are delineated with BEGIN and END statements. Include only a single MIB in the file that you import.

4. Click Browse, navigate to the MIB file to import, select the file, and click Open.
5. Click OK to begin the import.

When the import is complete, the MIB Import Complete dialog is displayed to inform you of the number of attributes and attribute groups that were created.

If issues are encountered during the import process, a warning is displayed to inform you that the import was successful but issues were encountered. If the import process fails, an error is displayed.

6. Click OK to close the dialog.

**Important!** When you create a model type, typically you also add support for traps, events, and alarms. You can use the MIB Tools utility in OneClick to add trap support and perform initial event configuration. For more information, see the Certification User Guide. You can then fully configure the events and alarms using the Event Configuration application in OneClick, as described in the Event Configuration User Guide. For more information about creating a management module (including creating model types), see the Certification User Guide.

---

**Set Model Type Flags**

To set and change the flags for a model type, you must be using the developer ID that was active when the model type was created.

**Follow these steps:**

1. Select the model type whose model type flags you want to set.

2. Click the Flags tab and click (Edit) in the top, left corner.

   The Edit Flags dialog opens.

3. Select (enable) and clear (disable) the flags as desired.

4. Click OK.

**More information:**

- [Model Type Flags](#) (see page 40)
- [Search for and Display Model Types](#) (see page 52)
Working with Attributes

You can add or remove attributes from a model type directly and indirectly:

- To add or remove attributes indirectly, add or remove base model types, as described in Working with Base Model Types (see page 56).
- To add or remove attributes directly, create or delete the attributes from the model type itself, as described in the topics in this section.

Add an Attribute to a Model Type

Typically, you will add attributes to model types that you own, that is, types that were created using the currently active developer ID. Because you own the model types, you can subsequently export them and their attributes.

While you can also add attributes to model types that you do not own, you cannot export these model types or their attributes. Typically, exporting model types that you do not own is only required for storing information in the related models when no available attribute is suitable.

Important! We recommend recording all changes that you make with the Model Type Editor. Some changes are not migrated when the database is updated to a later version of CA Spectrum. Specifically, if you modify attributes (such as flag settings), the model type hierarchy, or relations and the associated meta-rules in the model types supplied by CA or another vendor, the changes are typically not migrated on database upgrade. You will have to reapply these changes manually. For more information, see Migrate Changes to a New Version of CA Spectrum.
Follow these steps:

1. Select the model type to which you want to add an attribute.

2. Click the Attributes tab and click 

   The Create an Attribute dialog opens.

   ![Create an Attribute dialog](image)

   - **Name**: Enter the name of the attribute.
   - **Attribute Type**: Select the type of attribute (Boolean, List, etc.).
   - **Default Value**: Enter the default value for the attribute.
   - **Flags**: Select flags such as External, Readable, Writable, Shared, Guaranteed, Global, or Preserve Value.
   - **Extended Flags**: Select flags such as Memory, Database, Polled, or Logged.
   - **OID**: Enter the OID prefix and reference.
   - **Group**: Enter the group name and ID.

   * indicates a required field

   - **OK**
   - **Cancel**
3. Enter values for the attribute's descriptors.

   **Note:** An attribute name does not need to be unique across the modeling catalog, but you should enter a name that is unique across the attributes in the model type that were created using the same developer ID. In other words, within a single model type, two attributes can have the same name if they were created using different developer IDs. In addition, while it is allowed, reusing an attribute name is not recommended.

4. Click OK.

   The attribute is created and added to the working catalog, and it is displayed in the list of attributes on the Attributes tab.

**More information:**

- [Search for and Display Attributes](#) (see page 53)
- [Attribute Descriptors](#) (see page 17)

## Remove an Attribute from a Model Type

You can remove an attribute from the model type in which it was created (referred to as the originating model type) if you own the model type, that is, the model type was creating using the developer ID that is currently active.

When you remove an attribute from a model type, it is also removed from all derived model types that inherit it.

**Important!** We recommend recording all changes that you make with the Model Type Editor. Some changes are not migrated when the database is updated to a later version of CA Spectrum. Specifically, if you modify attributes (such as flag settings), the model type hierarchy, or relations and the associated meta-rules in the model types supplied by CA or another vendor, the changes are typically not migrated on database upgrade. You will have to reapply these changes manually. For more information, see *Migrate Changes to a New Version of CA Spectrum*.

### To remove an attribute from a model type

1. Verify the attribute to be removed is not critical to a derived model type.

   **Note:** If necessary, you can provide an alternate path via a different base model type, or you can recreate the attribute at the derived model type level.

2. Select the model type for which you want to remove an attribute.
3. Click the Attributes tab, select the attribute you want to remove, and click (Delete selected Attribute).
   A confirmation dialog opens.
4. Click Yes.
   The attribute is deleted from the working catalog.

More information:
Search for and Display Model Types (see page 52)

Edit an Attribute

Typically, you will want to modify the attributes of the model types that you own, that is, that were created using the developer ID that is currently active. Because you own the model types, you can subsequently export them and their attributes.

While you can also modify the attributes of model types that you do not own, you cannot export these model types or the changes to their attributes.

You can edit an attribute by doing the following:
- Changing its default value. See Modifying an Attribute's Default Value.
- Changing its other descriptor values. See Modifying an Attribute's Descriptors.

However, before editing an attribute's characteristics, you should verify that derived model types that inherit the attribute will not be adversely affected by the changes.

Important! We recommend recording all changes that you make with the Model Type Editor. Some changes are not migrated when the database is updated to a later version of CA Spectrum. Specifically, if you modify attributes (such as flag settings), the model type hierarchy, or relations and the associated meta-rules in the model types supplied by CA or another vendor, the changes are typically not migrated on database upgrade. You will have to reapply these changes manually. For more information, see Migrate Changes to a New Version of CA Spectrum.

More information:
Modify an Attribute's Default Value (see page 67)
Modify an Attribute's Descriptors (see page 68)
Modify an Attribute's Default Value

An attribute can inherit its default value from a base model type or specify its own value (a process called specialization). In the latter case, all model types derived from the specialized model type inherit the changed attribute value.

To modify the default value of an attribute

1. Set current the model type that has the attribute with the default value you want to change.
2. Click the Hierarchy tab, note any derived model types, and verify that they will not be adversely affected by the change you want to make.
3. Click the Attributes tab, select the attribute from the displayed list, and click (Edit).
   The Edit an Attribute dialog opens.
4. Do one of the following to change the default value:
   ■ If the attribute requires a single value, modify the value as desired.
   ■ If the attribute requires a list of values, click Edit, and modify the values as desired in the Edit List Values dialog. To add a value, click (Add), enter an appropriate Object ID index value (typically, an SNMP object identifier) and enter the value to associate with that index entry.
   The Object ID value is optional, and, if you specify the first one, the Model Type Editor uses that Object ID to generate a default, modifiable Object ID for each subsequent value that you add.
   To delete a value, click (Delete).

   Note: If the Default Value field does not have scroll bars, you can enter a value that fits in the provided area. If there are scroll bars, the field automatically enlarges as needed, and the only limitation is the impact on system performance.
5. Click OK twice.
Modify an Attribute's Descriptors

You can modify two types of attribute descriptors:

**Standard attribute descriptors**

When you modify one of these descriptors in a base model type, all derived (child) model types inherit the change.

**Attribute descriptors you can specialize**

When you modify one of these descriptors in a base model type, derived model types that have been specialized (specify their own values) do not inherit the change, but derived model types that have not been specialized do inherit the change.

If you are the owner of the attribute (that is, the attribute was created using the developer ID that is currently active), you can modify all of an attribute's descriptors *in the originating model type* regardless of whether you are the owner of the model type.

If you are not the owner of the attribute, or if the attribute is inherited, you can specialize a subset of the attribute descriptors. That is, you can modify the values of some of the descriptors at any level of the inheritance hierarchy; you are not limited to modifying the values in the originating model type.

**To edit the descriptors of an attribute**

1. Set current the model type that has the attribute with the descriptor values you want to change.
2. Click the Hierarchy tab, note any derived model types, and verify that the model types will not be adversely affected by the changes you want to make.
3. Click the Attributes tab, select the attribute from the displayed list, and click **(Edit selected Attribute)**.
   The Edit an Attribute dialog opens.
4. Modify the attribute descriptors as needed.
   **Note:** All changes that you make to flag settings are subject to the relationships described in **Flags** (see page 45).
5. Click OK.

**More information:**

- [Model Type Inheritance](#) (see page 17)
- [Attribute Descriptors](#) (see page 17)
Working with Attribute Groups

An attribute group is a logical collection of related attributes in a model type. Groups make working with related attributes easier in the Model Type Editor because they allow you to define and use a user-defined sorting mechanism. You can create groups, assign attributes to them, and then add the Group Name or Group ID as a column header in the table of attributes on the Attributes tab in the Contents panel. This lets you then click the column header to quickly group together and view together all of the attributes within a group.

More information:

Group Name and Group ID (see page 47)

Create an Attribute Group

You can create attribute groups.

Note: When you add an attribute group, you add it to a specific model type, and the group is inherited by derived model types in the same way that other attributes of a model type are inherited. Like for other attributes, if you add an attribute group to a model type that you do not own (its developer ID does not match the one that is currently active), you will not be able to export the group.

To create an attribute group

1. Set current the model type in which you want to create the attribute group.
2. Click the Attributes tab, and double-click any attribute that was created using the developer ID that is current active.
   The Edit an Attribute dialog opens.
3. Under Group, click Edit beside the value for Group ID.
   The Select Group dialog opens displaying a table view and a tree view of all of the attribute groups inherited from base model types or originating in the current model type. The Model Type column displays the originating model type for each group (that is, the model type in which the group was created).
4. Specify whether the new attribute group has a parent group:
   - If you do not want the attribute group to have a parent group, do nothing. This is the default behavior.
   - If you want the attribute group to have a parent group, select the group using either of the following methods:
     - Click the Table View tab, and use the table to select the desired parent group.
       To help you find the group, for Filter, you can enter a text string to filter the list to include only the groups whose names include the string. You can also click any column heading to change the sort order from ascending to descending and vice versa.
     - Click the Tree View tab, and navigate the hierarchical tree of parent groups and child groups to select the desired parent group.

5. Click \(\text{Add}\). The Create Group dialog opens.

6. Enter a name for Group Name
   
   **Note:** Do not enter a name that is being used by another attribute group created using the same developer ID. In addition, use a maximum of 128 characters, and use only numbers, letters, and underscore characters (_).

7. Click OK. The new attribute group is created.

**Modify an Attribute Group**

You can modify the name or parent group of an attribute group in the originating model type, that is, in the model type in which the group was created. You cannot modify the name or parent group of an inherited attribute group.

In addition, to modify an attribute group, you must be the owner of the model type in which the group was created. In other words, the active developer ID is the one that was used to create the model type.

**To modify an attribute group**

1. Set current the model type in which the attribute group was created.

2. Click the Attributes tab, and double-click any attribute that was created using the developer ID that is currently active.
   
   The Edit an Attribute dialog opens.
3. Under Group, click Edit beside the value for Group ID.

The Select Group dialog opens. The dialog displays a table view and a tree view of all of the attribute groups inherited from base model types or originating in the current model type. The Model Type column displays the originating model type for each group (that is, the model type in which the group was created).

4. Select the attribute group to modify using either of the following methods:
   - Click the Table View tab, and use the table to select the desired group.
     To help you find the group, for Filter, you can enter a text string to filter the list to include only the groups whose names include the string. You can also click any column heading to change the sort order from ascending to descending and vice versa.
   - Click the Tree View tab, and navigate the hierarchical tree of parent groups and child groups to select the desired group.

5. Click (Edit).

The Edit Group dialog opens.

6. If you want to change the group name, enter a new name for Group Name.

   **Note:** Do not enter a name that is being used by another attribute group created under the same developer ID. In addition, use a maximum of 128 characters, and use only numbers, letters, and underscore characters (_).

7. If you want to change the group’s parent group, do the following:
   a. Click Edit beside the value for Parent Group ID.
   b. In the Select Parent Group dialog, select a new parent attribute group using the Table View tab or the Tree View tab, and click OK.

8. Click OK.

   Your changes are saved and the Edit Group dialog closes.

---

**Delete Attribute Group**

You can delete an attribute group if the following two conditions are met:

- The group has no assigned attributes. If this is not the case, you can delete the group after you have reassigned the attributes.
- The group has no subgroups. If this is not the case, you can delete the group after you have reassigned or deleted the subgroups.
To delete an attribute group

1. Set current the model type in which the attribute group was created.
2. Click the Attributes tab, and double-click any attribute that was created using the developer ID that is currently active.
   The Edit an Attribute dialog opens.
3. Under Group, click Edit beside the value for Group ID.
   The Select Group dialog opens. The dialog displays a table view and a tree view of all of the attribute groups inherited from base model types or originating in the current model type. The Model Type column displays the originating model type for each group (that is, the model type in which the group was created).
4. Select the attribute group to delete and click (Delete).
   A confirmation dialog opens.
5. Click Yes.
   The attribute group is deleted.
Chapter 4: Working with Relations and Meta-Rules

This section contains the following topics:

About Working with Relations and Meta-Rules (see page 73)
Search for and Display Relations (see page 74)

About Working with Relations and Meta-Rules

The modeling catalog provided with the basic CA Spectrum package contains a number of predefined relations, many of which have associated meta-rules. These relations provide a framework that can replicate most relationships in a network. However, you can create additional relations as needed for your network design.

If you add a new relation, you must also do the following:

■ Create meta-rules in <met> that implement the new relation for specific model types.
■ Add intelligence to the model types so that models of those types react appropriately when they are associated based on the new meta-rules. You must implement the intelligence programmatically using the CORBA API.

As an example, assume that you create the following meta-rule:

User Sends_mail_to User

When the meta-rule is instantiated (for example, when one User model sends mail to a second User model), the first model may need to react to the fact that it (the user that it represents) has sent mail, and the second model may need to react to the fact that it has received mail. In this case, you must add intelligence to the User model type to implement these reactions.

Note: For information about using the CORBA API, see the Development API Reference Guide.

More information:

Create Meta-Rules (see page 77)
Search for and Display Relations

You can search for a relation by filtering the list of all relations in the modeling catalog to include only those that contain the text string you specify. By default, the Model Type Editor examines the supplied string against the names of the relations. However, if you display the Relation ID column in the Navigation panel, it will also examine the string against the relation IDs (handles).

Follow these steps:

1. In the Navigation panel, click the Relations tab.
   
   The names of all of the relations in the modeling catalog are listed.

2. In the list, select the name of the relation that you want to examine.
   
   To locate and select a specific relation, you can take the following steps:

   ■ Enter a text string in the Filter text box in order to filter the list to include only the relations whose names or IDs contain the string. To filter the list by ID, the ID column must be displayed in the table.

   ■ Click the Relation bar at the top of the list to change the alphabetical sorting from ascending to descending or vice versa.

   The selected relation becomes the current relation. The following information about the relation is displayed in the Contents panel:

   Developer ID
   
   Specifies the developer ID that was active when the relation was created.

   Relation Name
   
   Specifies the name of the relation.

   Relation ID
   
   Specifies the ID (handle) that is assigned to the relation.
   
   A handle is never reused even if you delete a relation.

   Relation Type
   
   Specifies the type of relation, either One-to-Many or Many-to-Many.

   Meta-Rules
   
   Specifies the list of meta-rules that apply the relation to specific model types, thereby defining how the model types can interact with one another.
3. To filter the list of meta-rules to include only those for a specific model type, enter the full or partial name of the model type in the Filter text box:

More information:

- Sorting and Filtering Lists (see page 35)
- Add and Remove Columns from Tables (see page 36)
Create Relations

You can create relations.

To create a relation

1. Click the Relations tab and click (Create Relation) in the Navigation panel.
   The Create Relation dialog opens.
2. Enter a name for the relation.
   The name should be a maximum length of 31 characters and should include alphanumeric characters and underscores but not spaces or punctuation.
   **Important!** The relation name does not need to be unique across the modeling catalog, but you should enter a name that is unique across the relations created under a given developer ID. In addition, while it is allowed, reusing a relation name is not recommended.
3. Select the type of relation to create:
   One-to-Many
   Relations of this type relate one model type to many model types.
   Many-to-Many
   Relations of this type relate many model types to many model types.
   **Note:** Once you create the relation, you cannot change the relation type. To specify a different relation type, you must delete the relation and create a new one.
4. Click OK.
   The relation is created and assigned a relation ID, and its information is displayed in the Contents panel.
   **Note:** A handle is never reused even if you delete the relation.
5. Create one or more meta-rules that use the relation.
6. Add intelligence to the relevant model types so that models of those types react appropriately when they are associated based on the new meta-rules. You must implement the intelligence programmatically using the CORBA API.
   **Note:** For information on using the CORBA API, see the Development API Reference Guide.

More information:

[Create Meta-Rules] (see page 77)
Relation Meta-Rules

Each relation normally has one or more meta-rules, each of which applies the relation to specific model types. Many of the model types in the core database have relations that are supplied without meta-rules; these function as "placeholders" for which you can supply customized meta-rules that are appropriate for your network design.

When a new model type is derived from one that is specified as a member of a meta-rule in a relation, the meta-rule automatically applies to the derived model type. For example, the modeling catalog provided with the basic CA Spectrum package has a relation called Contains and two model types named Room and Device. One of the meta-rules for the Contains relation is the following:

Room Contains Device

The modeling catalog also contains a Workstation model type that is derived from Device. As a result, the "Room Contains Device" also applies to the Workstation model type in the form of "Room Contains Workstation." You do not have to explicitly create a meta-rule that includes the Workstation model type.

About Creating Meta-Rules for General Model Types

Whenever possible, create meta-rules for general model types (model types near the top of the model type hierarchy) in order to maximize their application and reduce the need for more specific meta-rules. The more general the model type that contains the meta-rules, the more derived model types it has, and, therefore, the more model types inherit its meta-rules.

As an example, the Device model type is a general model type that is used in the following meta-rule for the Is_Adjacent_to relation:

Device Is_Adjacent_to Device

Consequently, any model type derived from Device or one of its descendants can be adjacent to any other model type derived from Device or one of its descendants.

Create Meta-Rules

You can create a meta-rule without restriction if you own the corresponding relation, that is, the relation was created using the developer ID that is currently active.
If the relation was created using a different developer ID, you can still create a meta-rule for it if at least one of the model types used in the rule was created using the developer ID that is currently active.

**To create a meta-rule**

1. Search for and display the relation for which to create the meta-rule.
   - The relation is displayed in the Contents panel.

2. On the Relation tab in the Contents panel, click ![Create a new Meta-Rule](Create a new Meta-Rule). The Create Meta Rule dialog opens.

3. In the list of model types on the left, select the antecedent (left) model type to include in the meta-rule.
   - **Note:** To help you locate and select a specific model type, you can enter a text string in the corresponding Filter text box in order to filter the list accordingly.

4. In the list of model types on the right, select the predicate (right) model type to include in the meta-rule, and click OK.
   - The new meta-rule is added to the list of meta-rules for the current relation; it is inserted into the list alphabetically according to the antecedent model type.
   - **Note:** You now need to add intelligence to the model types so that models of those types react appropriately when they are associated based on the new meta-rule. You must implement the intelligence programmatically using the CORBA API. For more information, see the *Development API Reference Guide*.

**More information:**

*Sorting and Filtering Lists* (see page 35)
*Search for and Display Relations* (see page 74)

**Delete Relations**

You can delete a relation if the following two conditions are met:

- The relation was created using the developer ID that is currently active.
- The relation does not have any associated meta-rules. If this is not the case, first delete the meta-rules.
To delete a relation
1. Search for and display the relation to delete.
2. On the Relations tab in the Navigation panel, select the relation, and click (Delete selected relation).
   A confirmation dialog opens.
3. Click Yes.
   The relation is deleted.

More information:
Search for and Display Relations (see page 74)
Delete Meta–Rules (see page 79)

Delete Meta–Rules

You can delete a meta-rule without restriction if you own the corresponding relation, that is, the relation was created using the developer ID that is currently active.

If the relation was created using a different developer ID, you can still delete the meta-rule if at least one of the model types used in the rule was created using the developer ID that is currently active.

To delete a meta-rule
1. Search for and display the relation that contains the meta-rule to delete.
2. On the Relation tab in the Contents panel, select the meta-rule and click (Delete selected meta-rule).
   A confirmation dialog opens.
3. Click Yes.
   The meta-rule is deleted.

More information:
Search for and Display Relations (see page 74)
Chapter 5: Importing and Exporting Model Types

This section contains the following topics:

- About Importing and Exporting Model Types (see page 81)
- Import Model Types Using the Model Type Editor (see page 81)
- Export Model Types Using the Model Type Editor (see page 83)
- Import and Export Model Types Using dbtool (see page 85)
- Send an Exported Catalog to a File or Printer (see page 86)

About Importing and Exporting Model Types

As your network grows, you may need to add new management modules and model types to your database to support additional types of devices. Moreover, you may want to add these new model types to your database without installing a completely new database, so you can keep intact all of the model types you have already created or modified. CA Spectrum provides two database utilities that lets you to import and export model types:

- **Import and export commands in the Model Type Editor**: These commands let you import and export model types from the *working catalog* for the current session. Because the commands do not operate on the permanent catalog in the SpectroSERVER database, you can make an explicit decision after an import as to whether to commit or discard the changes. Since you can only import or export one catalog file at a time, use these commands when you have only one or a few files to process.

- **A command-line utility named dbtool**: This command-line utility program lets you import and export model types from the *permanent catalog* in the SpectroSERVER database. Because you can specify multiple files as command-line arguments, use this utility to batch process a set of files.

In both cases, the transfer vehicle is a binary export file that has a .e extension. These files are referred to as *catalogs*.

Import Model Types Using the Model Type Editor

You can use the Model Type Editor to import model types, attributes, relations, and meta-rules into the SpectroSERVER database. The modeling catalog objects must be defined in a catalog file (.e file) that was created using the export feature in the Model Type Editor or the dbtool command-line utility.
Import Constraints

**Important!** You can only import modeling catalog objects that are stored in a compatible CA Spectrum database (a database that is running the same version of CA Spectrum as the destination database). For information about updating a CA Spectrum database, see the *Installation Guide*.

New model types are imported into the modeling catalog according to the following constraints:

- If the catalog file to import contains a model type that does not exist in the destination database, the model type is imported according to the rest of the constraints described in this section.

- All of the base model types for a new model type must already exist in the destination database. If they do not, you must import them before importing the new model type. Typically, the "core catalog" contains these prerequisite model types, and documentation accompanying any new catalog informs you of any such dependencies.

  If a base model type for a new model type does not exist, the import process is terminated. In this situation, you can identify the missing model type in the CA Spectrum Control Panel. You must then import a catalog file that contains the missing base model type, and then reinitiate the import process that was terminated.

- If the catalog file to import contains a model type that already exists in the destination database, the existing version is modified to match the version to import, for example:
  - The model type's derivation is updated
  - New attributes are added
  - Existing attributes are updated
  - Attributes in the existing model type that have the same developer ID as the existing model type are deleted if they originate in the existing model type but are not included in the version being imported

- If an error occurs due to an inability to write to the destination database or due to insufficient memory or system resources, the import process is terminated, and you are notified with an appropriate error message. Because this situation can leave the database in an incomplete or corrupted state, you should always back up the database before beginning an import operation.
Import Model Types

When you import model types using the Model Type Editor, the model types and associated catalog objects are imported into the working catalog. This lets you perform the import and then make an explicit decision as to whether to discard the changes or permanently commit them to the SpectroSERVER database.

To import the contents of a catalog
1. Back up the SpectroSERVER database into which you are importing a catalog.
   
   **Note:** For information about how to back up the database using the database utilities provided with CA Spectrum, see the *Database Management Guide*.

2. Select File, Import Model Types.
   
   The Open dialog opens.

3. Navigate to the catalog file (.e file) you want to import, select the file, and click Open.
   
   The modeling catalog information in the selected file is imported into the working catalog for the current session. At this point, you can manually commit the changes to the permanent catalog in the SpectroSERVER database, or you can do so when you are prompted when you exit the application. Alternatively, you can exit the application without committing the changes to discard them.

More information:

[Commit Changes to the SpectroSERVER Database](#) (see page 34)

Export Model Types Using the Model Type Editor

You can use the Model Type Editor to export a specific list of model types from the SpectroSERVER database to a catalog file. A catalog file is a .e file, and it is sometimes referred to as a CA Spectrum database export file.

When you are exporting model types and associated catalog objects, bear the following in mind:

- The export feature in the Model Type Editor exports the working catalog that is stored in memory. As a result, the resulting export file includes any changes you have made to model types and associated objects during the current session even if you have not yet committed those changes to the SpectroSERVER database.

To export the permanent catalog only, you can commit the changes and use the export feature in the Model Type Editor, or you can use the dbtool command-line utility instead.
The .e file produced by the export process contains the following information:

- The attribute descriptors that originated in the model type being exported.
- The attribute descriptors that have been specialized (for example, by specifying a default value to override an inherited one).
- The relations and associated meta-rules in which the model type and/or any ancestor model types participate as an antecedent or a predicate.

You can only export the model types, attributes, and relations (including their associated meta-rules) that were created using the active developer ID.

To export one or more model types to a catalog file

1. Identify the model types to export.

   Typically, the list of model types to export should include any base model types that are required by the model types being exported and that do not exist in the destination database.

   However, in general practice, dependencies normally are limited to certain commonly-used base model types that are contained in one or more "core" catalogs that are included as part of the basic CA Spectrum system. These core catalogs are a part of every installation, as shown in the following illustration.

   ![Diagram of model types and core catalogs]

2. Select File, Export Model Types.

   The Model Type Export dialog opens.

   By default, all unmodified model types that were created using the active developer ID are initially displayed in the Model Types column on the left.

   In addition, all model types that were created using the active developer ID and that have been modified during the current session are initially displayed in the Model Types to Export column on the right. Modified model types are those you have changed regardless of whether the changes are currently in the working catalog or committed to the database. Note that once you export a model type, it is no longer identified as modified because you have saved it to a catalog, in this case, to a CA Spectrum database export file.
3. Move the model types that you want to export to the Model Types to Export list.
   To search for the model types to export, in the Filter text box, you can enter a text string to filter the list to include only the model types with names that contain or match the string. The filter is not case-sensitive.

   To move all of the model types available for export to the Model Types to Export list, click \[\text{Move all}\]. To move a single model type to the list, double-click the model type name, or select the model type and click \[\text{Move}\].

   Similarly, you can remove model types from the export list by double-clicking them or using the corresponding left-arrow buttons.

4. Click Browse, and in the Save dialog, navigate to the folder in which to save the catalog file.

   **Important!** By default, the Save dialog opens to the CA Spectrum database (modeling catalog) directory. However, it is recommended that you save exported catalog files in a directory outside of the CA Spectrum installation area in order to prevent the loss of the files during a CA Spectrum update process.

5. Enter a name for the file (you do not need to include the .e extension) or select an existing .e file to overwrite, and click Save.

6. In the Model Type Export dialog, click OK.

   The model types are exported to the specified catalog file.

---

**Import and Export Model Types Using dbtool**

The CA Spectrum command-line utility program named dbtool can import or export model types from the permanent modeling catalog in the SpectroSERVER database. The dbtool utility lets you specify multiple files as command-line arguments. Therefore, it is a better tool to use than the import and export features of the Model Type Editor for batch processing a set of files.

When you use dbtool, you can export only the model types that you "own," that is, types that were created using the currently active developer ID. If a model type is not owned, an error message describes the relevant model types, and the export process is terminated.
Run the dbtool utility from the directory that contains the SpectroSERVER database that is used in the import or export. In addition, while the import or export is underway, no other program or process (for example, a VNM or the Model Type Editor) can access the database. Keep in mind that while CA-developed applications automatically lock out other CA-developed applications, third-party applications may not. The competition can result in database corruption.

**Note:** For more information about running dbtool, see the *Database Management Guide*. This guide also contains information about how to back up the database using the CA Spectrum database utilities. Perform the backup before you perform an import.

---

**Send an Exported Catalog to a File or Printer**

CA Spectrum includes a command-line utility program named *dbtool* that you can use to send or "dump" the contents of a catalog file (.e file) to a file or printer. You can use this tool to store or print the contents of a catalog file that was created using the export feature of the Model Type Editor or using dbtool itself.

Before running dbtool, you must shut down the SpectroSERVER and any other program that accesses the SpectroSERVER database.

**Note:** For information about running the dbtool utility, see the *Database Management Guide*. 
About Running Reports on Model Types and Relations

CA Spectrum includes a command-line utility program named reports that you can use to display, print, or export (to a file) information about the model types and relations in the modeling catalog.

You must run the reports utility from the directory that contains the SpectroSERVER database that contains the data you want to access. While the report is being generated, no other program or process (for example, a VNM or the Model Type Editor) can access the database.

Note: For information about running the reports utility, see the Database Management Guide.
Index

A
adding attributes to the SpectroSERVER database • 60
adding Developer ID to database • 27
adding vendor information • 27
attribute collapsing, defined • 25
attribute descriptors
  attribute ID (handle) • 44
  attribute name • 44
defined • 17
descriptors you can specialize • 17
developer ID • 38
flags • 45
standard • 17, 43
type • 44
attribute flags • 45
attribute groups
  creating • 69
defined • 69
deleting • 71
modifying • 70
attribute ID (handle) • 44
attributes
  adding to the SpectroSERVER database • 60
  creating an Attribute Group • 69
  editing • 66
  name • 44
  removing from a model type • 65
  searching for • 53

B
backing up your database • 82
base model types
  adding to a model type • 58
  identifying for a model type • 56
  import process and • 82
  removing from a model type • 58

C
catalogs

database
  access • 29
    adding attributes to • 60
  Database attribute flag • 48
  dbtool • 85, 86
  Default Value attribute descriptor • 48, 67
  dependencies • 82
  Derivable model type flag • 40
  derivation links • 82
derived model types • 39
developer ID
  access privileges • 28
  activating • 27
default • 27
declared • 38
  exporting model types and • 83
  importing MIBs and • 60
  model type ID and • 39
  model type name and • 38
  registered • 27

e
error messages • 82
export file • 81
exporting model types • 83
extended flags • 48

F
filtering lists • 35

G
GnSNMPDev management module • 9
group ID • 47
group name • 47

H
handles, model type • 39

exporting to • 83
importing into database • 82
committing changes to the database • 34
CORBA API, usage • 73
I
IDs, model type • 39
importing
catalog into database • 82
MIB file • 60
model type • 81
Instantiable model type flag • 40

L
Logged attribute flag • 48

M
Memory attribute flag • 48
menu bar • 34
meta-rules, defined • 15
MIB file, importing • 60
model • 14
model type
attributes and • 10
attributes of • 37
base • 10, 39
creating • 53
custom attributes • 42
defined • 10
deleting • 54
derived • 10, 39
exporting • 83
exporting with dbtool • 85
flags • 40
general • 77
hierarchy • 60
ID • 39
importing • 81
importing with dbtool • 85
name • 38
searching for • 52
setting flags • 62
Model Type Editor
committing changes to the database • 34
defined • 9
exiting • 36
starting • 31
user interface • 33
MTE (Model Type Editor) • 9

N
No Destroy model type flag • 40

noSuchName • 48

O
OID Prefix • 50
OID Reference • 50
Orig Model Type field • 25

P
permanent catalog • 34
Polled attribute flag • 48
polling group • 51
Preserve Value attribute flag • 30

R
relations
creating • 76
defined • 15
deleting • 78
many-to-many • 15
one-to-many • 15
searching for • 74
reports, running • 87
Required • 40
restraints on importing model types • 82
rules
creating • 77
creating for general model types • 77
defined • 77
deleting • 79

S
Save to Permanent Catalog option • 34
SMIv1 MIBs • 60
SMIv2 MIBs • 60
sorting lists • 35
SSdbload • 27, 29
SSdbsave • 29

T
tables, adding and removing columns • 36
transferring model types • 83
type attribute descriptor • 44

U
Unique model type flag • 40
unsupported_attr_poll_interval • 48
V
Value Defined In field • 25
Visible model type flag • 40

W
working catalog • 34