Management Tools

This chapter contains these sections:

Management Tools

Cisco Application Centric Infrastructure (ACI) tools help fabric administrators, network engineers, and developers to develop, configure, debug, and automate the deployment of tenants and applications.

About the Management GUI

The following management GUI features provide access to the fabric and its components (leaves and spines):

- Based on universal web standards (HTML5). No installers or plugins are required.
- Access to monitoring (statistics, faults, events, audit logs), operational and configuration data.
- Access to the APIC and spine and leaf switches through a single sign-on mechanism.
- Communication with the APIC using the same RESTful APIs that are available to third parties.

About the CLI

The CLI features an operational and configuration interface to the APIC, leaf, and spine switches:

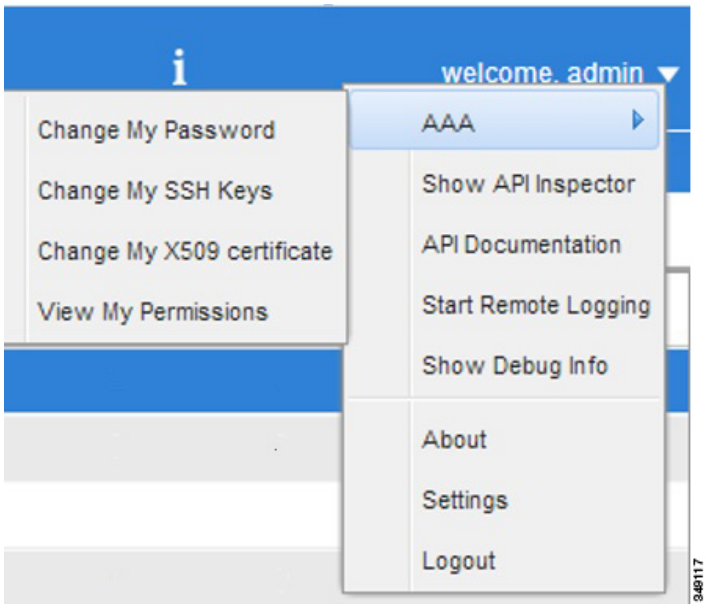
- Implemented from the ground up in Python; can switch between the Python interpreter and CLI
- Plugin architecture for extensibility

- Virtual Routing and Forwarding (VRF)-based access to monitoring, operation, and configuration data
- Automation through Python commands or batch scripting

User Login Menu Options

The user login drop-down menu provides several configuration, diagnostic, reference, and preference options. The figure below shows this drop-down menu.

Figure 1: User Login Menu Options



The options include the following:

- AAA options for changing the user password, SSH Keys, X509 Ce.jpgicate, and viewing the permissions of the logged-on user.

Note The ACI fabric must be configured with an active Network Time Protocol (NTP) policy to assure that the system clocks on all devices are correct. Otherwise, a ce.jpgicate could be rejected on nodes with out-of-sync time.

- Show API Inspector opens the API Inspector.
- API Documentation opens the Management Information Model reference.
- Remote Logging.
- Debug information.
- About the current version number of the software.
- Settings preferences for using the GUI.

- Logout to exit the system.

Customizing the GUI and CLI Banners

GUI and CLI banners can be in the Admin > AAA > Security management section of the GUI. The CLI banner displays before user login authentication. The CLI banner is a text based string printed as-is to the console. The GUI banner displays before user login authentication. The GUI banner is a URL. The URL must allow the being placed in an iFrame. If the URL `x-frame-option` is set to `deny` or `sameorigin`, the URL will not appear before user login authentication.

REST API

About the REST API

The Application Policy Infrastructure Controller (APIC) REST API is a programmatic interface that uses REST architecture. The API accepts and returns HTTP (not enabled by default) or HTTPS messages that contain JavaScript Object Notation (JSON) or Extensible Markup Language (XML) documents. You can use any programming language to generate the messages and the JSON or XML documents that contain the API methods or Managed Object (MO) descriptions.

The REST API is the interface into the management information tree (MIT) and allows manipulation of the object model state. The same REST interface is used by the APIC CLI, GUI, and SDK, so that whenever information is displayed, it is read through the REST API, and when configuration changes are made, they are written through the REST API. The REST API also provides an interface through which other information can be retrieved, including statistics, faults, and audit events. It even provides a means of subscribing to push-based event notification, so that when a change occurs in the MIT, an event can be sent through a web socket.

Standard REST methods are supported on the API, which includes POST, GET, and DELETE operations through HTTP. The POST and DELETE methods are idempotent, meaning that there is no additional effect if they are called more than once with the same input parameters. The GET method is nullipotent, meaning that it can be called zero or more times without making any changes (or that it is a read-only operation).

Payloads to and from the REST interface can be encapsulated through either XML or JSON encoding. In the case of XML, the encoding operation is simple: the element tag is the name of the package and class, and any properties of that object are specified as attributes of that element. Containment is defined by creating child elements.

For JSON, encoding requires definition of certain entities to reflect the tree-based hierarchy; however, the definition is repeated at all levels of the tree, so it is fairly simple to implement after it is initially understood.

- All objects are described as JSON dictionaries, in which the key is the name of the package and class. The value is another nested dictionary with two keys: attribute and children.
- The attribute key contains a further nested dictionary describing key-value pairs that define attributes on the object.
- The children key contains a list that defines all the child objects. The children in this list are dictionaries containing any nested objects, which are defined as described here.

Authentication

REST API username- and password-based authentication uses a special subset of request Universal Resource Identifiers (URIs), including **aaaLogin**, **aaaLogout**, and **aaaRefresh** as the DN targets of a POST operation. Their payloads contain a simple XML or JSON payload containing the MO representation of an **aaaUser** object with the attribute name and **pwd** defining the username and password: for example, **<aaaUser name='admin' pwd='password'/>**. The response to the POST operation will contain an authentication token as both a Set-Cookie header and an attribute to the **aaaLogin** object in the response named token, for which the XPath is **/imdata/aaaLogin/@token** if the encoding is XML. Subsequent operations on the REST API can use this token value as a cookie named **APIC-cookie** to authenticate future requests.

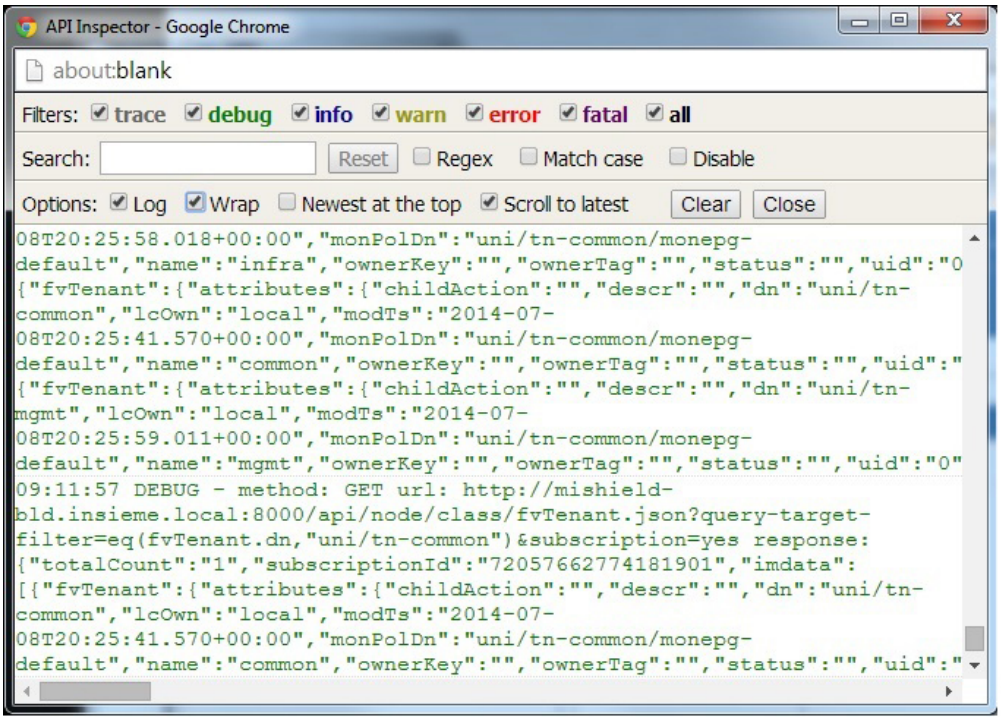
Subscription

The REST API supports the subscription to one or more MOs during your active API session. When any MO is created, changed, or deleted because of a user- or system-initiated action, an event is generated. If the event changes the data on any of the active subscribed queries, the APIC will send out a notification to the API client that created the subscription.

API Inspector

The API Inspector provides a real-time display of REST API commands that the APIC processes to perform GUI interactions. The figure below shows REST API commands that the API Inspector displays upon navigating to the main tenant section of the GUI.

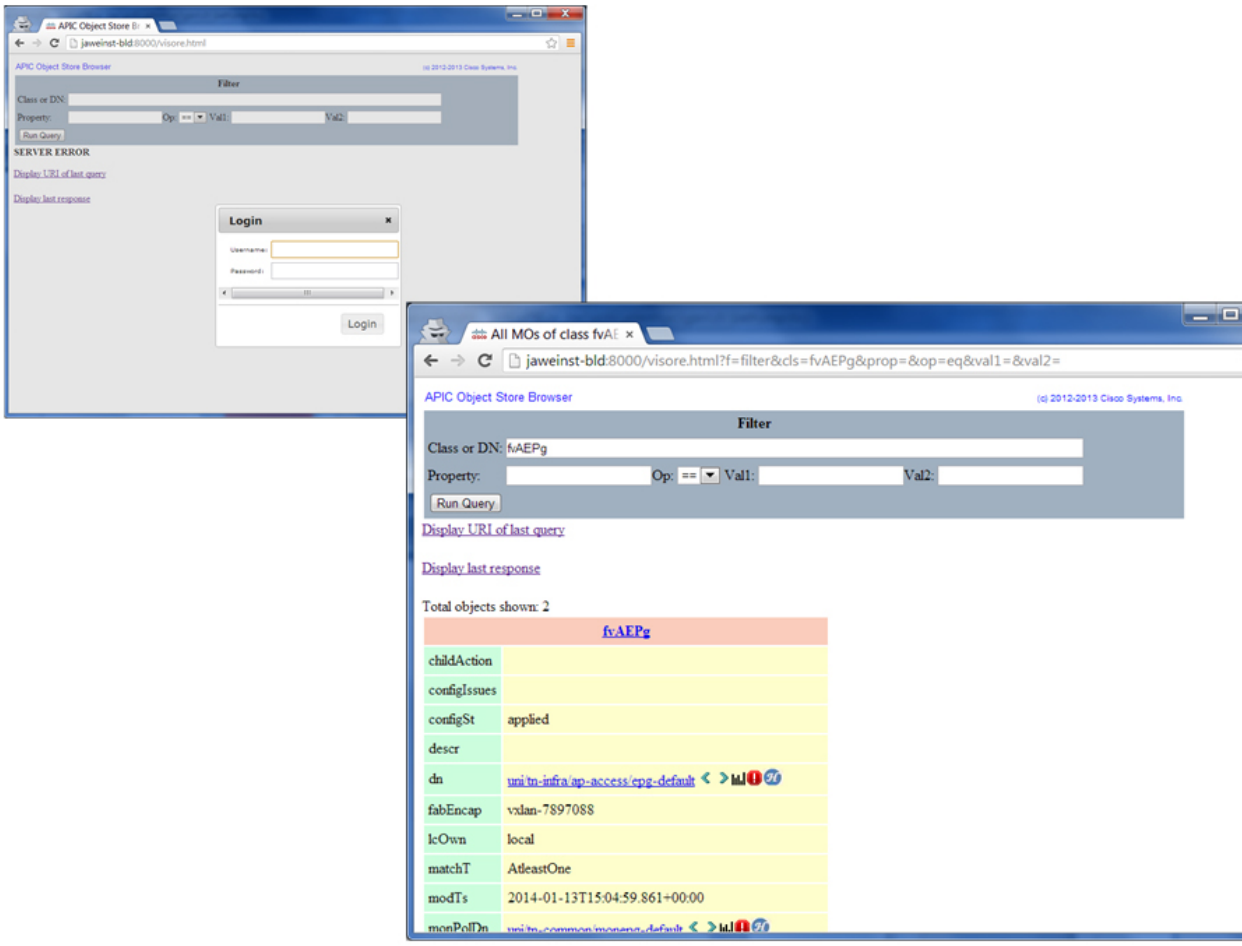
Figure 2: API Inspector



Visore Managed Object Viewer

Visore is a read-only management information tree (MIT) browser as shown in the figure below. It enables distinguished name (DN) and class queries with optional filters.

Figure 3: Visore MO Viewer



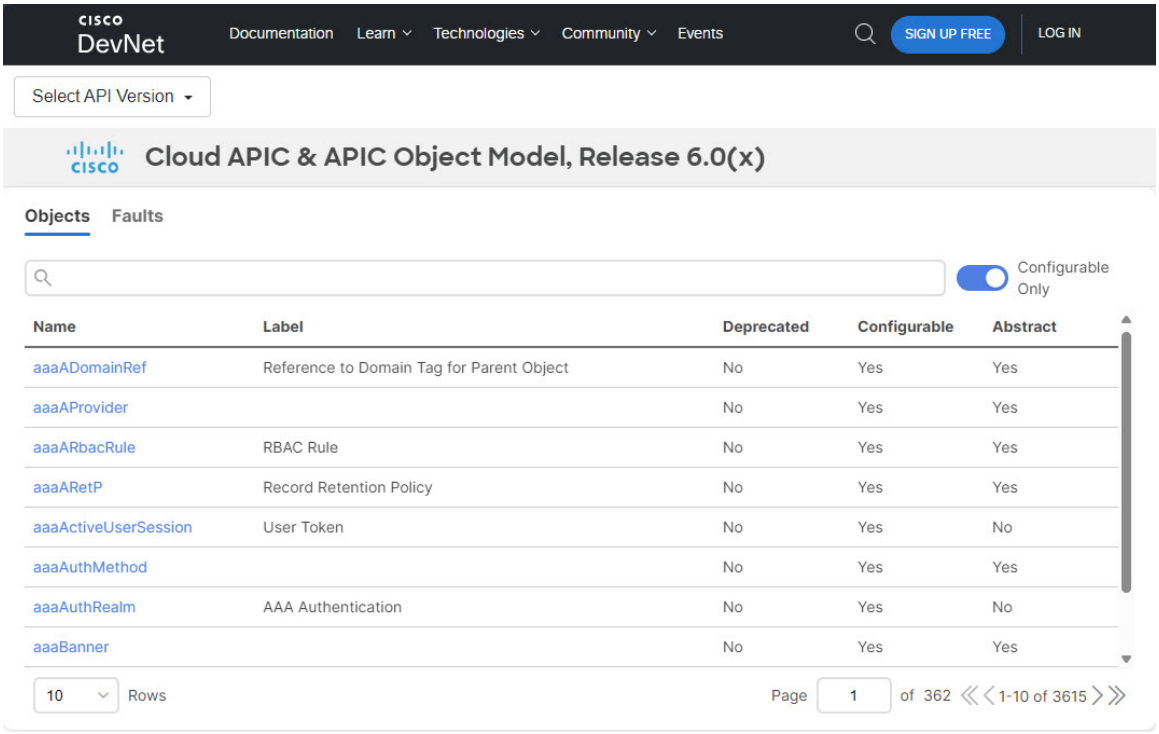
The Visore managed object viewer is at this location: `http(s)://host[:port]/visore.html`

Management Information Model Reference

The Management Information Model (MIM) contains all of the managed objects in the system and their properties. For details, see the *Cisco APIC Management Information Model Reference Guide*.

See the following figure for an example of how an administrator can use the MIM to research an object in the MIT in the Cisco APIC 6.0 releases.

Figure 4: MIM Reference for the Cisco APIC 6.0 Releases

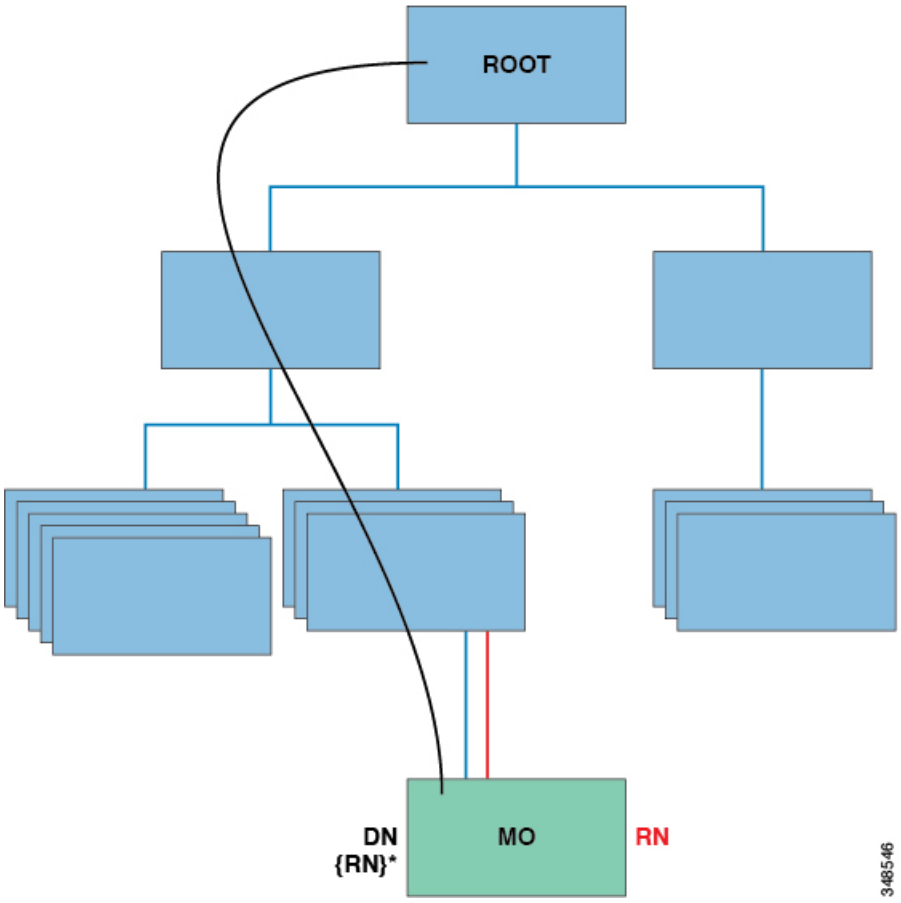


Locating Objects in the MIT

The Cisco ACI uses an information-model-based architecture (management information tree [MIT]) in which the model describes all the information that can be controlled by a management process. Object instances are referred to as managed objects (MOs).

The following figure shows the distinguished name, which uniquely represents any given MO instance, and the relative name, which represents the MO locally underneath its parent MO. All objects in the MIT exist under the root object.

Figure 5: MO Distinguished and Relative Names



Every MO in the system can be identified by a unique distinguished name (DN). This approach allows the object to be referred to globally. In addition to its distinguished name, each object can be referred to by its relative name (RN). The relative name identifies an object relative to its parent object. Any given object's distinguished name is derived from its own relative name that is appended to its parent object's distinguished name.

A DN is a sequence of relative names that uniquely identifies an object:

```
dn = {rn}/{rn}/{rn}/{rn}
```

```
dn =”sys/ch/lcslot-1/lc/leafport-1”
```

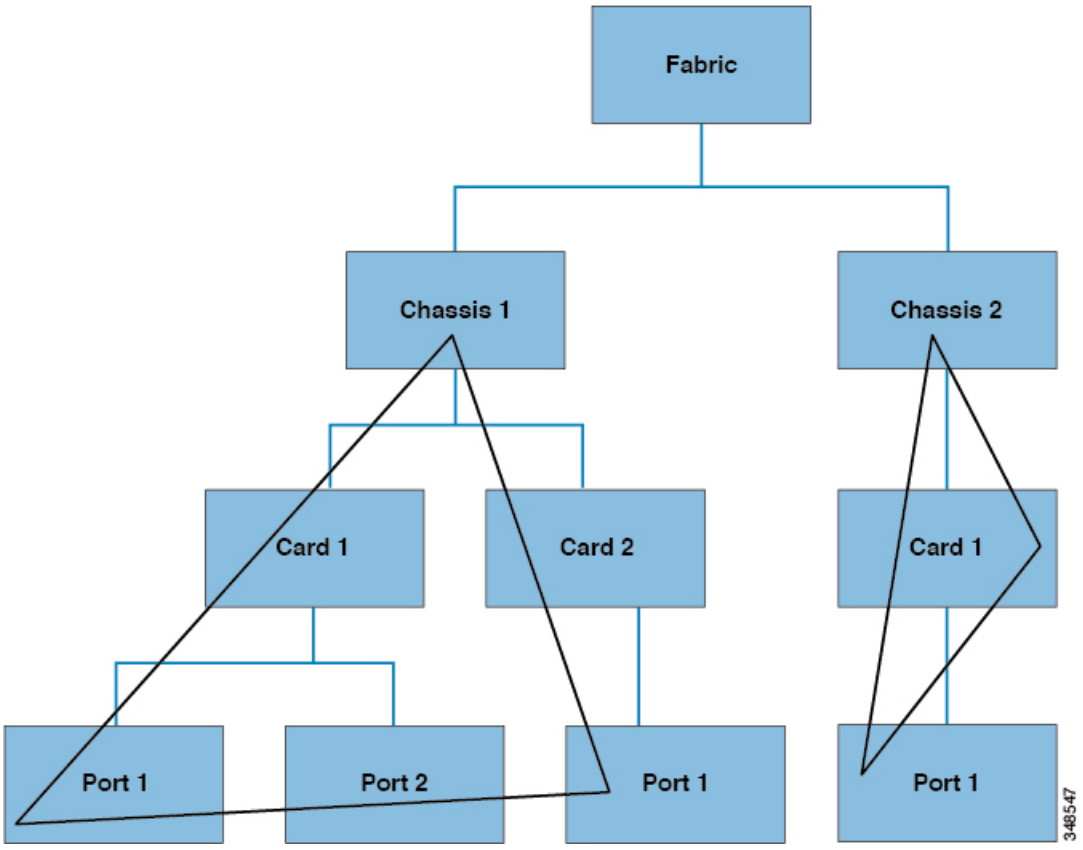
Distinguished names are directly mapped to URLs. Either the relative name or the distinguished name can be used to access an object, depending on the current location in the MIT.

Because of the hierarchical nature of the tree and the attribute system used to identify object classes, the tree can be queried in several ways for obtaining managed object information. Queries can be performed on an object itself through its distinguished name, on a class of objects such as a switch chassis, or on a tree-level to discover all members of an object.

Tree-Level Queries

The following figure shows two chassis that are queried at the tree level.

Figure 6: Tree-Level Queries

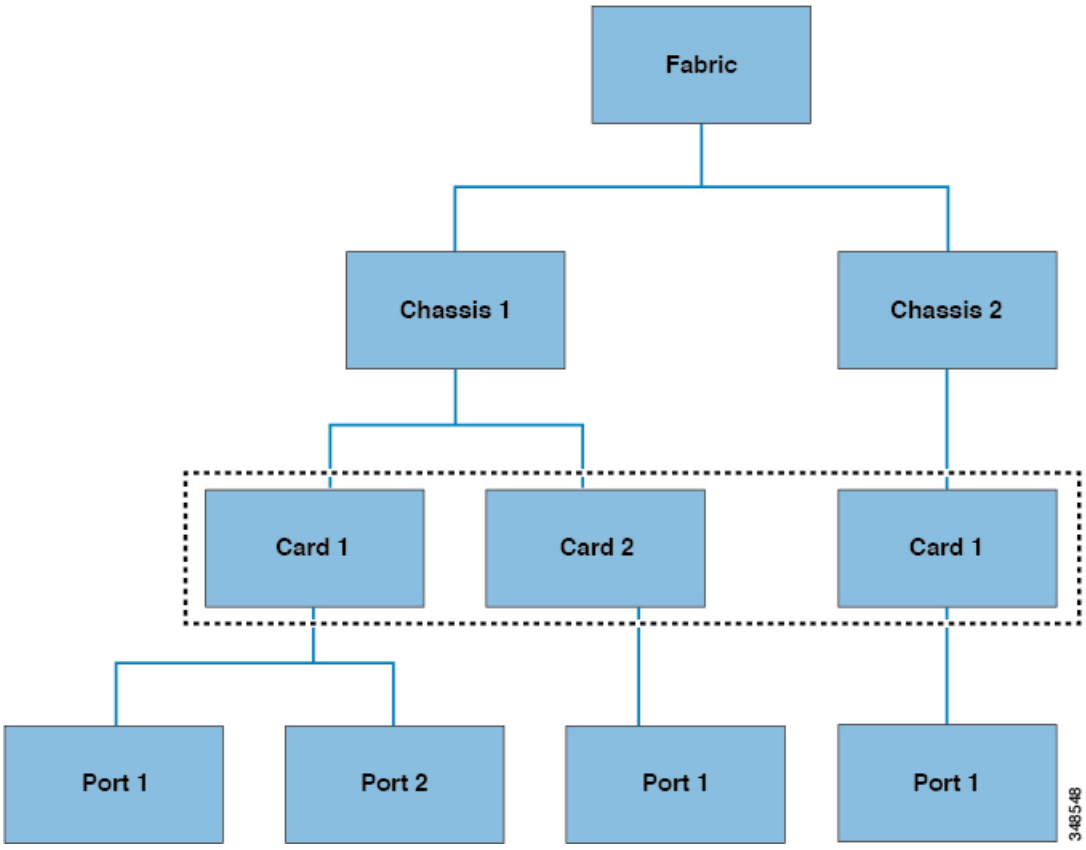


Both queries return the referenced object and its child objects. This approach is useful for discovering the components of a larger system. In this example, the query discovers the cards and ports of a given switch chassis.

Class-Level Queries

The following figure shows the second query type: the class-level query.

Figure 7: Class-Level Queries

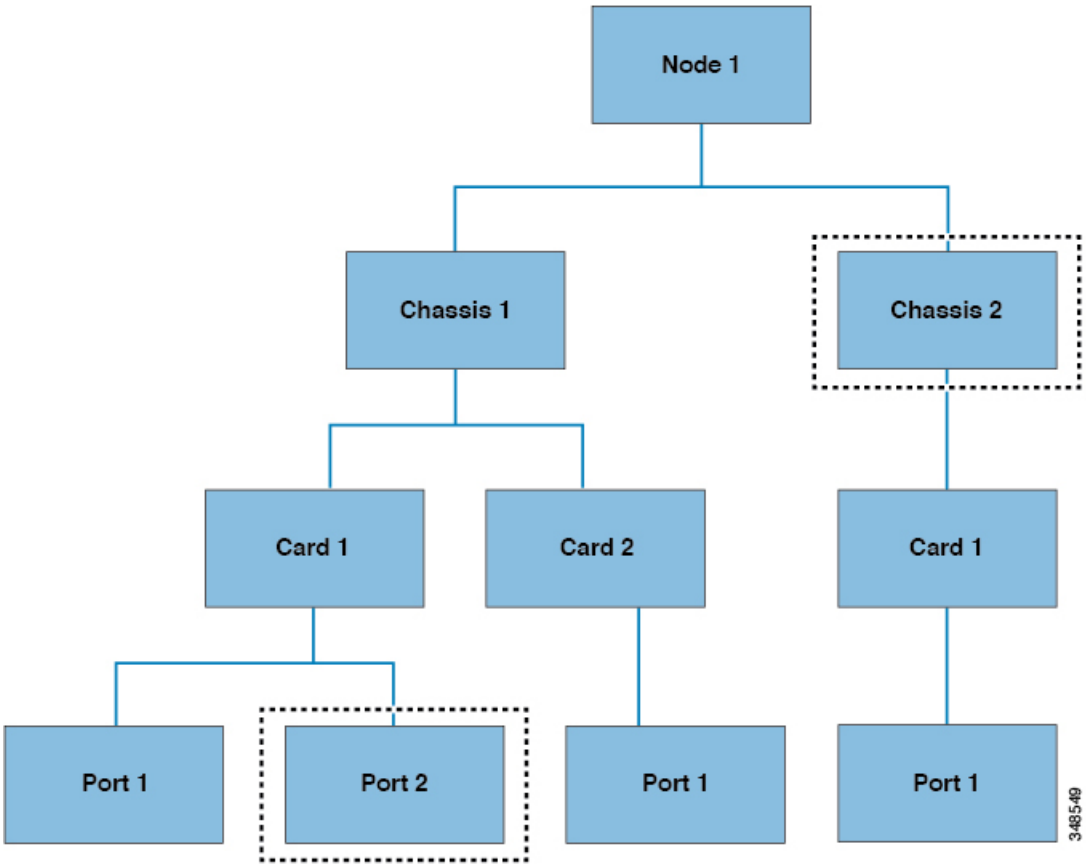


Class-level queries return all the objects of a given class. This approach is useful for discovering all the objects of a certain type that are available in the MIT. In this example, the class used is Cards, which returns all the objects of type Cards.

Object-Level Queries

The third query type is an object-level query. In an object-level query a distinguished name is used to return a specific object. The figure below shows two object-level queries: for Node 1 in Chassis 2, and one for Node 1 in Chassis 1 in Card 1 in Port 2.

Figure 8: Object-Level Queries

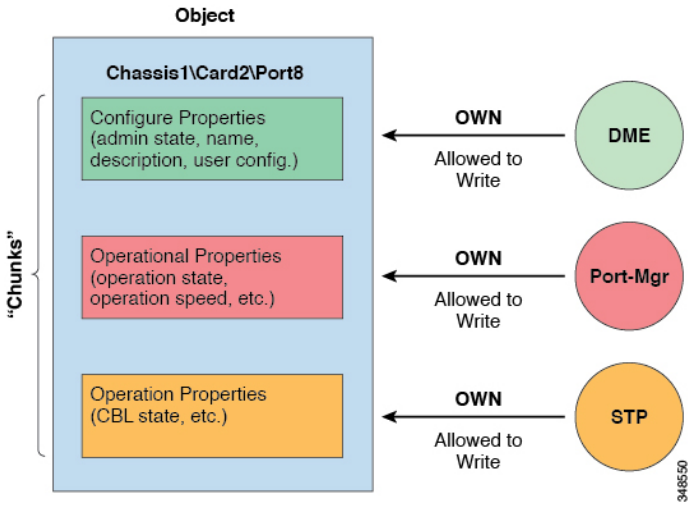


For all MIT queries, an administrator can optionally return the entire subtree or a partial subtree. Additionally, the role-based access control (RBAC) mechanism in the system dictates which objects are returned; only the objects that the user has rights to view will ever be returned.

Managed-Object Properties

Managed objects in the Cisco ACI contain properties that define the managed object. Properties in a managed object are divided into chunks that are managed by processes in the operating system. Any object can have several processes that access it. All these properties together are compiled at runtime and are presented to the user as a single object. The following figure shows an example of this relationship.

Figure 9: Managed Object Properties



The example object has three processes that write to property chunks that are in the object. The data management engine (DME), which is the interface between the Cisco APIC (the user) and the object, the port manager, which handles port configuration, and the spanning tree protocol (STP) all interact with chunks of this object. The APIC presents the object to the user as a single entity compiled at runtime.

Accessing the Object Data Through REST Interfaces

REST is a software architecture style for distributed systems such as the World Wide Web. REST has increasingly displaced other design models such as Simple Object Access Protocol (SOAP) and Web Services Description Language (WSDL) due to its simpler style. The Cisco APIC supports REST interfaces for programmatic access to the entire Cisco ACI solution.

The object-based information model of Cisco ACI makes it a very good fit for REST interfaces: URLs and URIs map directly to distinguished names that identify objects on the MIT, and any data on the MIT can be described as a self-contained structured text tree document that is encoded in XML or JSON. The objects have parent-child relationships that are identified using distinguished names and properties, which are read and modified by a set of create, read, update, and delete (CRUD) operations.

Objects can be accessed at their well-defined address, their REST URLs, using standard HTTP commands for retrieval and manipulation of Cisco APIC object data. The URL format used can be represented as follows:

```
<system>/api/[mo|class]/[dn|class][:method].[xml|json]?{options}
```

The various building blocks of the preceding URL are as follows:

- `system`: System identifier; an IP address or DNS-resolvable hostname
- `mo | class`: Indication of whether this is a MO in the MIT, or class-level query
- `class`: MO class (as specified in the information model) of the objects queried; the class name is represented as `<pkgName><ManagedObjectClassName>`
- `dn`: Distinguished name (unique hierarchical name of the object in the MIT) of the object queried
- `method`: Optional indication of the method being invoked on the object; applies only to HTTP POST requests
- `xml | json`: Encoding format

- `options`: Query options, filters, and arguments

With the capability to address and access an individual object or a class of objects with the REST URL, an administrator can achieve complete programmatic access to the entire object tree and to the entire system.

The following are REST query examples:

- Find all EPGs and their faults under tenant solar.

  ```
  http://192.168.10.1:7580/api/mo/uni/tn-solar.xml?query-target=subtree&target-subtree-class=fvAEPg&rsp-subtree-include=faults
  ```

- Filtered EPG query

  ```
  http://192.168.10.1:7580/api/class/fvAEPg.xml?query-target-filter=eq(fvAEPg.fabEncap,%20"vxlan-12780288")
  ```

Configuration Export/Import

All APIC policies and configuration data can be exported to create backups. This is configurable via an export policy that allows either scheduled or immediate backups to a remote server. Scheduled backups can be configured to execute periodic or recurring backup jobs. By default, all policies and tenants are backed up, but the administrator can optionally specify only a specific subtree of the management information tree. Backups can be imported into the APIC through an import policy, which allows the system to be restored to a previous configuration.

Configuration Database Sharding

The APIC cluster uses a large database technology called sharding. This technology provides scalability and reliability to the data sets generated and processed by the APIC. The data for APIC configurations is partitioned into logically bounded subsets called shards which are analogous to database shards. A shard is a unit of data management, and the APIC manages shards in the following ways:

- Each shard has three replicas.
- Shards are evenly distributed across the appliances that comprise the APIC cluster.

One or more shards are located on each APIC appliance. The shard data assignments are based on a predetermined hash function, and a static shard layout determines the assignment of shards to appliances.

Configuration File Encryption

As of release 1.1(2), the secure properties of APIC configuration files can be encrypted by enabling AES-256 encryption. AES encryption is a global configuration option; all secure properties conform to the AES configuration setting. It is not possible to export a subset of the ACI fabric configuration such as a tenant configuration with AES encryption while not encrypting the remainder of the fabric configuration. See the *Cisco Application Centric Infrastructure Fundamentals*, "Secure Properties" chapter for the list of secure properties.

The APIC uses a 16 to 32 character passphrase to generate the AES-256 keys. The APIC GUI displays a hash of the AES passphrase. This hash can be used to see if the same passphrases was used on two ACI fabrics. This hash can be copied to a client computer where it can be compared to the passphrase hash of another ACI fabric to see if they were generated with the same passphrase. The hash cannot be used to reconstruct the original passphrase or the AES-256 keys.

Observe the following guidelines when working with encrypted configuration files:

- Backward compatibility is supported for importing old ACI configurations into ACI fabrics that use the AES encryption configuration option.

Note Reverse compatibility is not supported; configurations exported from ACI fabrics that have enabled AES encryption cannot be imported into older versions of the APIC software.

- Always enable AES encryption when performing fabric backup configuration exports. Doing so will assure that all the secure properties of the configuration will be successfully imported when restoring the fabric.

Note If a fabric backup configuration is exported without AES encryption enabled, none of the secure properties will be included in the export. Since such an unencrypted backup would not include any of the secure properties, it is possible that importing such a file to restore a system could result in the administrator along with all users of the fabric being locked out of the system.

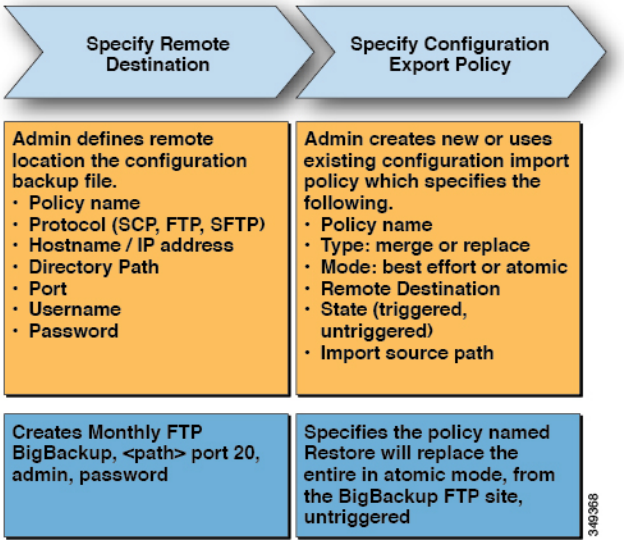
- The AES passphrase that generates the encryption keys cannot be recovered or read by an ACI administrator or any other user. The AES passphrase is not stored. The APIC uses the AES passphrase to generate the AES keys, then discards the passphrase. The AES keys are not exported. The AES keys cannot be recovered since they are not exported and cannot be retrieved via the REST API.
- The same AES-256 passphrase always generates the same AES-256 keys. Configuration export files can be imported into other ACI fabrics that use the same AES passphrase.
- For troubleshooting purposes, export a configuration file that does not contain the encrypted data of the secure properties. Temporarily turning off encryption before performing the configuration export removes the values of all secure properties from the exported configuration. To import such a configuration file that has all secure properties removed, use the import merge mode; do not use the import replace mode. Using the import merge mode will preserve the existing secure properties in the ACI fabric.
- By default, the APIC rejects configuration imports of files that contain fields that cannot be decrypted. Use caution when turning off this setting. Performing a configuration import inappropriately when this default setting is turned off could result in all the passwords of the ACI fabric to be removed upon the import of a configuration file that does not match the AES encryption settings of the fabric.

Note Failure to observe this guideline could result in all users, including fabric administrations, being locked out of the system.

Configuration Export

The following figure shows how the process works for configuring an export policy.

Figure 10: Workflow for Configuring an Export Policy



The APIC applies this policy in the following way:

- A complete system configuration backup is performed once a month.
- The backup is stored in XML format on the BigBackup FTP site.
- The policy is triggered (it is active).

Configuration Import

An administrator can create an import policy that performs the import in one of the following two modes:

- Best-effort—ignores objects within a shard that cannot be imported.

Note If the version of the incoming configuration is incompatible with the existing system, shards that are incompatible are not imported while the import proceeds with those that can be imported.

- Atomic—ignores shards that contain objects that cannot be imported while proceeding with shards that can be imported.

Note If the version of the incoming configuration is incompatible with the existing system, the import terminates. In this case, use Best-effort instead.

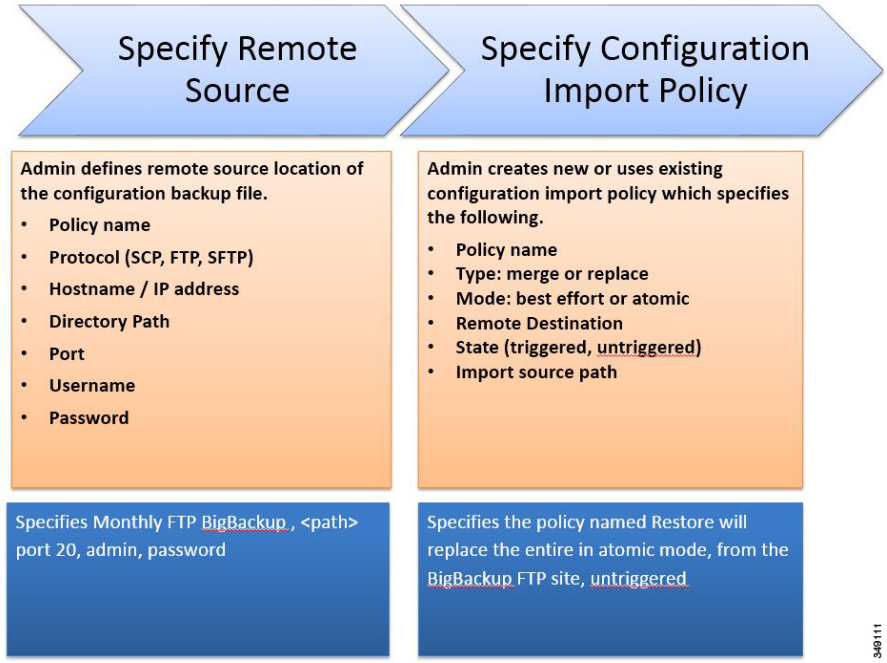
An import policy supports the following combinations of mode and type:

- Best-effort Merge—imported configuration is merged with existing configuration but ignores objects that cannot be imported.

- Atomic Merge—imported configuration is merged with the existing configuration, but ignores shards that contain objects that cannot be imported.
- Atomic Replace—overwrites existing configuration with imported configuration data. Any objects in the existing configuration that do not exist in the imported configuration are deleted. Objects are deleted from the existing configuration that have children in the existing configuration but do not have children in the incoming imported configuration. For example, if an existing configuration has two tenants, solar and wind, but the imported backed up configuration was saved before the tenant wind was created, tenant soar is restored from the backup but tenant wind is deleted.

The following figure shows how the process works for configuring an import policy.

Figure 11: Workflow for Configuring an Import Policy



The APIC applies this policy in the following way:

- A policy is created to perform a complete system configuration restore from monthly backup.
- The atomic replace mode does the following:
  - Overwrites the existing configuration.
  - Deletes any existing configuration objects that are not present in the imported file.
  - Deletes non-present children objects.
- The policy is untriggered (it is available but has not been activated).

Tech Support, Statistics, Core

An administrator can configure export policies in the APIC to export statistics, technical support collections, faults and events, to process core files and debug data from the fabric (APIC as well as switch) to any external

host. The exports can be in a variety of formats, including XML, JSON, web sockets, SCP, or HTTP. Exports are subscribable, and can be streaming, periodic, or on-demand.

Note The maximum number of statistics export policies is approximately equal to the number of tenants. Each tenant can have multiple statistics export policies and multiple tenants can share the same export policy, but the total number of policies is limited to approximately the number of tenants.

An administrator can configure policy details such as the transfer protocol, compression algorithm, and frequency of transfer. Policies can be configured by users who are authenticated using AAA. A security mechanism for the actual transfer is based on a username and password. Internally, a policy element handles the triggering of data.

Programmability Using Puppet

About Puppet

Puppet is a configuration management tool from Puppet Labs, Inc. Although Puppet was originally designed for large scale server management, many datacenter operators would like to consolidate server and network device provisioning using the same tool.

The following items are the primary components of a Puppet implementation:

- **Manifest** – A Puppet manifest is a collection of property definitions for setting the state of a managed device (node). The details for checking and setting these property states are abstracted so that a manifest can be used for more than one operating system or platform.
- **Master** – A Puppet master (server) typically runs on a separate dedicated server and serves multiple nodes. The Puppet master compiles configuration manifests and provides them to the nodes on request.
- **Agent or Device** – A Puppet agent runs on the node, where it periodically connects to the Puppet master to request a configuration manifest. The agent reconciles the received manifest with the current state of the node, updating the node state as necessary to resolve any differences. For nodes that cannot run an embedded Puppet agent or prefer not to, Puppet supports a construct called a Puppet device. A Puppet device is essentially a proxy mechanism, external to the node, that requests the manifest from the Puppet master on behalf of the node. The Puppet device then applies any updates required by the received manifest to the node. To leverage this capability, a vendor must provide a vendor-specific implementation of the device class along with a Puppet module that makes use of the device. The vendor-specific device class uses a proprietary protocol or API to configure the remote node.

For further information and documentation about Puppet, see the Puppet website at the following URL: https://puppet.com/.

Cisco ciscoacipuppet Puppet Module

An APIC controller does not run an embedded Puppet agent. Instead, Cisco provides a Puppet module ("ciscoacipuppet"), which uses a Cisco ACI-specific Puppet device to relay configuration management requests to the APIC controller. The ciscoacipuppet module interprets change information in the received Puppet

manifest and translates the change requests into APIC REST API messages to implement configuration changes in the ACI fabric.

For details on the installation, setup, and usage of the ciscoacipuppet module, refer to the documentation on GitHub and Puppet Forge at the following URLs:

- **GitHub** – https://github.com/cisco/cisco-network-puppet-module
- **Puppet Forge** – https://forge.puppet.com/puppetlabs/ciscoacipuppet

Puppet Guidelines and Limitations for ACI

- Only a subset of APIC managed objects can be provisioned using the ciscoacipuppet Puppet module. To understand the level of support and the limitations, refer to the ciscoacipuppet module documentation on GitHub and Puppet Forge.