CA Identity Manager™

Provisioning Reference Guide

12.6.5

CA Technologies Product References

This document references the following CA Technologies products:

- CA CloudMinder™ Identity Management
- CA Directory
- CA Identity Manager ™
- CA Identity Governance (formerly CA GovernanceMinder)
- CA SiteMinder®
- CA User Activity Reporting
- CA AuthMinder ™

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: Provisioning Manager

This section contains the following topics:

User Interface for Provisioning

Provisioning Manager is the user interface for advanced provisioning operations. This interface was formerly called eTrust Admin Manager. To use Provisioning Manager, choose Start, CA, CA Identity Manager, Provisioning Manager.

Note: Provisioning Manager will not be available in future releases, so we recommend using the CA Identity Manager User Console instead. This guide describes the provisioning features of CA Identity Manager that are not integrated into the CA Identity Manager User Console. For the features that are integrated into the User Console, such as account management tasks, see the *Administration Guide*.

Provisioning Server

The Provisioning Server is the server that manages additional accounts that are assigned to a CA Identity Manager user. When you assign a provisioning role to a CA Identity Manager user, the Provisioning Server creates accounts on endpoints that meet the requirements of the role. For example, if you assign a provisioning role that includes an Exchange account template, the Provisioning Server assigns an Exchange account to the user.

Administrator Authentication

Objects in the provisioning domain are protected at several different levels, but overall access to the domain is protected by authentication security, which requires all administrators to identify themselves. The global user name and password that the administrator enters are checked against information stored in the provisioning directory.

Note: You can configure the Provisioning Server to request authentication with a native system. For more information, see the *Administration Guide*.

Administrator Login

The first time you log on to the Provisioning Manager, you use the etaadmin global user, whose password was set up when the Provisioning Server was installed.

The Provisioning Server also provides other built-in global users that provide authentication information for use only by Provisioning Server components. You will not use these global users to log on; instead, they provide additional authentication information for the domain.

The etaadmin user is similar to a built-in superuser account. You set a password for this user when the Provisioning Server is installed. It is *imperative* to remember that password because you need it to log on as etaadmin the first time you use Provisioning Manager.

When you log on using the etaadmin object, you have access to all the objects in the domain. You should immediately create a global user object for yourself and assign the DomainAdministrator profile to it. When this user has been created, log in as that user. You must not perform any more actions as the non-specific etaadmin administrator. Avoiding the use of etaadmin improves the traceability of actions as seen in various logs. For more security, you can delete or suspend the etaadmin user after you create your own account.

Administrator Authorization

Authorization determines what administrators can do on the Provisioning Server. It defines the privileges that an administrator has in a domain. You can authorize an administrator by assigning an *admin profile* to the administrator's global user object or by assigning an admin profile to one of the administrator global user groups.

Admin Profiles

Admin profiles permit administrators certain types of access and privileges to manage objects in a domain. Admin profiles contain all the privileges that administrators need to perform different tasks. While administrative privileges can be assigned directly to global users, using admin profiles provides several advantages:

- Several administrators can be defined to a profile, and therefore, each receives the same administrative privileges.
- The operations that an administrator is allowed to perform is typically to necessitate being granted a long list of administrative privileges. Placing them into an admin profile is less error prone as it lets you define the profile once and then apply those privileges to multiple administrators.
- Admin profiles can be accessed from other domains, making it easy for other administrators to create new profiles from existing profiles in other domains.

Note: When assigning individual administrative privileges, you must give the administrator Read access to the object and its container. This access is necessary to list and search for objects.

Default Admin Profiles

The Provisioning Server provides default admin profiles that control the privileges of an administrator. These profiles give administrators access to the objects in the domain of the profile. Like the default administrator objects, such as etaadmin, the following profiles are created automatically when you install the Provisioning Server:

- **DomainAdministrator and DomainAdministrator-NoWeb-**Gives administrators full access to every object in the domain. Administrators who have this profile in the root domain have full access to all Provisioning Server objects and security information.
- **PasswordAdministrator**-Lets administrators change passwords and activate or suspend global users.
- **UserAdministrator**-Lets administrators manage users in the domain. Administrators with this profile cannot modify provisioning roles or account templates.
- **ReadAdministrator**-Lets administrators read every object in the domain.
- **SelfAdministrator**-Defines the actions that can be performed by self-administrators. By default, this profile authorizes self-administrators to read their own global user object, list their accounts, and modify specific attributes of their own global user or accounts. You can customize this profile to meet your self-administrator authorization requirements.

Chapter 2: Advanced Configuration Options

This section contains the following topics:

Advanced Configuration Options Overview

The advanced configuration options for the Provisioning Server fall into two categories:

Global Properties

Configuration settings that are saved in the provisioning directory and control the behavior of the Provisioning Server. Click the Global Properties button on the Provisioning Manager System Task frame.

Domain Configuration

Configuration settings that are saved in the provisioning directory and control the behavior of the Provisioning Server. Click the Domain Configuration button on the Provisioning Manager System Task frame.

Additional places to configure various components of the Provisioning Server include the following files:

***PSHOME*\Data\im-ps.conf**

Parameters control general behaviors of the Provisioning Server service not controllable through Domain Configuration parameters. See the *Installation Guide* for more details and comments in the file itself.

Note: PSHOME is the directory where the Provisioning Server is installed.

***PSOME*\Data\im_ccs.conf**

Parameters control general behaviors of the CA Identity Manager C++ Connector Server service. See the *Installation Guide* for more details and comments in the file itself.

***PSAHOME*\data\eta_pwdsync.conf**

Parameters control an optional Password Synchronization Agent that you can install on a Windows system. For more information, see the *Administration Guide*.

Note: *PSAHOME* is the directory where the Password Synchronization Agent is installed.

***%DXHOME%*\config\knowledge*.dxg and *.dxc**

Used to configure CA Directory failover. For more information, see the *Installation Guide*.

Note: %DXHOME%is the directory where CA Directory is installed.

Global Properties

Global Properties are stored in the Provisioning Directory. These properties control the entire enterprise and as such are stored outside of any of your domain-specific data. To view these properties you must have a privilege that grants Read access to the SystemSettings object in the ETA domain. All predefined admin profiles in any domain grant this read access.

To change these properties you must have a privilege that grants Modify access to the SystemSettings object in the domain. The DomainAdministrator and DomainAdministrator-NoWeb admin profiles in any domain grant this modify access.

Updates to global properties in most cases take effect immediately, no restarting of services or programs is necessary. Two specific exceptions are:

- Properties that control Manager behaviors, such as UID controls, and Full Name controls, do not affect property sheets that are already displayed. You may have to close a property sheet and reopen it to have the change take effect for your Manager.
- Logging settings are broadcast to affected Provisioning and CA Identity Manager Provisioning Connector server services. However, this broadcast currently only goes to one Provisioning Server service per domain. If you have installed a failover or load-balancing configuration with multiple Provisioning Server services for a single domain, you will need to restart all Provisioning and Connector Server services for that domain to ensure that the new logging settings are recognized by all affected components.

Domain Configuration

Domain configuration parameters are also stored in the provisioning directory. You manage these from the System Task of Provisioning Manager using the Domain Configuration button. Parameters are organized into a tree hierarchy using folders so that related parameters are easier to manage. These parameters control the Provisioning Server for a single domain.

If you configured multiple alternative servers, each with its own Provisioning Server for the same domain, all servers for the domain share the same configuration parameter settings. There are a few parameters that you might want to set to different values on different servers, even in the same domain. Per-server values are referred to as specializations. Use the Add Specialization or Remove Specialization menu items to work with server-specific values. These server-specific specializations are displayed in the tree hierarchy under the domain parameter. If there is no specialization for a particular server, the domain parameter value applies to that server. In most cases, the Provisioning Server lets you create a specialization for a parameter even if that could result in inconsistent behaviors from the alternative servers for a domain. This lets you have a dedicated server that is used for a specialized purpose where you actually want that different behavior. However, for a small set of parameters, specializations are not allowed. A typical reason would be because client code needs to know the value of the parameter even when it does not know which server handles its request. In those cases, the Add Specialization menu item is disabled.

To view these parameters, you must have a privilege that grants Read access to Configuration Parameter objects in the domain where they reside. All the predefined admin profiles grant this Read access to the domain configuration parameters in their own domain, including any subordinate domain.

To change these parameters, you must have a privilege that grants Modify access to Configuration Parameter objects in the domain where they reside. The DomainAdministrator and DomainAdministrator-NoWeb admin profiles grant Modify access to the domain configuration parameters in their own domain and any subordinate domain. You can create a custom admin profile that grants Read or Modify access to specific configuration parameters if you need scoping control.

Domain Configuration parameter updates take effect immediately on the provisioning server where the update was processed. However, if you configured multiple alternative provisioning servers for the domain, the other servers will not take the changed parameters into account immediately. The updated parameters are stored in the provisioning directory immediately, but each affected Provisioning Server refreshes its knowledge of the parameter values periodically. By default the update frequency is every 10 minutes; however you may change this value with the parameter Configuration Setup/Parameter Update Time described later. Thus you have to wait up to 10 minutes for the refresh to take place. The refresh is recorded in the Provisioning Server Trace log with messages that include the text “ETA::Configuration update completed”.

Note: For more information, see the Transaction Log (see page 52) section.

You may choose to restart the affected Provisioning Server services to ensure that the parameters are updated. When the service starts, the service writes information to the Provisioning Server trace log about configuration parameters. This log can be valuable in understanding what parameters were in effect at any particular point. The following information is written to the trace log at startup:

- If the Transaction Log/Level domain configuration parameter is set to a value of 0 or greater and Transaction Log/Enabled is Yes, non-default configuration parameters values are written.
- If the Transaction Log/Level domain configuration parameter is set to a value of 1 or greater and Transaction Log/Enabled is Yes, all configuration parameters values are written.

Note: A few parameters do not take effect even after the periodic configuration parameter update. They only take effect on the restart of the Provisioning Server service. Such parameters display the following warning on their properties: Changing this parameter requires restarting all affected servers.

Provisioning Directory Parameters

The provisioning directory configuration folder contains parameters you can use if you have a non-default installation of your provisioning directory.

Provisioning Directory/Entry Count Attribute

Values: dxEntryCount (default) or <unset>

Description: Set this attribute to dxEntryCount if CA Directory is being used for the provisioning directory; but clear it otherwise. This attribute is used in queries sent to the provisioning directory to check whether size limits will be reached; but only if the client is not requesting partial results.

Authentication Parameters

The Authentication configuration folder contains parameters that you can use to customize user authentication behaviors of the Provisioning Server.

Disable Maintenance User

Values: No (default) or Yes

Description: Set this parameter to yes to disable the ability to authenticate to the Provisioning Server using the built-in user with the Distinguished Name cn=etaserver,dc=eta. This user, whose password is controlled by the pwdmgr utility, is used internally during installation.

After installation, this user is only needed for maintenance functions such as resetting an administrator's password. We recommend that you disable this user after installation.

Authorization Parameters

The Authorization configuration folder contains parameters that you can use to customize authorization behaviors of the Provisioning Server.

Check Owner Access on Indirect Privileges

Values: Yes (default) or No

Description: Controls what access checks are performed when assigning a global user group or admin profile to a global user, global user group or admin profile.

Regardless of the setting of this parameter, the Provisioning Server checks for Modify access to a specific attribute of the object being assigned and Modify access to a specific attribute of the object to which it is assigned.

If this parameter is Yes (the default), the Provisioning Server will also check for Owner access to each of the objects to which the assigned admin profile or global user group grants access. This prevents one from being able to assign privileges through a global user group or admin profile that one could not have assigned directly to the target global user, global user group or admin profile.

If you do not need added protection, this parameter can be set to No to disable additional Owner access checks. Doing so lets you have one set of administrators who define admin profiles and another set of administrators who assign those admin profiles to users.

Cache Parameters

The Cache configuration folder contains parameters that allow you to tune the Provisioning Server internal caches.

Important! Changes to cache parameters do not take effect until the Provisioning Server service is restarted.

The following parameters control each cache:

Maximum Age

The maximum time in seconds that an item remains in the cache without being reread from the provisioning directory.

Maximum Size

The maximum number of unused items to retain in the cache. When an operation uses a cache item, the item is considered in-use. There is no limit on the number of in-use cache items. However, when all operations finish with the cache item, the item is marked unused and it is retained only when the number of used and unused items in the cache is no more than the configured maximum size.

Cache items are also removed from a cache when explicitly canceled. This occurs when a change is made to the provisioning directory data from which the cache item originates. This cache invalidation only occurs on the Provisioning Server that processed that provisioning directory update. If you have multiple provisioning domains or alternative servers serving a single domain, other servers may have cache items that are still derived from the prior data. That is why there is a cache maximum age parameter.

Cache items also are canceled when access is to be denied. The privilege caches (Admin Profile, Global User and Global User Group) contain privilege information that is used to perform authorization checks. If you have recently assigned a privilege to someone, you do not want to have to wait up to 10 minutes (the default cache maximum age for these caches) for that privilege addition to be recognized. Therefore if an authorization check using cached privileges is about to report DENIED, the cache items are canceled and re-initialized from the provisioning directory. If the result is still DENIED, that authorization failure is reported to the administrator.

Important! When you remove a privilege from a global user, admin profile, or global user group, expect that this change will take place at most 10 minutes (the default) from the time of the change. In most cases this is sufficient. However, if the reason for removing the access is to remove an imminent security threat, to ensure immediate enforcement of that privilege change requires you to restart all affected Provisioning Server services.

Admin Profile Privilege Cache

Each admin profile privilege cache item stores information that is obtained from an admin profile, including administrative privileges, and the names of included admin profiles.

Parameter: Cache/Admin Profile Privilege Cache Maximum Age

Default Value: 600 seconds (equals 10 minutes)

Parameter: Cache/Admin Profile Privilege Cache Maximum Size

Default Value: 10

Domain Cache

Each domain cache item stores information obtained from a Domain (DSA) Registration object. These objects record information that is necessary for one Provisioning Server to talk to another Provisioning Server.

Parameter: Cache/Domain Cache Maximum Age

Default Value: 3600 seconds (equals 1 hour)

Parameter: Cache/Domain Cache Maximum Size

Default Value: 20

Global User Group Privilege Cache

Each global user group privilege cache item stores information obtained from a global user group, including administrative privileges, the names of included admin profiles, and the names of included global user groups.

Parameter: Cache/Global User Group Privilege Cache Maximum Age

Default Value: 600 seconds (equals 10 minutes)

Parameter: Cache/Global User Group Privilege Cache Maximum Size

Default Value: 10

Global User Privilege Cache

Each global user privilege cache item stores information that is obtained from a global user and its assigned admin profiles and global user groups. This information includes administrative privileges, password, suspension status and the names of included admin profiles and including global user groups.

Unlike the admin profile privilege cache and global user group privilege cache, the global user privilege cache items also store indirect information obtained from referenced items, so it contains a full list of the accesses privileges that a global user has.

Each time a global user privilege cache item is initialized, the global user's full list of effective privileges and assigned admin profiles and global user groups is written to the server trace log. This information, written only if Transaction Log/Level is set to 4 or greater, includes information that obtained directly from the global user, information obtained indirectly from assigned global user groups and admin profiles, and information obtained implicitly based on assigned web or workflow privileges. Look for the text "EFFECTIVE PRIVILEGE LIST INITIALIZED" in the server trace log. This will be followed by the distinguished name of a global user and a list of privileges, a list of admin profiles and a list of global user groups.

Parameter: Cache/Global User Privilege Cache Maximum Age

Default Value: 600 seconds (equals 10 minutes)

Parameter: Cache/Global User Privilege Cache Maximum Size

Default Value: 20

Nested Role Search Cache

Each nested role search item stores information from the nested role search results. This is for user role removal synchronization only. For no object limit, set the size to -1. You can disable this cache by setting the size to 0.

Parameter: Cache/Nested Role Search Cache Maximum Age

Default Value: 600 seconds (equals 10 minutes)

Parameter: Cache/Nested Role Search Cache Maximum Size

Default Value: 0 (disabled)

Notification Config Cache

The Notification Configuration Cache stores configuration information that drives the CA Identity Manager Server Notification feature. This configuration information, which is stored in the provisioning directory and updatable by the service utility etaloadnotificationconf, defines the mapping between provisioning actions and notification records that are sent to the IMS.

Parameter: Cache/Notification Config Cache Maximum Age

Default Value: 600 seconds (equals 10 minutes)

Parameter: Cache/ Notification Config Cache Maximum Size

Default Value: 10

Operation Cache

Each operation cache item stores information about an ongoing or recently completed operation. When you do an explore operation, for instance, the displayed Operation detail count value is obtained from an operation cache item.

Parameter: Cache/Operation Cache Maximum Age

Default Value: 600 seconds (equals 10 minutes)

Parameter: Cache/Operation Cache Maximum Size

Default Value: 10

Password Profile Cache

Each password profile cache item stores information from a password profile. Currently there is only one password profile per domain.

Parameter: Cache/Password Profile Cache Maximum Age

Default Value: 600 seconds (equals 10 minutes)

Parameter: Cache/Password Profile Cache Maximum Size

Default Value: 10

Connector Server Cache

Each C++ Connector Server cache item stores a pool of connections between the Provisioning Server and the C++ Connector Server. The C++ Connector Server, also known as the Connector Server service, is the component that loads each endpoint type agent module.

Some endpoint types (for example Active Directory) provide a feature to use administrator credentials rather than a configured set of proxy credentials for authenticating to the managed directory. Each C++ Connector Server cache item represents a pool of LDAP connections using a single set of administrator credentials.

Parameter: Cache/Connector Server Cache Maximum Age

Default Value: 3600 seconds (equals 1 hour)

Parameter: Cache/Connector Server Cache Maximum Size

Default Value: 20

Compatibility Parameters

The Compatibility configuration folder contains parameters that you can use to provide temporary backwards compatibility with prior releases of eTrust Admin.

Enable Operation Details

Values: Yes (default) or No

Description: Tracks changes to accounts affected by provisioning operations in etautil or the Provisioning Manager, such as Explore and Correlate. If you make no provisioning changes in the Provisioning Manager or etautil, you can set this parameter to no. Disabling it could keep the management of operation detail records from affecting CA Directory performance.

With or without this parameter, inbound synchronization still updates the CA Identity Manager server. Each provisioning server sends account details to the CA Identity Manager server or cluster of servers. Those details include attribute-level information not found in the operation details.

Relax Self Q&A Reads

Values: No (default) or Yes

Description: By default (No), global user self authentication Q&A attributes are returned only when explicitly asked for by users. This allows the Provisioning Server to log when these questions and answers are viewed. Some older clients depend on the prior behavior where these attributes could be retrieved along with all other Global User attributes. Set this parameter to Yes to re-instate the prior behavior to allow these applications to work with the current Provisioning Server.

Configuration Setup Parameters

The Configuration Setup configuration folder contains parameters you can use to configure the processing of these domain configuration parameters.

Parameters Update Time

Default Value: 600 seconds (equals 10 minutes)

Description: Configuration parameters are read periodically from the provisioning directory. This parameter defines how often, in seconds, this refresh of parameters occurs. Hence, this parameter defines the maximum amount of time one would need to wait after making a change to parameter before being assured the change has taken effect.

The minimum value for this parameter is 30 seconds.

Connections Parameters

The Connections configuration folder contains parameters you can use to tune the connection management mechanisms within the Provisioning Server.

The Provisioning Server maintains pools of LDAP connections that it uses for communicating with the provisioning directory, with connector servers and with other LDAP servers. A dedicated thread within the provisioning server (the connection monitor thread) wakes up periodically to adjust the pools by closing excess idle connections and attempting to create connections to LDAP servers previously believed to be unavailable.

The configuration parameters in this folder are consulted by the connection monitor thread as it performs its functions.

CS Pool Maximum Size

Default Value: 200

Description: The maximum size of each of the Provisioning Server CS Connection Pools. A CS Connection Pool is a reusable set of LDAP connections that are used to communicate with a specific connector server.

CS Pool Minimum Size

Default Value: 2

Description: The minimum size of each of the Provisioning Server's CS Connection Pools. The connection monitor thread, when it closes expired idle connections, will retain at least this many connections in each CS Connection Pool.

DB Pool Maximum Size

Default Value: 40

Description: The maximum size of the Provisioning Server DB Connection Pool. The DB Connection Pool is a reusable set of LDAP connections that are used to communicate with the Provisioning Directory.

DB Pool Minimum Size

Default Value: 5

Description: The minimum size of the Provisioning Server DB Connection Pool. The connection monitor thread, when it closes expired idle connections, will retain at least this many connections in DB Connection Pool.

Expiration Time

Default Value: 1800 seconds (30 minutes)

Description: The time, in seconds, after which an idle connection in the provisioning server's LDAP connection pools will be considered expired. An expired connection is a candidate for being closed by the connection monitor thread.

Other Pool Maximum Size

Default Value: 20

Description: The maximum size of each of the Provisioning Server's Ad Hoc Connection Pools. Each Ad Hoc Connection Pool is a reusable set of LDAP connections that are used to communicate with a specific LDAP server other than the provisioning directory or regularly used connector servers. For example, changes to endpoint or endpoint type attributes may need to be sent to another provisioning server's connector server, and the connection pool to communicate with that connector server is governed by this parameter.

Other Pool Minimum Size

Default Value: 0

Description: The minimum size of each of the provisioning server Ad Hoc Connection Pools. This value is typically zero as there is rarely a need to retain idle connections to these LDAP servers past their normal expiration time.

Refresh Time

Default Value: 300 seconds (5 minutes)

Description: The time, in seconds, that the provisioning server connection monitor thread waits between iterations. Each time this thread awakens, it identifies expired connections in its LDAP connection pools and closes them. It also attempts to establish LDAP connections to servers that were believed to be unavailable (but only for pools with a minimum size greater than zero).

Endpoint Parameters

The endpoint configuration folder contains parameters you can use to enable or disable features on an endpoint type-by-endpoint type or endpoint-by-endpoint basis. Each parameter can be set to an ordered list of values, each of which can be one of the following:

Parameter	Description
ALL	Enabled for all endpoints of all endpoint types.
-ALL	Disabled for all endpoints of all endpoint types.
EndpointType	Enabled for all endpoints of the specified endpoint type.

Parameter	Description
-EndpointType	Disabled for all endpoint of the specified endpoint type.
EndpointType:*Endpoint*	Enabled for the specified endpoint.
-EndpointType:Endpoint	Disabled for the specified endpoint.

If more than one value for the same parameter specifies the same directory, the last value that specifies the endpoint determines whether the feature is enabled or disabled for that endpoint. This lets you provide a more general rule first (enabled for all directories or all endpoint types) and follow that up with a more specific rule (disabled for endpoint ABC of the endpoint type ActiveDirectory).

Check Account Passwords

Default Value: -ALL (disabled for all endpoints of all endpoint types)

Description: When this parameter is enabled for a specific endpoint, the Provisioning Server checks any password in a password change of an existing account on that directory, including attempts to set an empty password.

During account creation, the Provisioning Server performs password quality checking when a password is provided. If no password is provided, no checking is performed unless the Check Empty Account Passwords parameter is also enabled for the directory.

Account password quality checking uses the Password Profile that exists in the domain of the global user that owns the account. If the account is not associated with any global user, then the Password Profile that exists in the domain of the account is used. If the password profile located based on the global user or the account's domain is disabled, account password quality checking is also disabled for that account.

Account password quality checking does not include the checks on self-changes that depend on history of recent password-change activity. Password reuse frequency (history) and minimum time between changes (interval checking) are only applicable to the global user password changes where the Provisioning Server retains an accurate history of recent changes. Account passwords and password history are not stored in the Provisioning Server. They are stored only in the managed directory and the Provisioning Server makes no assumption that all password changes are visible to the Provisioning Server.

A synchronized account password is an account password meeting the following criteria:

- Account is correlated to a non-restricted global user
- Account resides on a directory for which Disable password propagation to accounts has not been enabled
- Account has not been deleted (it is not in Delete Pending state)

An attempt to change a synchronized account password to the value of the current global user password will be accepted regardless of the setting of this configuration parameter. Also, the settings of the following configuration parameters can control the effect of password quality checking on synchronized account passwords:

`Passwords\Enforce Synchronized Account Passwords`

`Passwords\Use External Password Policies`

Check Empty Account Passwords

Default Value: -ALL (disabled for all endpoints of all endpoint types)

Description: When this parameter is enabled for a specific managed endpoint, the Provisioning Server checks any empty password in an Add request for an account on that endpoint. This parameter is ignored if Check Account Passwords is not also enabled for this endpoint.

This parameter is separate from Check Account Passwords because it is acceptable in some endpoint types to create an account with no password.

Use Account Template Status

Default Value: -ALL (disabled for all endpoint of all endpoint types)

Description: Parameter controls whether to ignore a global user status of suspended and allow the creation of accounts in the active state. By default accounts created from account templates for a suspended global user are suspended regardless of the suspended status indicated in the account template.

Validate Endpoint Credentials

Default Value: -ALL (disabled for all endpoint of all endpoint types)

Description: Parameter controls whether the Provisioning Server sends changes to passwords, updated on endpoint property sheets, to the applicable connector for immediate validation or if only the provisioning directory is updated. This functionality is not applicable to CA-provided connectors as they have all been updated to always have the behavior that is controlled by this parameter.

If you have a custom connector that is written using the Software Development Toolkit from a previous release of eTrust Admin and that connector stores proxy credentials in its endpoint properties, verify the behavior of your endpoint type by enabling this parameter for your endpoints and attempting to change those proxy credentials.

Explore and Correlate Parameters

The Explore and Correlate configuration folder contains parameters that you can use to configure the explore and correlate functions that are used while acquiring managed directories.

Explore Backlog Limitation

Default Value:

20480

Description: Prevents the exploration procedure from fetching too many objects from a connector server before handling them.

Note: This feature is only effective when 'Explore Lower Memory Cache' is enabled.

Correlation Attribute

Value Syntax: *GUAttrName*[=*Namespace*:*AccountAttrName*[:*Offset*,*Length*]]

Default Values:

GlobalUserName

FullName

Description: Controls the correlation matching algorithm used by the correlate phase of explore/correlate.

The correlation algorithm uses this parameter when determining how accounts are associated with global users.

Each value defines a global user attribute that will be compared against an account attribute. The list is ordered, and only values applicable for an endpoint type are used when correlating accounts from an endpoint of that endpoint type. If there are two defined mappings for the same global user attribute that are applicable to the endpoint type where correlate is being run, then the first parameter value is used.

You can provide this mapping in one of the following ways:

GUAttrName

In this form, you name only the global user attribute and not the corresponding account attribute. This value assumes for the omitted account attribute name the account attribute that is predefined by the endpoint type to correspond to this global user attribute. For information about the predefined mappings, see the endpoint type's *Connector Guide*.

A parameter value in this form applies to all endpoint types for which an account mapping is defined. All endpoint types define mappings for GlobalUserName (typically the account name). Most endpoint types define mappings for Full Name.

GUAttrName=Namespace:AccountAttrName

In this form, you name the global user attribute and a specific account attribute of a specific endpoint type. A parameter value in this form applies only to the indicated endpoint type. Use this form rather than the first form to match global users on an attribute such as Full Name in one endpoint type but not in all endpoint types.

GUAttrName=Namespace:AccountAttrName:Offset,Length

In this form, you name the global user attribute and a specified substring of an account attribute of a specific endpoint type. *Offset* indicates the start of the substring, the value 1 indicating the start of the attribute value. *Length* indicates the number of characters in the substring value. If the full account value is shorter than (*Length* + *Offset* - 1) characters, the substring value that is used will be shorter than *Length* characters.

A parameter value in this form applies only to the indicated endpoint type. Use this form if you know that an account attribute value (for example, description) has a form where the first eight characters are known to contain a unique employee identifier that can be matched to a global user attribute value.

For example, assume that the configured parameter values are the following:

```
GlobalUserName
FullName=LDAP Namespace:globalFullName
FullName=ActiveDirectory:DisplayName
 CustomField01=ActiveDirectory:Telephone
```

The following occurs for each previously uncorrelated account that is found while correlating accounts in an Active Directory container:

1. The Provisioning Server starts with the first parameter value (GlobalUserName) and determines that the Active Directory endpoint type's defined account attribute that maps to GlobalUserName is NT_AccountID (LDAP attribute name eTADSsAMAccountName). It attempts to find the unique global user whose name is equal to the account's NT_AccountID attribute value. If a unique match is found, the Provisioning Server associates the account with the global user. If more than one match is found, the Provisioning Server performs Step 5. If no match is found, the Provisioning Server performs the next step.
2. The Provisioning Server considers the second parameter value (FullName=LDAP Namespace:globalFullName). Since this value is specific to another endpoint type, it is skipped and the Provisioning Server performs the next step.
3. The Provisioning Server considers the third parameter value (FullName=ActiveDirectory:DisplayName). Since this value is specific to Active Directory, it is used. It attempts to find the unique global user whose FullName is equal to the account's DisplayName attribute value. If a unique match is found, the Provisioning Server associates the account with the global user. If more than one match is found, the Provisioning Server performs Step 5. If no match is found, the Provisioning Server performs step 4.
4. The Provisioning Server considers the final parameter value (CustomField01=ActiveDirectory:Telephone). Because this value is specific to Active Directory, it is used. It attempts to find the unique global user whose Custom Field #01 attribute is equal to the account's Telephone attribute value. Note that the name you gave to the custom global user attribute using global properties of the System Task is not displayed here. If a unique match is found, the Provisioning Server associates the account with the global user. If more than one match is found, the Provisioning Server performs Step 5. If no match is found, the Provisioning Server performs the next step.
5. The Provisioning Server associates the account with the [default user] object in the domain specified by the configuration parameter Explore and Correlate/Create Users Domain. If the [default user] object does not already exist, it is created.

Correlation Domain

Values: Root Domain

Description: A value indicating which domain or domains should be searched for global users during the Correlate with existing global users phase of explore/correlate. This parameter is deprecated as the root domain is the only choice now.

Create Users Default Attributes

Value Syntax: *GUAttrName=Value*.

Default Values: None

Description: The parameter provides default values for global user attributes for global users that are created during the Create Global Users phase of explore/correlate.

For example, use 'SelfAdministration=1' to enable self administration for your new global users. Use this feature to assign constant values to optional global user attributes for global users that are created during the acquisition of a primary directory.

Create Users Domain

Values: Root Domain

Description: The domain in which global users are to be created during the Create Global Users phase of explore/correlate. A value indicating which domain or domains should be searched for global users during the Correlate with existing global users phase of explore/correlate. This parameter is deprecated as the root domain is the only choice now.

Create Users Verify Not Correlated

Values: Yes (default) or No

Description: Temporarily set this parameter to No to enable the alternate behavior whereby the Create Global Users phase of explore/correlate will skip the check that the account is not already correlated to a global user.

This capability is deprecated since exploring primary endpoints no longer applies.

The Create Global Users function works as follows for each account present in the container being correlated.

- Check to see if the account is already correlated to an existing global user. If so, leave this account that is still correlated to that global user. On an initial acquisition, no accounts will be correlated. However, on a later re-explore/recorrelate this step is important so that the accounts remain correlated to the global user to which they were previously correlated.

  Note: In a primary endpoint, you would not expect accounts to be correlated to any global user other than the one named the same as the account. There are various scenarios where this could occur. For instance, you may have renamed the account or the global user at some point after they were correlated to one another. Or you might have some system accounts on your primary endpoint that you do not want to correlate to separate global users - opting instead to correlate them to a single restricted global user.

- Attempt to create a global user named the same as the account. If this global user already exists, go on to the next step. This can happen if the global user was also present in another primary endpoint or you deleted your primary endpoint and re-acquired, correlating to global users created during the prior acquisition.
- Create an inclusion between the account and the global user, correlating the account with the global user.

If this configuration parameter is set to No, the first step is skipped. This can greatly improve the performance since that test is time-consuming and slows down as the Provisioning Directory becomes untuned. If you are acquiring an endpoint with more than thousands of accounts, the Provisioning Directory needs to be tuned, but you do not have the chance to do that tuning in the middle of the long-running Correlate operation.

Important! Set this parameter to No only during the initial acquisition of your primary endpoints. Subsequent use can result in accounts that are incorrectly correlated to multiple global users. Once the acquisition is completed, set the parameter to Yes.

Map User ID to Lowercase

Values: No (default) or Yes

Description: Map all user IDs to lowercase when creating global users during the Create Users phase of explore/correlate. When you acquire one of your primary directories, you create global users for the accounts on that directory. Two of the global user properties that get set by this creation of global users are the following:

Account Name (LDAP attribute name eTUserid): Set to the corresponding account's account name property.

Global User Name (LDAP attribute name eTGlobalUserName): Set to be the same as the Account Name property, but translated to lowercase.

If you acquire a primary directory with mixed case account names, this will by default result in the created global user's Account Name property also having mixed case. Set the configuration parameter to Yes to force the Account Name property to be the same as the Global User Name property - always in lowercase.

Preserve the original case in the Account Name property by leaving the configuration parameter set to No if you have no endpoint types such as UNIX for which account names are case-sensitive.

An alternative to setting this parameter to Yes is to define your case-sensitive endpoint types' account template rule expressions for the account name to use the TOLOWER built-in rule function, *%$$TOLOWER(%AC%)%*, instead of the normal account name rule expression, which is *%AC%*.

Note: If your primary directory has only uppercase account names, this configuration parameter has no effect. The global user's Account Name property will already be translated to lowercase.

Explore Compare in Memory

Values: No (default) or Yes

Description: Obtains two lists of objects at a time: one from the endpoint system being explored and one from the provisioning directory. These lists are compared in memory to determine what changes should be applied to the provisioning directory. If this parameter is No, only a single list of objects at a time is explored and correlated, which uses far less virtual memory when working with large lists of objects.

Explore Lower Memory Cache

Values: No (default) or Yes

Description: Performs the alternative (classic) exploration algorithm which does not reserve the search results in memory when searching endpoints or the Provisioning Directory. Before using this parameter, disable the Explore Compare In Memory parameter.

CA Identity Manager Server Parameters

The CA Identity Manager Server configuration folder contains parameters you can use to control interactions between the Provisioning Server and the CA Identity Manager Server. Before enabling any of these parameters, you should verify the configuration of the communication between the Provisioning Server and the CA Identity Manager using the “CA Identity Manager Setup” button on the System task of Provisioning Manager,

Enable User Access

Values: No (default) or Yes

Description: Enables/disables retrieval of a corporate user attributes from the CA Identity Manager Server during account template evaluation.

Important! This feature was not available at the publication time for this document. Please check the availability of the feature in the release notes before enabling this parameter.

Allow Concurrent Modification on Same Global User

Values: Yes (default) or No

In some high availability environments, under heavy load, CA Identity Manager can send concurrent requests to the Provisioning Server.These concurrent requests can introduce race conditions in the Provisioning Server when handling parallel modification requests on same Global User. Changing this parameter to No can prevent the race condition.

Note: If a Program Exit accesses Global Users, do not disable this setting.

The change takes effect when the Provisioning Server restarts.

Enable Notification

Values: No (default) or Yes

Description: Enables or disables the collection of audit data (notifications) by the Provisioning Server for transmission to the CA Identity Manager Server. When this parameter is enabled, any changes to data managed by the Provisioning Server, other than changes directly initiated by the CA Identity Manager Server, generate notifications. The notifications are queued in the Notification DSA and then later sent to the CA Identity Manager Server. Upon receipt at the CA Identity Manager Server, certain notifications trigger events, while most are simply added to the full CA Identity Manager audit data.

Notify Batch Size

Default Value: 100

Description: The number of notifications that are processed in one batch. When sending notifications to the CA Identity Manager Server, the Provisioning Server will retrieve at most this many records (a batch) from the Notification DSA, process those entries, and then continue with additional batches.

Notify Retry Time

Default Value: 600 seconds (10 minutes)

Description: The time, in seconds, that the notification thread pauses between iterations. The notification thread is a dedicated thread within the Provisioning Server that wakes up periodically and attempts to transmit (or retransmit) any queued notifications.

Notify Timeout

Default Value: 30 seconds

Description: The timeout value, in seconds, for sending notifications or password validations to the CA Identity Manager Server. A value of zero indicates an unlimited timeout.

Use External Password Policies

Description: When set to Yes, CA Identity Manager validates changes to global user passwords or synchronized account passwords against CA Identity Manager password policies. Synchronized account passwords are the passwords for user accounts on endpoints for which the Disable Password Propagation property is disabled. Set the parameter Enforce Synchronized Account Passwords to Yes whenever Use External Password Policies is set to Yes. When this parameter is set to Yes, the Provisioning Server password rules that are applicable to users changing their own passwords (Password history checks and Minimum interval between self-changes) are no longer consulted.

Values: Yes (default) or No

Note: Even when integration with CA Identity Manager password policies is enabled with this configuration parameter, the Provisioning Server uses its per-domain password profiles in various situations. In particular, Administrative password changes, initial global user passwords, changes to unsynchronized account passwords and generating temporary initial passwords all consult the Provisioning Server password profile. In addition, the Locking and Password Expiration features defined in the Provisioning Server password profile are always used. However, the Provisioning Server password profile rules that are applicable to users changing their own passwords (Password history checks and Minimum interval between self-changes) are not consulted when this configuration parameter is Yes.

Operation Details Parameters

The Operation Details configuration folder contains parameters that you can use to control the behavior of operation details. Operation Details is the function that tracks the status of child operation that is spawned from higher-level operations such as Explore, Synchronization, or Propagation. When you perform one of these higher-level operations from CA Identity Manager tasks or from Provisioning Manager, you receive a message in the message summarizing the results of the child operation.

The following is a sample summary message for a User Synchronization request:

```
(accounts created: 1, updated: 1, recreated: 0, failures: 0)
```

If you ask to view status details for the task (or double-click the icon next to the summary message when using Provisioning Manager), this displays a screen with operation details including a series of success, failure, or warning messages corresponding to the statistics present in the summary message.

Maximum Operation Detail

Default Value: 100

Description: The maximum number of operation detail items which can be retrieved in one search of an operation object. When you perform a high-level operation that spawns hundreds or thousands of child operations and you call up the Operation Status window, this parameter controls how the details are returned from the Provisioning Server to the Provisioning Manager or other client application.

Operation Details Expiration Time

Default Value: 96 hours (equals 4 days)

Description: The number of hours to keep operation details in the provisioning directory.

Operation details are maintained in the server in the following parts:

1. An operation object stored in the provisioning directory (one per high-level operation).
2. An XML data file stored in the Operations folder containing the operation details, concatenated one after another.

Both parts are deleted when the operation object is deleted. Some clients delete their operation objects as soon as they retrieve the operation details or when the client terminates. Other clients such as Provisioning Manager leave the operation objects in the directory until they expire and are deleted in four days (by default).

Operations Folder

Default Value: Operations

Description: The name of the folder on the Provisioning Server where the XML data files storing operation details reside. This value can be a simple filename or a relative path name. However, it may not be an absolute path name.

Its value is relative to one of the following file path names:

```
%ETAVARHOME%
PSHOME
..
```

Normally, this means that the operations folder is placed along side the Data and Logs folder with a path name like the following:

```
C:\Program Files\CA\Identity Manager\Provisioning Server\Operations
```

However, to relocate this folder to another drive (so as to be able to run from a read-only drive), you should set the environment variable %ETAVARHOME% to a value such as D:\ProvisioningData before restarting the Provisioning Server service. Then the operations XML files will be placed instead into the following folder:

```
D:\ProvisioningData\Operations
```

The ETAVARHOME value can also be set as a registry value instead of an environment variable by using the eta-env utility that is installed with the provisioning server:

```
eta-env action=set name="ETAVARHOME" value="D:\ProvisioningData"
```

Important! Changes to this parameter do not take effect until the Provisioning Server service is restarted.

Password Synchronization Parameters

The Password Synchronization configuration folder contains parameters that you can use to control the behavior of password synchronization operations. Password synchronization is the feature that involves installing the Password Synchronization Agent on a Windows system or other systems to intercept password changes, send password validation requests, and password notification requests to the Provisioning Server.

Update Only Global User

Values: No (default) or Yes

Description: This parameter controls what action is carried out when the Provisioning Server receives a password change notification. By default, the new endpoint account password received in a password change notification is used first to update the global user's password and then to update all of that global user's account passwords for accounts other than the one from which the notification arrived.

Set this parameter to Yes to change this behavior so that only the global user password is updated. No account passwords will then be updated.

There are various situations in which the global user and affected accounts are not updated, including the following:

- Global user Enable Password Synchronization Agent property is not enabled. Global Users and account passwords are not updated.
- Password change notification occurred during the Agent Response Threshold period and is treated as a false password change notification. Global Users and account passwords are not updated.
- An endpoint containing one of the global user's accounts is marked on its Endpoint Properties for Disable propagation to accounts. The accounts on this endpoint are not updated.
- Global user Restricted property is enabled. Restricted global users such as [default user] are protected from accidentally propagating changes to their associated accounts.

Agent Response Threshold

Default Value: 600 seconds (equals 10 minutes)

Description: Maximum expected duration (in seconds) of each password change that the Provisioning Server sends to a managed endpoint on which a password synchronization agent is installed. This parameter allows the Provisioning Server to recognize when a Password Synchronization agent is processing a password change that is sent to it by the Provisioning Server as distinct from a password change originating on that managed endpoint.

When installing a password synchronization agent, you must check that the Password synchronization agent is installed check box on the Endpoint Settings tab. Then when the Provisioning Server sends a password change to the managed endpoint, it records the time when the password was sent. For a number of seconds set by the Agent Response Threshold, any password change notification or password validation request that is received for this account is rejected. Only password changes originating on the native system initiate password synchronization. Account password changes originating in the Provisioning Server update the account but not the global user or other accounts.

If, during the Agent Response Threshold, a password other than the password just sent to the managed endpoint is provided in a password validation or password change notification, this password is rejected. Two concurrent password changes to the same account are not allowed.

Password Parameters

The Password configuration folder contains parameters that you can use to control the behavior of certain password operations.

Enforce Synchronized Account Passwords

Values: Yes (default) or No

Description: When Yes, users cannot change any of their synchronized account passwords to a value other than the current value of their global user password. We recommend that you set this parameter to Yes whenever the CA Identity Manager Server\Use External Password Policies parameter is set to Yes.

Users' synchronized account passwords are the passwords for their accounts on endpoints for which the Disable Password Propagation property is disabled and which have not been marked as Delete Pending.

Pre-expire Passwords

Values: No (default) or Yes

Description: Controls having new global users that are created with their passwords already expired, forcing users to change their passwords during the initial login. If you set this parameter to Yes, global users that are created from non-interactive interfaces have their password that is initially set as expired. This is represented in the global user properties as a value of 1 for the property PwdPreExpired. This option appears on the global user's property sheet as the Force one-time expiration (mark password as temporary) check box.

The setting of the password as initially expired occurs when global users are created through the following interfaces:

- Correlate. When acquiring a primary endpoint, global users are created for each account. These users will generally not have a password unless you set a constant value using the Create Users Default Attributes parameter that described previously. Enabling the Pre-expire Passwords parameter will cause the global users to be created with passwords that are initially expired. If you set a value for PwdPreExpired using the Create Users Default Attributes parameter, that value takes precedence over one specified by enabling the Pre-expire Passwords parameter.
- CA Identity Manager Server, ETAUTIL, or other on-demand clients. If you create a single global user using the batch utility (ETAUTIL) or some other on demand LDAP client, these users will start out with expired passwords if you enable the Pre-expire Passwords parameter and do not otherwise specify a value of the PwdPreExpired property.

If you create a global user using an interactive client such as Provisioning Manager, whether the global user's password is initially expired or not is determined from the value of the PwdPreExpired property that provided when the global user is created. In Provisioning Manager, you control this value by selecting or clearing the check box labeled Force immediate expiration (one-time). Provisioning Manager automatically selects the Force one-time expiration (mark password as temporary) field if the Pre-expire passwords parameter is enabled. To disable this default behavior, clear the field before creating the global user.

Store User Passwords

Values: Yes (default) or No

Description: Controls whether the EncryptedPassword global user attribute is stored and whether %P% rule variables are supported.

By default the Provisioning Server encrypts the global user password and stores it in the provisioning directory as a global user attribute named EncryptedPassword. When you later attempt to create an account for that global user using an account template with the %P% expression for the password rule, then the Provisioning Server decrypts the stored EncryptedPassword value and provides it to the endpoint as the initial Password attribute for the account being created.

However, if you will not be creating any accounts using account templates with %P% rule expressions, then you can improve security by not storing these passwords.

Note: By not storing the EncryptedPassword attribute, you are only giving up %P% rule evaluation. You can authenticate users by using the global user password. When the Store User Passwords parameter is set to No, the Provisioning Server stores a one-way hash of the password to authenticate user passwords during the login.

Processes Parameters

The Processes configuration folder contains parameters that you can use to control the process behaviors on Windows provisioning servers.

Catch Program Exit Exceptions

Values: Yes (default) and No

Description: This parameter controls the behavior of the Provisioning Server when invoked program exits throw runtime exceptions. By default (yes), the exception is caught and the current operation fails with an uncaught exception error message. However, if you are developing new program exits you may choose to set this parameter to no and allow the uncaught exception to result in server termination, which provides more information about the exception. This parameter only affects common program exits of the DLL type.

Child Operation Thread Pool Size

Default Value: 200

Description: This parameter defines the maximum number of threads in the server-wide child operation thread pool. When the server decides to split up a single operation into multiple sub-operations, those sub-operations are carried out by the threads in the child operation thread pool. The larger you make the value for this parameter, the more work the Provisioning Server attempts in parallel.

Currently the Provisioning Server only uses the child operation thread pool to carry out the multi-account search and multi-account update functions that are submitted by the Web interface. For synchronization and propagation operation, regardless of which client submits these requests, and even though they also spawn child operations, the child operations are carried out in series in the main operation's thread.

This parameter does not affect the primary server thread pool that is used for processing separate requests received from client applications. This thread pool size is controlled by the SLAPD parameter that is called threads in the eta_slapd.conf file. See on the Provisioning Manager help for editing parameters in this configuration file.

Important! Changes to this parameter do not take effect until the Provisioning Server service is restarted.

Parallel Propagation

Values: Yes (default) and No

Description: This parameter controls whether account passwords updated as part of a global user to account password propagation are carried out in parallel or sequentially. By default, account passwords are updated in parallel, and the degree or parallelization is controlled by the Processes/Child Operation Thread Pool Size parameter.

This parameter has no effect on requests from clients that carry out the account updates explicitly. It only affects those clients that direct the Provisioning Server to update a global user password and propagate that change immediately to all synchronized account passwords.

Processor Parameters

The Processes configuration folder contains parameters you can use to control values related to the use of CPU processors. These parameters control operating system values that are process-wide and as such affect the entire Provisioning Server service. This is specifically relevant if you install any additional backends into the slapd process that runs your Provisioning Server.

Important! Changes to these parameters do not take effect until the Provisioning Server service is restarted.

Process Affinity Mask

Default Value: 0 (no restrictions)

Description: Specifies the process affinity mask for the Provisioning Server service process. The process affinity mask is a bit vector in which each bit represents the processor of a multiprocessor server on which the threads of the process are allowed to run.

The value 0 (default) signifies no restrictions.

The values 1, 2, 4, 8, 16, 32, 64, or 128 restrict the threads to running on the 1st, 2nd, 3rd, 4th, 5th, 6th, 7th, or 8th processor, respectively.

For example, the values can be combined so the value 5 (1+4) can be used to allow running on processors 1 and 3.

Process Priority

Default Value: 0 (use system default priority)

Description: Specifies the scheduling priority of the Provisioning Server service process.

The value 0 (default) uses the system default.

The only other recommended value is 16384, indicating below-normal priority. This value should be used when the Provisioning Server runs on the same server as its provisioning directory. This effectively raises the priority of the provisioning directory and so increases over-all server performance.

Provisioning Directory Parameters

Entry Count Attribute

Values: dxEntryCount (if CA Directory is being used for the provisioning directory) or clear (otherwise)

Description: This attribute is used in queries that are sent to the provisioning directory to check whether size limits will be reached, but only if the client is not requesting partial results.

FIFO Entry Order

Values: No (default) or Yes

Description: Allows the Provisioning Server to work with directories that return search results in the same order they were written during standard one level search operations. This parameter allows backward compatibility support, so that Notifications work with provisioning directories other than dxgrid.

If this parameter is set to No, the Provisioning Server uses LDAP controls to instruct the provisioning directory to return records from the notification database by sequence number.

Some directories do not support this control mechanism. When working with other directories, set this parameter to yes to disable this control.

Search Parameters

The Search configuration folder contains parameters that you can use to control search behaviors.

Allow Partial Results

Values: Yes (default) or No

Description: Allow search requests to return less than the full number of matched entries when the search size limit is reached. If you set this parameter to No, or if the client fails to request partial results, partial results are not returned and clients receive a size limit exceeded error when the size limit is reached.

For example, if partial results are permitted, the search limit is 200, and the search found 5000 entries, 200 of them are returned. If partial results are not permitted, this search would return *no* results.

Note: The Provisioning Server does not define *which* 200 of the 5000 entries are returned. The Provisioning Server does not necessarily return the *first* 200 entries, either alphabetically or using any other ordering method.

Two settings are required to activate Partial Results:

- The domain configuration parameter Allow Partial Results that described here.
- A selection in the Provisioning Manager GUI Search Preference control.

The effective partial result setting is the combination of these two settings. A partial result is returned only if Allow Partial Results is set to Yes *and* the GUI Search Preference control has Show Partial Result Lists selected.

The Provisioning Server is most efficient when partial results are not returned. When partial results are not needed, the Provisioning Server can report quickly when a search would exceed the size limit. However, if necessary to return the number of entries indicated by size limit, this will add processing load to the provisioning directory, Provisioning Server and client application. This load will take away from the processing load available to other users' queries or modifications.

Max Scope Filter Objects

Default Value: 10

Description: The maximum number of objects that will be placed into a search filter during scoped searches. When a search of a container is initiated by an administrator who has access to only some of the objects in the container, the Provisioning Server augments the client-supplied filter with a scope filter that restricts the objects that are returned to those for which the administrator has access. However, if the administrator has access to more than Max Scope Filter Objects objects, this is deemed too many to be placed into the filter that will be sent to the provisioning directory and/or managed endpoint and the Provisioning Server uses its backup algorithm. In this case, the Provisioning Server will ask for all objects the client requested and then discard those that the scope filter would have excluded.

Search Size Limit

Default Value: 0 (unlimited)

Description: The maximum number of entries that are returned by the Provisioning Server in a search request. The effective size limit for a search is the smallest of this value, the SLAPD size limit parameter, and the client-provided size limit operation parameter.

Limiting the number of entries returned in searches is important for good interactive performance of the Provisioning Server. If the effective size limit is too large, poorly formed search requests may return long lists of global users, accounts, or other objects that are not easy for administrators to work with. Conversely, if the effective size limit is too small, it may limit the administrator's ability to browse provisioning roles or other objects whose number is more moderate than that of global users or accounts.

There are three ways to control the effective size limit of a search request:

- The SLAPD *sizelimit* parameter in the following files:

  PSHOME\Data\im_ps.conf
  PSHOME\Data\im_ccs.conf

  This value is set to 0 (unlimited) by default and controls all LDAP servers running in the Provisioning and Provisioning Connector server services, respectively. There should be no need to change this parameter. Doing so would limit operations like exploration that perform searches of accounts in managed directories and relies on being able to receive all accounts present in a single container in a single search request.

- The Domain Configuration parameter Search Size Limit

  This value is also set to 0 (unlimited) by default. It controls the maximum number of entries that are returned to provisioning clients in a single search request. If you set this to a non-zero value (500), this will prevent any client from being able to receive more than 500 entries from the Provisioning Server in a search request.

  Set this parameter only if you can be sure that no clients require receiving more than this number of entries from any search.

  You should leave the Search Size Limit configuration parameter as 0 (unlimited) on any interactive or mixed-use Provisioning Server. If you have an environment where certain Provisioning Servers are dedicated to interactive use and other Provisioning Servers are available for batch activity, you may want to set the Search Size Limit from 500 through 1000 on the interactive servers only. Use the Add Specialization menu item to set a server-specific Domain Configuration parameter.

- The Provisioning Manager's size limit preference setting

  To change this preference, select File, Preferences, click the Search tab, and change the value in the Limit on Returned Items for a Search field.

  The Provisioning Manager preset return limit is 500. Each user can increase or decrease this preference, but increasing the value above the server's Search Size Limit value has no effect because the server's effective size limit is the smallest of the three size limit controls.

If a search operation encounters a search-limit failure, assuming that you enabled the retrieval of partial results, in some cases the number of items that are displayed may be different than the actual search limit because a single search operation (from the perspective of the user) might require several searches (from the perspective of the Provisioning Server).

Depending on how a client combines the results of the multiple search operations (for example, through a union or intersection) when displaying the results, the net display may contain more or fewer items than the search-limit. In all cases, an error message is displayed informing you that the results were truncated due to the search limits.

Servers Parameters

The Servers configuration folder contains read-only parameters that identify which Provisioning Servers are installed for your domain. After a default installation, a single server is listed.

If you install alternative servers for failover or load-balancing, multiple servers appear in this list. For each server listed, read-only parameters identify the Build, Patch and Version numbers for the Provisioning Server software. An additional parameter identifies whether the FIPS 140-2 encryption feature is enabled for that Provisioning Server.

Note: The server names listed here are the same server names that are used when creating specialization parameters. These are also the names you should use for the server parameter in the csfconfig command-line utility when creating specializations for connector server configuration objects.

Statistics Parameters

The Statistics configuration folder contains parameters you can use to control how statistics are maintained by the Provisioning Server. Most objects stored in the provisioning directory have statistic attributes to record when and by whom the object was created; and when and by whom the object was last updated. These statistics are displayed on the Statistics tab of the respective objects' properties.

Enabled

Values: Yes (default) or No

Description: When disabled, statistic attributes on objects in the provisioning directory are not updated by the Provisioning Server as those objects are created or updated. This can improve performance during large scale changes or in installations where maintaining creation and update statistics is not necessary.

Certain statistics on global users such as the password update date and time, and suspension update date and time are required for correct operation of server functions. These statistics are updated even when the Statistics/Enabled configuration parameter is set to No.

Node Stats from Connection

Values: No (default) or Yes

Description: Use the client node name that is taken from the LDAP connection object when recording node statistics. The default (No) behavior is to take the node name that is provided by the client application.

The Node statistic that is displayed on the Statistics tab of the Global User property page and other objects' property pages is not always updated when other statistics such as the date, time, userid, and username are. This behavior is a result of the way the Node name is determined. By default, the node name must be provided by client applications in their requests. If the clients fail to do so, or the client submits a high-level operation, such as Synchronize, that spawns child operations to carry out the individual object updates, the server has no Node value to use to update that statistic. To rectify this problem, use the Node Stats From Connection configuration parameter. Change it's setting from the default No to Yes to select the alternative Node statistic algorithm.

The alternate algorithm uses the Node information that is obtained from the LDAP connection, which identifies the host that was the immediate client sending the request. This is often the same as the originating client, but can be another system.

For example, requests that originates from the CA Identity Manager Server would be recorded as originating from the computer where the CA Identity Manager Server is running, not the computer where the administrator is running a web browser. Also, if the clients connect to a dXRouter process for a failover or load-balancing between replicated Provisioning Servers and have the dXRouter send the request to the Provisioning Server, you should not enable the Node Stats From Connection parameter. It will result in all Node statistics indicating the router system.

Synchronization Parameters

The Synchronization configuration folder contains parameters that you can use to choose from alternative variants of the Provisioning Server's synchronization functions.

Automatic Correlation

Values: No (default) or Yes

Description: Enable the alternative User Synchronization behavior whereby an attempt to update an existing, uncorrelated account triggers an automatic correlation of the account to the global user prior to the update of the account. If the parameter is No (default), the attempt to update the account will fail with a message indicating the account has not yet been correlated to this global user.

Remove Account Template Values from Accounts

Values: Yes (default) or No

Description: When Yes, the Weak Synchronization algorithm will consider that capability account values (for example, account group membership) prescribed by an account template should be removed when that account template is removed from an account. Set this parameter to No to restore the prior Weak Synchronization behavior where account attribute values are never removed when synchronizing an account with its weak-synchronization account templates. This parameter only affects multivalued attributes. String, integer, or Boolean single-valued attributes are only increased in capability by weak synchronization.

Only certain multivalued attributes that are designated as SyncRemoveValues attributes are affected by this feature. Consult the eTaCapability.txt file for a list of which multivalued capability attributes may have values that are removed by the SyncRemoveValues feature that described here.

To generate the eTaCapability.txt file, use the following command:

```
PSHOME\bin\dumpptt –c > eTaCapability.txt
```

Exclude Nested Roles on User Role Removal

Values: No (default) or Yes

Description: Enables the alternative user removal synchronization behavior that excludes nested roles from evaluation. Nested Roles are roles that include other roles. Enabling this option has performance benefits as nested role evaluation is an expensive operation.

Important! Do not enable this option if nested roles are in use. Accounts may be deleted unexpectedly from the user.

Force Single Account Across Multiple Containers

Values: Endpoint types that are separated by semicolons

Description: On some hierarchical endpoints, creates one account for a certain endpoint instance when a global user's account templates specify the same account name in different account containers (on same endpoint). In this case only one account is created despite the account container differences.

This behavior can be useful if the assigned account templates nominate different account containers on the same endpoint where you only want to create one account in one of these account containers.

Note: If you enable this option, be sure to also enable the Use Existing Account synchronization setting.

Remove Deleted Account Template Attribute Values From Accounts

Value: Account attributes that are separated by semicolons

Description: Select a list of multi-valued capability attributes whose attribute values are removed if the corresponding values are removed from the Account Template during Weak Synchronization. Specify the attribute names as the parameter which is separated with a semicolon. (For example: eTADSmemberOf;eTETCGroupMembers).

Note: This feature is only applies when Remove Account Template Values From Accounts is enabled.

Use Existing Accounts

Values: No (default) or Yes

Description: Enable the alternative User Synchronization behavior whereby a global user's set of assigned account templates (through assigned provisioning roles) will only attempt to prescribe one account that is correlated to the global user on any particular managed endpoint. This behavior can be useful if some accounts already correlated to the global user are named differently or are in different containers than what is prescribed by the account templates included in the global user's provisioning roles and only one account is needed or allowed. If the parameter is enabled and multiple account templates for one endpoint prescribe different names and/or different containers for the account only one account will be created.

If a global user already has multiple accounts on a single endpoint, the User Synchronization function (when Use Existing Accounts is set to Yes) attempts to figure out which account is required by which account templateaccount template. This is done through a heuristic that attempts to handle situations where a user's provisioning roles do in fact prescribe multiple accounts on one endpoint.

For example, if global users have two accounts (A1 and A2) on endpoint E and their provisioning roles indicate that they should have one account on endpoint E through account template AT1 and one account on endpoint E through account template AT2, User Synchronization pairs each account template (AT1 and AT2) with one of the existing accounts. The pairing is done with the following heuristic:

- Match account template with an account with exactly the DN specified by the account template.
- Match account template with an account already belonging to the specified account template. If more than one account matches, pick the first one.
- Match account template with an account whose endpoint type-specific account name attribute matches the global user's name. In some endpoint types, for instance Active endpoint, the account name is represented by an attribute of the account whereas the name as seen when you list the account is a display name (a full name). This rule accounts for such endpoint types.
- Account with name value matching the name value specified in the account template. That is, it matches an account with the right name but the wrong container. If there is more than one matching account, pick the first one.
- Pick the first account.

Note: When Use Existing Accounts parameter is set to No, only the first of these rules (exact matching that is based on account DN) is applied.

Continuing with the example, if the previous rules resulted in pairing both account template AT1 and account template AT2 with account A1, then User Synchronization would correct the accounts for this user by doing the following:

Deleting account A2 (assuming the administrator selected the Delete extra accounts or extra account templateaccount template assignments option of User Synchronization); and

Assigning either account template AT1 or AT2 to account A1 that was not already assigned.

These rules ensure that User Synchronization (with Use Existing Accounts enabled) never attempts to create additional accounts on an endpoint where a user already has an account. If your business requires you to create multiple accounts for your users on a single endpoint from provisioning roles, do not enable this configuration parameter. For more information about synchronization, see the Administration Guide.

Transaction Log Parameters

The Transaction Log configuration folder contains parameters that you can use to control transaction logging, also known as a Provisioning Server trace logging. This is the log you use to monitor activity performed by the Provisioning Server while it processes requests received from its client applications.

Enable

Values: Yes (default) or No

Description: Set this parameter to No to completely disable the logging of information to the server trace log. Typically, you control the amount of information you want logged using the Level parameter. However, even at level 0 some important items are logged to the server trace log. To disable these items from being logged, set the Enable parameter to No.

Enable/Configuration

Values: Yes (default) or No

Description: This parameter enables or disables logging of diagnostic output from the Provisioning Server Configuration subsystem. The configuration subsystem checks every 10 minutes (by default) to see whether any of the configuration parameters have been changed. Each time this periodic refresh occurs, a line such as the following is written to the server trace log: ETA::Configuration update completed. No changes found.

Alternative messages are written if actual changes were found. To suppress all of these messages from the server trace log, set this configuration parameter to No.

Enable/Connector Server Framework

Values: Yes (default) or No

Description: Enables/disables logging of the diagnostic output from the Provisioning Connector Server Framework (CSF).

Enable/LDAP

Values: Yes (default) or No

Description: This parameter enables or disables logging of diagnostic output from the Provisioning Server LDAP subsystem. The LDAP subsystem manages the communications between each Provisioning Server and other LDAP servers, including the provisioning directory and Connector Servers

File Name

Default Value: etatrans

Description: This parameter defines the transaction log's base file name. The suffix *YYYYMMDD-HHMM.log* will be appended to this base file name to build the log file name. You can change this parameter to use a different base file name in the *PSHOME*\Logs folder or to relocate the log file to another folder on your server.

The value of this parameter can be any of the following:

- Simple base name (for example, etatrans), the log is created in the Logs folder in the folder where you installed the Provisioning Server. By default, this makes the log file named C:\Program Files\CA\Identity Manager\Provisioning Server\Logs\etatrans*YYYYMMDD-HHMM.log*.
- Relative path (for example, ..\Logs\etatrans), the log path name will be relative to the current directory of the Provisioning Server service (PSHOME\bin). For the example given, this will result in the same pathname as before (C:\Program Files\CA\Identity Manager\Provisioning Server\Logs\etatrans*YYYYMMDD-HHMM.log*).
- Absolute path (for example, D:\ProvisioningData\Logs\etatrans), you can specify an alternative drive for your log file. For this example, the resulting log file would be *D:\ProvisioningData\Logs\etatransYYYYMMDD-HHMM.log*.

The Provisioning Server switches to a new log file every day, every time the Provisioning Server restarts, and any time the log file size exceeds 100 Megabytes.

Level

Values: 0 through 7 (default)

Description: This parameter lets you set the level of logging for the Server Trace log. Valid values are:

Value	Description
0	No trans logging
1	Log external/child errors
2	Log external operations
3	Log child operations
4	Log informative messages
5	Log DSA (Directory Service Agent) errors
6	Log DSA operations
7	Log search operations

Note: Alert log entries are logged at all logging levels (1 - 7).

After installation, the log level is set to the maximum value (7). This ensures that any problems during or immediately after installation are logged. After installation, you may select alternative logging levels to meet your logging requirements. Many customers run with level 7 for maximum information when the problems are reported by users. Other customers select a more modest level such as level 3 that reports failures without much of the internal tracing information associated with the processing of requests. Another useful level is level 6 that removes the many search operations that could dominate the log while maintaining all other information.

Log user-friendly Attribute and Object Class Names

The CA Identity Manager Provisioning Server currently logs attribute values in its server trace log (etatransYYYYMMDD-hhmm.log) as it logs the attributes in Add and Modify operations, listing the LDAP attribute names and the LDAP objectClass values. For DYN connectors, the LDAP attributes and object classes are generic names (such as eTDYN-str-multi-01, eTDYNObject001) which are not that meaningful. For release 12.5, these log entries are expanded to list the LDAP attribute names and object class values, and the user-friendly names taken from the metadata.

Chapter 3: Provisioning Servers on UNIX

The Provisioning Server, the C++ Connector (SuperAgent) Server and various utilities that work with these servers can run on either Windows or UNIX platforms. For the most part the servers and utilities behave the same and therefore are documented with a single description. This appendix describes the major differences based on the operating system.

This section contains the following topics:

No UNIX GUI Clients or Utilities

Other than installation and web clients, no graphical user interface (GUI) clients or utilities exist on UNIX. For example, Provisioning Manager (etadmin.exe) with its endpoint type-specific GUI plug-ins run only on a Windows system and access the UNIX Provisioning Server remotely. Also, the pwdmgr utility has a different format on UNIX, without a GUI, and must be run from Windows.

Command Line Examples

The CA Identity Manager documentation includes examples of invoking commands from a command prompt. On Windows, this prompt is a Command (DOS) Window; on UNIX, the prompt use one of various shells. Except where noted, you can assume that the examples are for Windows. You can understand the environment variables and path separators that would be necessary to use a given commands on UNIX by replacing Windows pathnames such as

```
%VARNAME%\data\im_ps.conf
```

with

```
$VARNAME/data/im_ps.conf
```

Also, on UNIX nearly all directories (folders) and file names are in lower-case. Since case is significant in file names on UNIX but insignificant on Windows, some examples in the documentation that refer PSHOME\Data function correctly on Windows, even though new installations name that folder data instead of Data. If you are unsure about the case used for a directory on UNIX, use the ls command to locate the exact directory name.

When UNIX examples are given, they apply to any command shell, but were tested to work with the Bourne shell (/bin/sh).

Also, note that quoting rules are different in Windows and UNIX command interpreters. Consult the respective interpreter documentation for how to quote or escape data that requires quoting or escaping.

Libraries and Executables

Libraries and executables differ on UNIX and Windows as follows:

Windows	**UNIX**
Libraries are named LibraryName.DLL (mixed-case and dll suffix) and typically installed into a folder such as PSHOME\bin.	Libraries are named liblibraryname.so (lower-case, lib prefix and so suffix) and installed into a directory such as $PSHOME/lib.
Executable programs are called ProgramName.EXE (mixed-case, exe suffix) and installed into PSHOME\bin.	Executable programs are called programname (lower-case, no suffix) and installed into $PSHOME/bin.
Scripts are named Script.bat (mixed-case, bat suffix).	Scripts are named script or script.sh (lower-case, optional sh suffix).
Message files are named FileName.DLL (mixed-case and dll suffix) and installed in PSHOME\bin.	Message catalogs are named filename.res (lower-case and res suffix) and installed in PSHOME\data.

Registry Access

On Windows, configuration information is stored in the Windows registry and edited with a native Windows utility such as regedt32 or regedit. On UNIX, the registry is emulated as files in the file system (/opt/CA/SharedComponents/EnterpriseCommonServices/registry). Protect these files as you would the contents of other configuration files. Installation will protect the CA Identity Manager keys by default. Only imps group users can read them and only the imps user can write them.

To dump out the entire registry, you can use the command eCSoption /r. To view, modify or delete specific registry settings that are specific to CA Identity Manager, use the CA Identity Manager utility eta-env.

For example to view a registry setting, you could use these commands:

```
eta-env action=get name=”etrust_bindtodb_need_tls” type=int
eta-env action=get name="logging/caldap_client_logfile"eta-env action=get
name="/enterprise_common_services/installpath"
```

and to set a registry value

```
eta-env action=set name=”etrust_bindtodb_need_tls” value=1 type=int
eta-env action=set name="logging/caldap_client_logfile" value=my_file_name
```

Names that begin with / (slash), are relative to:

```
[HKEY_LOCAL_SYSTEM]\SOFTWARE\ComputerAssociates
```

Simple names, without a / (slash), are relative to:

```
[HKEY_LOCAL_SYSTEM]\SOFTWARE\ComputerAssociates\Identity Manager\Provisioning
Server
```

So, these two command invocations

```
eta-env action=get name=”/Identity Manager\Provisioning
Server/etrust_bindtodb_need_tls” type=int
eta-env action=get name=”etrust_bindtodb_need_tls” type=int
```

refer the same configuration parameter.

Note: The registry path of “Identity Manager\Provisioning Server” is set in the $ETAHOME/data/reg_path.conf.file. The preceding eta-env commands are valid on UNIX and Windows; however, Windows has multiple eta-env.exe commands installed. If you run the eta-env.exe command from the provisioning server installation, it consults the reg_path.conf file from that installation and the registry keys and values are as shown in this section with UNIX. However, if you run the eta-env.exe command that is installed with the provisioning manager installation, it consults the reg_path.conf file from that installation and the registry keys and values being accessed are those under “Identity Manager\Provisioning Manager” instead.

Parser Tables

Parser table files are compiled files with suffix ptt that are installed in PSHOME\data on Windows ($PSHOME/data on UNIX). They are read by the Provisioning Server and various utilities, such as dumpptt, etautil, showpttdit, and schemagen. The format is a platform-neutral format so that it can be freely copied between Windows and UNIX systems.

UNIX Services for Provisioning

The Provisioning Server (im_ps.exe) and C++ Connector Server (im_ccs.exe) are typically run as services on Windows. Thus you would typically start and stop them on Windows by going to the Services application. Alternatively you could start and stop them from the command line with commands such as the net start im_ps and net stop im_ccs.

On UNIX, the Provisioning Server executable is called slapd and both servers normally start automatically through control files installed in /etc/rc*.d. To view, start, and stop the services manually, you can use commands such as imps status, imps start im_ps, and imps stop im_ccs. The command “imps” is also available as the command “eta” for backwards compatibility with prior eTrust Admin installations.

Working with Hung or Crashed Servers

On Windows, a crashed server may cause information to be written to the system’s drwtsn32.log file, a file that CA Customer Support may ask you to send to help analyze the problem.

On UNIX, a crashed server creates a core file in $PSHOME/bin unless you have configured your server not to generate core files. If a core file is generated, please do not send it to CA unless instructed to do so. Instead, run the command pstack core > pstack.txt to capture the stack traces of all threads running within the crashed application. This output is valuable in diagnosing the failure.

On Windows, a hung server (one where one or more requests did not run to completion) can only be debugged using the provisioning server trace log (*PSHOME*\logs\etatran*yyyymmdd-hhmm*.log) in conjunction with the analyzelog utility. You will generally be asked to capture the provisioning server trace log (at logging level 7 if at all possible) and CA will use “analyzelog” to locate operations that have not yet completed. The C++ Connector Server trace log (satrans*yyyymmdd-hhmm*.log) and sometimes other logs are also useful to collect.

On UNIX, capturing the provisioning server trace log and running analyzelog is still useful. But another option that often provides additional information is once again the pstack command. Locate the process ID (pid) of the hung service by reading the contents of the file $PSHOME/data/pid/servicename.pid, and then issue the command pstack pid > pstack.txt to capture the stack traces of all active threads within the running process. Please include this output file along with the provisioning server trace logs.

Scheduling Periodic Actions

The UNIX cron command is useful for scheduling periodic tasks such as script invocations. This includes invocations of etautil commands (for checking or performing synchronization of accounts or users or performing refresh explore or correlate operations) and invocations of other utilities, such as etadailybatch and etacreateouglobalgroups. However, using etautil for invoking the period explore or correlate operations is no longer recommended. Instead you can configure explore/correlate tasks directly within CA Identity Manager.

Passwords on Command Lines

In UNIX, command-line arguments are public to anyone who can use the ps command on the UNIX system. Therefore, you should never supply a password or other sensitive information as a command-line argument. Each CA Identity Manager command accepts input from a file so you can avoid entering data on the command line. Often, the command-line parameter is still allowed for backwards compatibility with Windows.

Server Event Logging Destinations

Some of the server event logging destinations behave differently on UNIX from how they behave on Windows. In particular, the System log destination logs to syslogd on UNIX and to the Windows Event Viewer on Windows. Also, the eTrust Log destination should be avoided on UNIX. It logs to a local file that cannot be viewed locally since there is no UNIX version of the eCS Log Viewer utility. It is not recommended that you run the eCS Log Daemon on UNIX to export the log contents to a remote Windows system because you cannot control who can view the log remotely. These same logging destinations also apply to directory-level logging.

Program Exit Definitions

When defining a common program exit, you enter fields that are interpreted by the Provisioning Server, which invokes the program exit routine. In entering these fields, consider the operating system (Windows or UNIX) of that domain's Provisioning Servers so that these fields work on that operating system. If the domain includes Windows and UNIX Provisioning Servers, be sure that these fields work on both operating systems.

For the SOAP program exits, the WSDL can be specified by a URI or its fully qualified name as seen from the Provisioning Server or by a pathname relative to PSHOME\bin ($PSHOME/bin on UNIX), which is the current working directory of the Provisioning Server.

For the DLL program exits, the library name can be a fully qualified name or it can be a common name with or without the lib prefix or the .dll or .so suffix. When only one Provisioning Server exists for a domain, no restrictions exist for how you specify the library name. But when a domain has multiple servers, the library name must be valid for all the servers and since UNIX and Windows have different path syntaxes, files systems, prefixes, and suffixes, the library name should be defined as a common name without any prefix or suffix.

Thus the preferred way to define a program exit object for the CommonExit sample exit is to enter the string CommonExit for library name (in this exact case). The UNIX server will search LD_LIBRARY_PATH for a library named libCommonExit.so. On Windows, this will locate CommonExit.dll by searching the PATH environment variable.

C++ Connector Server on Solaris

The C++ Connector Server installed on Solaris can manage only Solaris UNIX ETC and ACC endpoints. For all other Connectors, install the C++ Connector Server on a Windows system and register it with the Provisioning Server installed on Solaris. During installation, specify that this Connector Server is your default C++ Connector Server.

Chapter 4: Program Exits

This section contains the following topics:

Program Exits Overview

Program exits let you write software that executes during certain Provisioning Server actions. Program exits let you reference custom code from in the Provisioning Server process flow, extending the framework of the Provisioning Server to allow additional functionality that changes or augments standard behavior. Numerous exit points are available where custom code can be referenced, depending on the type of object. For example, you may want to install some files on a system every time a UNIX account is created. You could write a program exit that performs the file creations, and specify that it be run whenever a UNIX account is created.

There are two types of program exits:

- *Common Exits* are executed from the Provisioning Server core infrastructure.
- *Native Exits* are executed from the managed endpoint types.

The type of program exit is determined by where it is handled, not where it is referenced.

Note: For information about native exits, see the endpoint type-specific *Connector Guide*.

Program exits are implemented as separate objects, allowing you to define the necessary exits and associate them at the points where they need to be referenced. The following objects reference program exits:

- Common Configuration Objects
- Provisioning Roles
- Account Templates
- Endpoints

Each of these objects can reference multiple program exits, including multiple exits of the same type. For example, a directory can reference two exits that handle routines to be executed before creating an account.

Ordering of Program Exit Invocations

A single request processed by the Provisioning Server may make multiple program exit invocations. The order in which these program exits are invoked depends on:

- The type of program exit
- The location of the program exit reference
- The priority number assigned to the program exit reference

Each program exit type identifies a place in the Provisioning Server's control flow where that particular type of program exits gets a chance to affect the Provisioning Server's behavior. Therefore, to understand the order in which different program exit types are invoked requires understanding how requests are processed.

For instance, a single request to the Provisioning Server might change a global user password and then propagate that password to one or more of that user's accounts. The processing of this request is done as a high-level global user operation that spawns separate account operations.

This results in invoking the program exits in the following order:

1. PRE_CHANGE_GLOBAL_USER_PWD
2. PRE_CHANGE_ACCOUNT_PASSWORD
3. POST_CHANGE_ACCOUNT_PASSWORD
4. PRE_CHANGE_ACCOUNT_PASSWORD
5. POST_CHANGE_ACCOUNT_PASSWORD
6. POST_CHANGE_GLOBAL_USER_PWD

In some cases, multiple program exit types apply to the same object class (PRE_CHANGE_GLOBAL_USER_PWD and PRE_MODIFY_GLOBAL_USER) and could potentially be applicable to the same request (a single global user modification that changes both the password and full name, say). In such a case, all exits of one of these exit types will be called before all exits of another of these exit types. But the order is unspecified and you shouldn't assume that the ordering will remain unchanged in future versions of the Provisioning Server.

For a single exit type, sometimes you have a choice as to the class of object on which you define the program exit reference. In particular, references to some exit types that affect global users can be defined on a provisioning role (affecting only users in that role) or on the common configuration object (affecting all users in the domain). If you define exit references on both kinds of objects, then the Provisioning Server invokes the ones defined on the provisioning roles before invoking the ones that are defined on the common configuration object.

Similarly, references to some exit types that affect accounts can be defined on an account template (affecting only accounts assigned to that account template) or on the endpoint (affecting all accounts in the endpoint). If you define exit references on both kinds of objects, then the Provisioning Server invokes the ones defined on the account templates before invoking the ones that are defined on the endpoint.

Finally, exit references of the same type and defined on the same class of object are invoked in priority order using the priority number you assigned when you created the program exit reference, such as priority 1 first, priority 2 second, and so on. Two program exit references of the same priority are invoked in unspecified order.

Basic Structure of Program Exits

Program exits are referred to as pre-operation or post-operation (that is, some operation that Provisioning Manager is used to performing). Program exits have a single common interface for their calling structure. This interface consists of a single argument and a single return value. The single argument is an XML buffer representation, encoded in Unicode Transformation Format 8 (UTF-8), of the object being acted upon combined with any custom information from the definition of the exit. The return value is status information about the result of the program exit execution as well as any documented custom information that is required for a particular exit.

Define Common Exits in the Provisioning Manager

You can define a common exit that will be executed in the Provisioning Server core infrastructure by using the Program Exit property sheet.

Note: To define a Native Exit to be executed in a managed endpoint type, see the specific *Connector Guide*.

To define a common program exit

1. Click Endpoints.
2. Select Common Program Exit from the drop-down list in the Object Type field.
3. Click New. The Common Program Exit dialog appears.
4. Fill in the name and description of the exit to be invoked on the Program Exit tab. If the Disabled box is selected, the program exit will not be invoked, even if it is referenced in another object.
5. Specify whether the Provisioning Server uses the Simple Object Access Protocol (SOAP) or a Dynamic Link Library (DLL) file to invoke the exit on the Common Parameters tab. You can specify that the information be sent securely by selecting SSL Enabled.

6. Enter the path that points to the DLL file or the address of the SOAP service in the Location field. In the Method field, provide the name of the exported function in the DLL file or the name of the function defined by the SOAP service.

   For DLL program exits, you can enter for location either a full path, such as:

   ```
   c:\yourfolder\yourlibrary.dll
   ```

   or you can enter just the common name of the library

   ```
   yourlibrary
   ```

   If you enter just the common name, you can have more than one Provisioning Server for the domain, where the library does not have to appear at exactly the same path. This is important if the domain has a mix of Solaris and Windows servers, because these operating systems have different pathname syntax.

   If you provide just a common name (yourlibrary), the Provisioning Server will locate the library in the following way:

   - For Windows, locate yourlibrary.dll file on the Provisioning Server service's execution path as defined by the PATH environment variable. We recommend that you place the library in the PSHOME\bin folder which is known to already be on PATH.
   - For Solaris, locate the libyourlibrary.so file on the Provisioning Server service's library path as defined by the LD_LIBRARY_PATH environment variable. We recommend that you place the library into the $PSHOME/lib directory which is known to already be on LD_LIBRARY_PATH.

7. Select an Authentication Type to provide information for authentication data to be passed to the invoked program exit on the Authentication tab:

   - Select None to pass no authentication data to the exit.
   - Select Current User to pass the authentication data of the global user who is logged on at the time the exit is invoked.
   - Select Proxy User to select a specific global user that will be used for authorization of the operations.

   Note: When you select Proxy User, the information you provide must be that of a valid global user.

   - Select Other to enable the Authentication Details group field, which lets you select an arbitrary name and password. The exit code uses this information for authentication.

8. Click OK to complete the definition of the common exit.

Chapter 5: Common Program Exit Reference

Common program exits let you write software to run during certain Provisioning Server actions, extending the framework of the Provisioning Server with added functionality. Common program exits are called by the Provisioning Server during the processing of user-provisioning operations.

Native program exits are optional exits that may be present in some connectors where such facilities are available and their use is warranted (for example, the OS400 connector). Native exits are called from their respective connector plug-in running under the C++ Connector Server.

This section contains the following topics:

Program Exit Architecture

Program exits let you reference custom code from the Provisioning Server process flow. Many entry points are available where custom code can be referenced. In addition, you can invoke program exits as custom functions during policy rule evaluation so you can write custom logic to compute account attribute values. For example, to install some files on a system every time a UNIX account is created, you can write a program exit that creates the file, and indicate that the program exit be run whenever a UNIX account is created.

Program exits belong to the following types:

- *Common exits* are executed in the Provisioning Server core infrastructure.
- *Native exits* are executed in the managed endpoints. For more information about native exits, see the connector guide for the specific endpoint.

Where the program exit is handled determines which type of exit it is, not where the exit is referenced. Program exits are implemented as separate objects in the endpoint and are referenced in these objects, allowing you to define only the exits that are necessary and associate them where they need to be referenced.

The following objects reference program exits:

- Common configuration objects
- Provisioning roles
- Account templates
- Endpoints

Each object can reference multiple program exits, including multiple exits of the same type. For example, an endpoint can reference two PRE_CREATE_ACCOUNT exits.

Program Exit Hierarchy and Order

Program exits are serialized in the Provisioning Server process flow and are both hierarchical and ordered, as described below:

- In terms of a hierarchy, the exits are called as referenced from the following objects in the following order:
  - Common configuration objects
  - Provisioning roles
  - Account templates
  - Endpoints
- In terms of order, in a given hierarchy exits are called in a specified order.

An operation on a global user checks for exits to be invoked in the common object and all roles to which the global user belongs. Exits referenced by the common object are invoked before exits referenced by the provisioning role. Similarly, an operation on an account checks the account templates and the endpoints to which the account belongs for exits to be invoked. Exits referenced by the account templates are invoked before exits referenced by the endpoint.

Common Program Exit Structure

Common program exits are referred to in terms of “pre” or “post” in relation to an operation that the Provisioning Server commonly performs. Program exits have a single common interface for their calling structure. This interface consists of a single argument and a single return value. The input argument is an XML buffer representation, encoded in UTF-8, of the object being acted upon combined with any custom information from the definition of the exit. The return value is status information on the result of the program exit execution as well as any documented custom information that is required for a particular exit.

There are two types of common exits:

- DLL deployed
- SOAP executable

Program Exit Input Argument

Program exits have a single interface consisting of a single input argument, which is an XML buffer. All program exits are passed to the XML buffer with the following format:

```
<eTExitInvoke eTExitType={one of the exit types}>
   <{the objectclass of the object being processed}>
   <dn>{the full DN of the object}</dn>
   <name>{the name, that is, RDN value, of the object}</name>
   <{attribute type}>{attribute value}</{attribute type}>
   ...
   </{the objectclass of the object being processed}>
   <Authentication>
      <Type> </Type>
      <User> </User>
      <Password> </Password>
   </Authentication>
</eTExitInvoke>
```

For example:

```
<eTExitInvoke eTExitType=PRE_ADD_ACCOUNT>
   <eTSDKAccount>
      <dn>eTSDKAccountName=test1, eTSDKAccountContainerName=SDK Accounts,
         eTSDKDirectoryName=Team1, dc=Dev</dn>
      <name>test1</name>
      <eTSDKCity>Renton</eTSDKCity>
   </eTSDKAccount>
   <Authentication>
      <Type>GLOBAL_USER</Type>
      <User>{the DN of the global user}</User>
      <Password>{the password of the global user}</Password}
   </Authentication>
</eTExitInvoke>
```

For modify operations, the modify mode is specific in each tag. The possible modify modes are ADD, DELETE, and REPLACE. For example:

```
<eTExitInvoke eTExitType=PRE_MOIDFY_ACCOUNT>
   <eTSDKAccount>
      <dn>eTSDKAccountName=test1, eTSDKAccountContainerName=SDK Accounts,
         eTSDKDirectoryName=Team1, dc=Dev</dn>
      <name>test1</name>
      <eTSDKCity modify-mode=”replace”>Kirkland</eTSDKCity>
      <eTSDKDescription modify-mode=”delete”
         Old description</eTSDKDescription>
   </eTSDKAccount>
</eTExitInvoke>
```

The program exit parses this input argument to get the data it needs to perform its specific task.

If a program exit is defined to handle only a specific type of exit, it should check the eTExitType to make sure that it can handle that specific type. For example, if a program exit is designed to handle the exit type PRE_ADD_ACCOUNT, it should check eTExitType and should perform only its task, if the exit type is correct. If the exit type is not handled by the program exit, it should do nothing and should return a warning.

Input XML Buffer Authentication Type

Each input XML buffer may contain an optional authentication XML block. The format of the authentication XML block is always defined as follows.

```
<Authentication>
   <Type> </Type>
   <User> </User>
   <Password> </Password>
</Authentication>
```

The data in the authentication XML block depends on the type of authentication defined for the program exit. The following are the possible authentication types:

NONE

No credentials are passed to the method being invoked. Thus, the input XML buffer does not contain an authentication block.

GLOBAL_USER

The credentials of the currently logged on global user are passed to the program exit being invoked. The <User> tag contains the DN of the global user. The <Password> tag contains the password for that global user.

Note: The password is not encrypted.

PROXY

The credentials of a specific global user are passed to the program exit being invoked. The <User> tag contains the DN of the specific global user. The <Password> tag contains the password for that global user.

Note: The password is not encrypted.

OTHER

Indicates that the <User> and <Password> tags are program-exit specific. The <User> and <Password> tags can be any free form text. It is up to the program exits to define what these fields mean.

Program Exit Return Value

Program exits have a single return value, which is an XML buffer. Program exits must return an XML buffer, which has the following format:

```
<eTExitReturn>
   <eTExitReturnCategory> </eTExitReturnCategory>
   <eTExitReturnNative> </eTExitReturnNative>
   <eTExitLogMsg> </eTExitLogMsg>
   <eTExitContinue> </eTExitContinue>
   <eTExitCustom> </eTExitCustom>
   <eTPersistentFailure> </eTPersistentFailure>
</eTExitReturn>
```

eTExitReturnCategory XML

Requirement

This value is not required.

Purpose

Groups various native return codes into one of three categories for the purpose of simplifying process flow.

Valid Values

SUCCESS

WARNING

FAILURE

Default Values

If no value is specified, SUCCESS is assumed.

eTExitReturnNative XML

Requirement

This value is not required.

Purpose

Specifies the return value from the native program exit call.

Valid Values

This value is a string representation of what occurred.

Default Values

None.

eTExitLogMsg XML

Requirement

This value is not required. It is, however, highly recommended to enter a value for failure or warning responses:

- Without eTExitLogMsg value, the server will send the eTExitReturnNative code for logging.
- Without eTExitLogMsg and etExitReturnNative values, the server will make up a generic message indicating no message present and that there was an error/ warning.

Purpose

Specifies a string value that the native program exit wants the server to log.

Valid Values

This value will be a UTF-8 string.

Default Values

None.

eTExitContinue XML

Requirement

This value is not required.

Purpose

Specifies whether to continue the process flow after the return from the program exit. This value overrides the default behavior. See Default Values.

Valid Values

TRUE - Continue Execution.

FALSE - Stop Execution.

Default Values

The default values are based on the eTExitReturnCategory attribute.

TRUE - If eTExitReturnCategory is SUCCESS or WARNING.

FALSE - If eTExitReturnCategory is FAILURE.

eTExitCustom XML

Requirement

This value is not required.

Purpose

For common program exits, this value is reserved for future use.

For native exits, this value is connector-specific.

Valid Values

Any valid XML document.

Default Values

None.

The program exit parses this input argument to get the data it needs to perform its specific task.

eTPersistentFailure

Requirement

This value is not required.

Purpose

Used only in responses from IMS Notifications, which share with program exits the same XML buffers for encoding requests and responses. A persistent failure is a notification that is rejected based on a problem in the content (likely a programming error) rather than based on some retry-able situation.

Valid Values

TRUE - Indicates a persistent failure.

FALSE - Indicates a transient failure, one that might succeed later if retried.

Default Values

FALSE

Common Exits DLL Interface

DLL deployed program exits must export the function with the following prototype:

```
int function_name(
   char Input_XML,
   char *Return_XML,
   int *Return_Buffer_Length)
```

The following list describes the parameters for the DLL deployed program exit prototype:

function_name

Name of the program exit. One DLL can export multiple program exits, where each program exit is an exported function with the prototype defined above.

InputXML

Character buffer in UTF-8 format. It contains the XML buffer that the Provisioning Server passes to the program exit.

ReturnXML

Character buffer in UTF-8 format., Iit is an empty buffer that the Provisioning Server passes to the program exit for it to send a return value back to the Provisioning Server. The size of the buffer is passed to the program exit is the Return_Buffer_Length parameter.

Return_Buffer_Length

Both an input and output parameter:

- On input, Return_Buffer_Length indicates the maximum length, in characters, that the Return_XML buffer can contain. The program exits must not exceed this length when building the return XML buffer.
- On output, Return_Buffer_Length contains the actual length of the return XML buffer the program exit built. That is, after the program exit builds the return XML buffer, it sets Return_Buffer_Length to the actual length of the buffer being returned.

Common Exits SOAP Interface

SOAP-deployed program exits must present an external interface like the following prototype:

```
char * function_name ( char * Input_XML )
```

The following list describes the parameters for the SOAP-deployed program exit prototype:

function_name

Name of the program exit. The Web Services Description Language (WSDL) file that describes the program exit contains a definition of the function_name. This is also the name of the character buffer in UTF-8 format that is returned. This buffer is allocated by the referenced program, but must be cleared from the calling program.

Input_XML

Character buffer in UTF-8 format. Input_XML contains the XML buffer the Provisioning Server passes to the program exit.

The definition of this interface needs to be presented in the following ways:

- A way that the SOAP client can understand.
- A way that the SOAP server can understand.

The SOAP client relies upon WSDL to specify the interface. For a description of WSDL, see http://www.w3.org/tr/wsdl.html.

The SOAP server described here is the Apache SOAP server. The Apache SOAP server requires an XML document known as a Deployment Descriptor. The Deployment Descriptor indicates to the SOAP server what the interface to the SOAP program is. For a more complete description of deployment descriptors, see the Deployment Descriptors section in the *User Guide* at http://ws.apache.org/soap/docs/index.html.

An example of a SOAP exit can be found in the following folder:

```
Samples/ProgramExitSOAP
```

eTExitType

Exit types determine the circumstances under which an exit is called. A value is entered for eTExitType, in the input XML buffer passed to the program exit.

The exit types with ACCOUNT in their names can be common or native exits, meaning that common code and connector code can be triggered to process them.

All other exit types, however, must be common exits. It should also be noted that not all program exit types are referenced from the various object types.

Notes:

- In all cases, the name of the object being passed is sent. This is formatted in both DN and Common Name format.
- To have complete control over passwords, either at the global user or the account levels, you must provide exits both for create user/account and for change password user/account. In other words, for new global users (accounts), the change password exit is not called. For new global users (accounts), the password is passed in as part of the attribute for the create exit (for example, PRE_CREATE_GLOBAL_USER).

More information:

Valid Values for eTExitType (see page **75**)

Valid Values for eTExitType

The following values are valid for eTExitType:

PRE_ADD_ACCOUNT

The account information that is being passed to the create account request is also passed to this program. Unlike the Modify operation, the password is passed to the Create operation as part of the account information.

POST_ADD_ACCOUNT

The account information that is being passed to the create account request is also passed to this program. Unlike the Modify operation, the password is passed to the Create operation as part of the account information.

PRE_MODIFY_ACCOUNT

The account information that is being passed to the modify account request is also passed to this program. The only exclusion to this is the password attribute.

POST_MODIFY_ACCOUNT

The account information that is being passed to the modify account request is also passed to this program. The only exclusion to this is the password attribute.

PRE_CHANGE_ACCOUNT_PASSWORD

A special case of MODIFY. This exit is triggered when the password attribute for the account changes. If the password attribute is the only change, the other modify code is not triggered. If other attributes change, this code is triggered. The account name and the password attributes contain the only information available to this program exit.

POST_CHANGE_ACCOUNT_PASSWORD

A special case of MODIFY. This exit is triggered when the password attribute for the account changes. If the password attribute is the only change, the other modify code is not triggered. If other attributes change, this code is triggered. The account name and the password attribute contain the only information available to this program exit.

PRE_ENABLE_ACCOUNT

A special case of MODIFY. This exit is triggered when the enable attribute for the account changes. If the enable attribute is the only change, the other modify code is not triggered. If other attributes change, this is triggered. The account name is the only account attribute available to this exit.

POST_ENABLE_ACCOUNT

A special case of MODIFY. This exit is triggered when the enable attribute for the account changes. If the enable attribute is the only change, the other modify code is not triggered. If other attributes change, this is triggered. The account name is the only account attribute available to this exit.

PRE_DISABLE_ACCOUNT

A special case of MODIFY. This exit is triggered when the disable attribute for the account changes. If the disable attribute is the only change, the other modify code is not triggered. If other attributes change, this is triggered. The account name is the only account attribute available to this exit.

POST_DISABLE_ACCOUNT

A special case of MODIFY. This exit is triggered when the disable attribute for the account changes. If the disable attribute is the only change, the other modify code is not triggered. If other attributes change, this is triggered. The account name is the only account attribute available to this exit.

PRE_DELETE_ACCOUNT

Triggered prior to a DELETE request. The account name is the only account attribute available to this exit.

POST_DELETE_ACCOUNT

Triggered after a DELETE request. The account name is the only account attribute available to this exit.

PRE_ADD_GLOBAL_USER

The global user information that is being passed to the create request is also passed to this program.

POST_ADD_GLOBAL_USER

The global user information that is being passed to the create request is also passed to this program.

PRE_MODIFY_GLOBAL_USER

The global user information that is being passed to the modify request is also passed to this program. The only exclusion to this is the password attribute.

POST_MODIFY_GLOBAL_USER

The global user information that is being passed to the modify request is also passed to this program. The only exclusion to this is the password attribute.

PRE_CHANGE_GLOBAL_USER_PWD

A special case of MODIFY. This exit is triggered when the password attribute for the global user changes. If the password attribute is the only change, the other modify code is not triggered. If other attributes change, this code is triggered. The global user name, the password, and optionally the password clue attributes are the only information available to this program exit.

POST_CHANGE_GLOBAL_USER_PWD

A special case of MODIFY. This exit is triggered when the password attribute for the global user changes. If the password attribute is the only change, the other modify code is not triggered. If other attributes change, this code is triggered. The global user name, the password, and optionally the password clue attributes are the only information available to this program exit.

PRE_ENABLE_GLOBAL_USER

A special case of MODIFY. This exit is triggered when the enable attribute for the global user changes. If the enable attribute is the only change, the other modify code is not triggered. If other attributes change, this code is triggered. The global user name is the only attribute available to this exit.

POST_ENABLE_GLOBAL_USER

A special case of MODIFY. This exit is triggered when the enable attribute for the global user changes. If the enable attribute is the only change, the other modify code is not triggered. If other attributes change, this code is triggered. The global user name is the only attribute available to this exit.

PRE_DISABLE_GLOBAL_USER

A special case of MODIFY. This exit is triggered when the disable attribute for the global user changes. If the disable attribute is the only change, the other modify code is not triggered. If other attributes change, this code is triggered. The global user name is the only attribute available to this exit.

POST_DISABLE_GLOBAL_USER

A special case of MODIFY. This exit is triggered when the disable attribute for the global user changes. If the disable attribute is the only change, the other modify code is not triggered. If other attributes change, this code is triggered. The global user name is the only attribute available to this exit.

PRE_DELETE_GLOBAL_USER

Triggered prior to a DELETE request. The global user name is the only attribute available to this exit.

POST_DELETE_GLOBAL_USER

Triggered after a DELETE request. The global user name is the only attribute available to this exit.

PRE_ASSOCIATE_ROLE

Refers to the changing of provisioning role membership in the Provisioning Server, regardless of what happens at the account level. The global user name and the provisioning role name are the only information available to this program exit.

POST_ASSOCIATE_ROLE

Refers to the changing of provisioning role membership in the Provisioning Server, regardless of what happens at the account level. The global user name and the provisioning role name is the only information available to this program exit.

PRE_DISASSOCIATE_ROLE

Refers to the changing of provisioning role membership in the Provisioning Server, regardless of what happens at the account level. The global user name and the provisioning role name are the only information available to this program exit.

Note: This value is called only for incremental provisioning role changes. If you use a replace-mode modification of global user's provisioning roles to replace one set of provisioning roles with another, this value calls the associate-role exits only. The exit would not read the database to find out which provisioning roles were previously included to see which were being set that were previously set and which were being removed.

POST_DISASSOCIATE_ROLE

Refers to the changing of provisioning role membership in the Provisioning Server, regardless of what happens at the account level. The global user name and the provisioning role name are the only information that is available to this program exit.

Note: This value is called only for incremental provisioning role changes. If one uses a replace-mode modification of global user's provisioning roles to replace one set of provisioning roles with another, this value calls the associate-role exits only. The exit would not read the database to find out which provisioning roles were previously included to see which were being set that were previously set and which were being removed.

PRE_ADD_GLOBAL_GROUP

The global group information that is being passed to the add request is also passed to this program.

POST_ADD_GLOBAL_GROUP

The global group information that is being passed to the add request is also passed to this program.

PRE_MODIFY_GLOBAL_GROUP

The global group information that is being passed to the modify request is also passed to this program.

POST_MODIFY_GLOBAL_GROUP

The global group information that is being passed to the modify request is also passed to this program.

PRE_DELETE_GLOBAL_GROUP

Triggered prior to a delete request. The global group name is the only attribute available to this exit.

POST_DELETE_GLOBAL_GROUP

Triggered after a delete request. The global group name is the only attribute available to this exit.

PRE_ADD_ROLE

The provisioning role information that is being passed to the add request is also passed to this program.

POST_ADD_ROLE

The provisioning role information that is being passed to the add request is also passed to this program.

PRE_MODIFY_ROLE

The provisioning role information that is being passed to the modify request is also passed to this program.

POST_MODIFY_ROLE

The provisioning role information that is being passed to the modify request is also passed to this program.

PRE_DELETE_ROLE

Triggered prior to a delete request. The provisioning role name is the only attribute available to this exit.

POST_DELETE_ROLE

Triggered after a delete request. The provisioning role name is the only attribute available to this exit.

CUSTOM_FUNCTION

Triggered when a program exit is invoked though a policy rule expression such as %$funcname(%UN%,%AC%)%.

Containment

Containment refers to the allowed combination of objects and program reference type, and not to X.500 containment.

Common Configuration Object

The Common Configuration Object is used to assign program exits for the global user, global user group, or provisioning role object classes. For example, if certain program exits should be called when global users are processed, the common configuration object should reference those exits. In addition, the common configuration object is needed to provide a way to call exits during the add operation.

- PRE_ADD_GLOBAL_USER
- POST_ADD_GLOBAL_USER
- PRE_MODIFY_GLOBAL_USER
- POST_MODIFY_GLOBAL_USER
- PRE_CHANGE_GLOBAL_USER_PWD
- POST_CHANGE_GLOBAL_USER_PWD
- PRE_ENABLE_GLOBAL_USER
- POST_ENABLE_GLOBAL_USER
- PRE_DISABLE_GLOBAL_USER
- POST_DISABLE_GLOBAL_USER
- PRE_DELETE_GLOBAL_USER
- POST_DELETE_GLOBAL_USER
- PRE_ADD_GLOBAL_GROUP
- POST_ADD_GLOBAL_GROUP
- PRE_MODIFY_GLOBAL_GROUP
- POST_MODIFY_GLOBAL_GROUP
- PRE_DELETE_GLOBAL_GROUP
- POST_DELETE_GLOBAL_GROUP
- PRE_ADD_ROLE
- POST_ADD_ROLE
- PRE_MODIFY_ROLE
- POST_MODIFY_ROLE
- PRE_DELETE_ROLE
- POST_DELETE_ROLE

Provisioning Roles

A role object can reference the program exits to assign the exits that are invoked for various operations on global users associated with that provisioning role. If a provisioning role references a program exit, those exits are called in addition to the exits referenced by the Common Configuration Object. The exits defined on the common configuration object are invoked before exits defined on the provisioning role (hierarchy order).

When adding a global user (PRE_ADD_GLOBAL_USER and POST_ADD_GLOBAL_USER exit types), program exits are invoked based on the initial set of provisioning roles being assigned to the user.

When associating or disassociating a provisioning role with a global user (PRE_ASSOCIATE_ROLE, POST_ASSOCIATE_ROLE, PRE_DISASSOCIATE_ROLE and POST_DISASSOCIATE_ROLE exit types), the program exits referenced by the provisioning roles being associated or disassociated are invoked.

When modifying an existing global user in other ways (using the exit types listed below), all provisioning roles to which the global user belongs are consulted to identify the program exits to invoke.

Other Exit Types

The common configuration object handles add exits for a global user.

- PRE_ADD_GLOBAL_USER
- POST_ADD_GLOBAL_USER
- PRE_ASSOCIATE_ROLE
- POST_ASSOCIATE_ROLE
- PRE_DISASSOCIATE_ROLE
- POST_DISASSOCIATE_ROLE
- PRE_MODIFY_GLOBAL_USER
- POST_MODIFY_GLOBAL_USER
- PRE_CHANGE_GLOBAL_USER_PWD
- POST_CHANGE_GLOBAL_USER_PWD
- PRE_ENABLE_GLOBAL_USER
- POST_ENABLE_GLOBAL_USER

- PRE_DISABLE_GLOBAL_USER
- POST_DISABLE_GLOBAL_USER
- PRE_DELETE_GLOBAL_USER
- POST_DELETE_GLOBAL_USER

Account Templates

An account template object can reference program exits to affect the accounts associated with that template.

If an account template references program exits, these exits are called in addition to the exits that are referenced by the endpoint to which the account belongs. The exits defined on the account template are invoked before the exits on the endpoint (hierarchy order).

If an account is being created from one or more account templates (PRE_ADD_ACCOUNT and POST_ADD_ACCOUNT exit types), those template exits are called.

When working with an existing account, whether the current set of assigned account templates is being adjusted or not, it is the initial set of assigned templates whose program exits are invoked.

- PRE_ADD_ACCOUNT
- POST_ADD_ACCOUNT
- PRE_MODIFY_ACCOUNT
- POST_MODIFY_ACCOUNT
- PRE_CHANGE_ACCOUNT_PASSWORD
- POST_CHANGE_ACCOUNT_PASSWORD
- PRE_ENABLE_ACCOUNT
- POST_ENABLE_ACCOUNT
- PRE_DISABLE_ACCOUNT
- POST_DISABLE_ACCOUNT
- PRE_DELETE_ACCOUNT
- POST_DELETE_ACCOUNT

Endpoints

Endpoint objects are used to assign program exits to accounts. For example, if certain program exits should be called when accounts are processed, the endpoint objects should reference those exits.

In addition, endpoint objects are needed to provide a way to call exits during an add operation.

- PRE_ADD_ACCOUNT
- POST_ADD_ACCOUNT
- PRE_MODIFY_ACCOUNT
- POST_MODIFY_ACCOUNT
- PRE_CHANGE_ACCOUNT_PASSWORD
- POST_CHANGE_ACCOUNT_PASSWORD
- PRE_ENABLE_ACCOUNT
- POST_ENABLE_ACCOUNT
- PRE_DISABLE_ACCOUNT
- POST_DISABLE_ACCOUNT
- PRE_DELETE_ACCOUNT
- POST_DELETE_ACCOUNT

Custom Function Program Exits

A custom function program exit is invoked from an account template rule expression. Custom function program exits share the following characteristics:

- The exit type is always CUSTOM_FUNCTION. There are no PRE or POST variants.
- The exit must be a common program exit (DLL or SOAP). Native exits cannot be used to compute the custom function.
- The exit must be registered in the same domain as the account being created or updated from the policy. The reference to a custom function program exit (%$funcname(...)%) contains the name of the exit (funcname), but there is no rule string syntax to let you specify the domain of the program exit so it is always presumed to be in the domain of the account.
- The input to the program exit includes zero or more single- or multi-valued parameters.

For example, if the global user for the account being created or updated has the following attribute settings:

```
eTCustomField01:  { value1a, value1b }      (that is, two values assigned)
eTCustomField02:  value2
```

The %*$FuncName(%*UCU01%, %UCU02%)% rule expression is evaluated.

The input XML passes all values of eTCustomField01 and the value of eTCustomField02 as follows:

```
<eTExitInvoke eTExitType=CUSTOM_FUNCTION>
   <eTFunction>
      <eTFuncParam1>value1a</eTFuncParam1>
      <eTFuncParam1>value1b</eTFuncParam1>
      <eTFuncParam2>value2</eTFuncParam2>
   </eTFunction>
   <Authentication>
      <Type>GLOBAL_USER</Type>
      <User>{the DN of the global user}</User>
      <Password>{the password of the global user}</Password}
   </Authentication>
</eTExitInvoke>
```

The output from the program exit can indicate an error (as with any other program exit) so that the creation or update of the account is not attempted, or can contain a single- or multi-valued output parameter. For example, a program exit could return the following XML block to indicate two values (ReturnValue1 and ReturnValue2) to set for the corresponding account attribute:

```
<eTExitReturn>
   <eTExitReturnCategory>SUCCESS</eTExitReturnCategory>
   <eTExitReturnNative>0</eTExitReturnNative>
   <eTExitContinue>TRUE</eTExitContinue>
   <eTExitCustom>
      <eTFuncReturn>ReturnValue1</eTFuncReturn>
      <eTFuncReturn>ReturnValue2</eTFuncReturn>
   </eTExitCustom>
</eTExitReturn>
```

The function rule expression controls the number of values to set as follows:

- If the $FuncName in the rule expression is preceded by * (asterisk) as in the example above, this will set 0 or more values of the attribute depending on what is included in the output XML document.
- If the function rule expression does not have the * preceding $FuncName, only the first value returned is relevant. Additional values are ignored.

Obscured Returned Values

The program exit returns information inside the eTFuncReturn XML block, for example:

```
<eTFuncReturn>Returned value from program exit</eTFuncReturn>
```

If logging is enabled, then this XML block can be read.

However, if the program exit returns information like a password, then you may not want the information to be logged. In this case, you can flag the returned value as obscured to prevent it from being logged.

The format of the obscured value is:

```
<eTFuncReturn obscured="yes">MyPassword</eTFuncReturn>
```

This tells the Provisioning Server to replace the value with the string ** NOT SHOWN ** as it does for attribute names that are recognized as storing sensitive attributes. For example:

```
<eTFuncReturn obscured="yes">** NOT SHOWN **</eTFuncReturn>
```

Note: The obscured attribute is case-sensitive. For example, the result will not be replaced correctly if the attribute is set to "YES".

Code Examples

Use the code examples located in the following directory as a guide when coding your Common Program Exits:

```
Samples\ProgramExits
Samples\ProgramExitSOAP
```

Chapter 6: Program Exits In Connectors

This chapter covers information about supporting program exits you have created in the Provisioning Server SDK for the connectors you have created.

This section contains the following topics:

Execution Flow (Logic)

Program exits are referred to as either pre-exits or post-exits, depending on the operation that the Provisioning Server performs.

Pre-Exits

The Provisioning Server framework invokes a pre-exit before executing a particular operation. For common exits, the Provisioning Server framework interprets the return XML buffer to determine whether to continue execution of the particular operation. For native exits, the agent plug-in must interpret the return XML buffer.

If the agent plug-in returns a success status code (LDAP_SUCCESS), the Provisioning Server continues to perform the operation. If the agent plug-in returns an error code, the operation is aborted.

Note: For native program exits, your custom connector agent plug-in must interpret the return XML buffer. Furthermore, your agent plug-in must return either success or failure to the Provisioning Server framework. Success lets the operation continue. Failure aborts the operation.

If one pre-exit fails, the operation will be aborted even if other pre-exits let the operation continue. In addition, as soon as a pre-exit aborts the operation, other pre-exits (with the same or lower priority) will not be called.

Priority of Pre-Exits

Pre-exits are called in order of the hierarchy and priority. If two program exits are referenced with the same priority, the order in which they are called is undefined. Priority guarantees only that higher priority exits are called before lower priority exits. The highest priority is “1” and the larger the number, the lower its priority.

More information:

Program Exit Hierarchy and Order (see page **66**)

Operation

Once the Provisioning Server framework has invoked all pre-exits associated with the operation and each pre-exit lets the operation continue, the Provisioning Server framework executes the operation. Execution of the operation may result in another request to your agent plug-in.

Note: The agent plug-in receives multiple requests: one for the pre-exit, one for the operation, and one for the post-exit.

Post-Exit

Once the operation is completed successfully, the Provisioning Server framework invokes the post-exits referenced for that operation. Post-exits should be handled the same way as pre-exits. Your agent plug-in should not differentiate between a pre-exit and a post-exit, and you can use the exact same code to handle pre- and post-exits.

Priority of Post-Exits

Post-exits are called in order of hierarchy and priority. If two program exits are referenced with the same priority, the order in which the exits are called is not guaranteed. Priority only guarantees that higher priority exits are called before lower priority exits. The highest priority is “1” and the larger the number, the lower its priority.

More information:

Program Exit Hierarchy and Order (see page **66**)

Support for Common Exits

Common exits are processed by the Provisioning Server framework. Thus, supporting common exits requires minimal changes. You only need to enhance your GUI plug-in. No agent plug-in change is needed.

Parser Table Enhancement

The parser table files already define new attributes for exit reference. Option parser table files include these parser table files, so you do not need to make any changes to your parser table.

GUI Plug-In Enhancement

Provisioning Manager provides a standard property page for referencing program exits. To support common exits, you only need to add this property page to your account and endpoint property sheets. This enables your account and endpoint objects to reference program exits.

The standard exit reference property page is managed by the CosExitRefPage class in the COS module. For example, to add the property page to your account and directory property sheets, add the following line to your property sheet code:

```
propertyPages->AddTail(new CosExitRefPage(this));
```

Agent Plug-In Enhancement

No agent plug-in change is needed to support common exits.

Support for Native Exits

Native exits are processed by the endpoint. Thus, to support native exits, the endpoint must enhance both its GUI plug-in and agent plug-in.

Parser Table Enhancement

The parser table files already define new attributes for exit reference. Parser table files include these parser table files, so you do not need to make any changes to your parser table.

Since you are providing native program exits, you need to define your custom program exit object in the parser table. The exit object must have the following CLASS definition line:

```
CLASS Exit.<exit class name>,eTExit.<exit class name>,etavlcor,secobjar
```

For example, the SDK defines the exit object class as follows:

```
CLASS Exit.SDKExit,eTExit.eTSDKExit,etavlcor,secobjar
```

For a complete example of how to define an exit object, see the SDK sdkparse.pty sample file, which is provided with the SDK.

GUI Plug-In Enhancement

Exit Reference

The Provisioning Server GUI framework provides a standard property page for referencing program exits. To support common exits, you need to add the property page to your account and directory property sheets. This lets your account and endpoint objects reference program exits.

The standard exit reference property page is managed by the CosExitRefPage class in the COS module.

For example, to add CosExitRefPage to your account and directory property sheets, add the following line to your property sheet code:

```
propertyPages->AddTail(new CosExitRefPage(this));
```

Exit Definition

A property sheet must be provided to define a native program exit. This endpoint-specific exit definition property sheet is used to enter specific information that will be passed to the agent plug-in during exit invocation.

Your endpoint must define an XML format for this data, which is stored as an XML buffer in the eTExitPayload attribute in the custom program exit object. When an exit is invoked, the Provisioning Server framework sends this data to the agent plug-in. The agent plug-in parses the eTExitPayload attribute to get the data it needs to invoke the program exit.

Agent Plug-in Enhancement

Program Exit Invocation Request

The Provisioning Server framework sends all native program exit invocation requests to the endpoint. Even if the exit is referenced by an account object, the invocation request is sent to the code that manages the endpoint modify operation. Specifically, the directory DEmodify() function is called.

On a program exit invocation request, the Provisioning Server framework includes the eTExitPayload and eTExitInvoke attributes.

eTExitPayload contains the data regarding the definition of the exit. The value of eTExitPayload is the XML buffer stored in the program exit object that was defined by the program exit definition property sheet that you added to your GUI plug-in.

eTExitInvoke is an XML buffer. This data should be passed to the program exit, which needs to process it. The agent plug-in can process this information; however, often it does not need to do so.

The agent plug-in must be enhanced for performing the following tasks to support the program exit:

1. Determine whether an operation is an exit invocation request.
2. Invoke the program exit.
3. Interpret the result from the program exit.

More information:

Program Exit Input Argument

Determine Exit Invocation Request

Typically, the directory DEmodify() function processes requests to change values in the directory object. To support the native program exits, you must enhance the directory DEmodify() function to also handle native exit invocation.

For a program exit invocation request, the Provisioning Server framework sends the eTExitInvoke attribute as part of the modify operation. The presence of the eTExitInvoke attribute is an indication that the request is an exit invocation request and not a normal modify request, as shown in the following example:

```
/*
|| The special attribute UTFEXITINVOKE indicates a request to invoke a
|| program exit.
 */
if (pMods->find_mod(UTFEXITINVOKE)) {

// Invoke the exit.

}
else {

// Normal directory modify request.

}
```

Invoke the Program Exits

You must define how your custom connector agent plug-in invokes the program exit.

The common exit has the following invocation methods:

- Through the DLL function call
- Through the SOAP method invocation

Your agent plug-in will probably define some other form of program exit invocation. That data is passed to the agent plug-in in the eTExitPayload attribute, which is an XML buffer.

The method of invoking program exits is to execute a command line utility. Thus the only information it needs is the utility name (including the path). You can define the SDK exit object as having a payload that only contains the full path to the utility.

The sample SDK exit payload is the following:

```
<eTSDKExit>
     <Program>program to execute</Program>
</eTSDKExit>
```

Interpret the Result from the Program Exit

Each program exit must return an XML buffer.

The agent plug-in must interpret this return XML buffer and return an appropriate status code to the Provisioning Server framework. For pre-exits, returning a success status to the Provisioning Server framework lets the operation continue. Returning a failure status will abort the operation.

Note: If the operation has multiple pre-exits, one pre-exit might return a success, which would let the operation continue; but another pre-exit could return failure, thus aborting the operation. If one pre-exit aborts the operation, it is aborted, even if other pre-exits let the operation continue. In addition, as soon as a pre-exit aborts the operation, other pre-exits with the same or lower priority will not be called.

More information:

Program Exit Return Value (see page **70**)

Exit Types

The Provisioning Server framework only sends an exit invocation request to the agent plug-in if the exit type is one that can be handled by the agent plug-in. In general, your agent plug-in does not need to handle exit types. However, if your custom connector only permits certain program exit types, it must check the eTExitType tag attribute in the eTExitInvoke attribute.

Exit Type Functionality

Exit type is a value that determines the circumstances under which an exit is called. One of the types of exits is entered for eTExitType (in the input XML buffer passed to the program exit). The first 12 exit types (values) can be common or native exits, that is, common code and namespace (connector) code can be triggered to process them. The remaining exit types, however, can be common exits only. It should be noted that not all program exit types are referenced from various object types.

Notes: In all cases the name of the object being passed is sent. This is formatted in both DN and Common Name format.

To have complete control over passwords (either at the global user or the account levels), you must provide exits both for create user/account and for change password user/account. In other words, for new global users (accounts), the change password exit is not called. For new global users (accounts), the password is passed in as part of the attribute for the create exit (for example, PRE_CREATE_GLOBAL_USER).

More information:

Valid Values for Exit Types (see page **75**)

Code Examples for Program Exits in Options

See the SDK sample connector. The exit handling code is in the SDKDirectory.cpp file.

The method SDKDirectory::DEmodify() determines whether a request is an exit invocation request,and if it is, calls the SDKDirectory::InvokeExit() method.

Chapter 7: Provisioning Maintenance

This section contains the following topics:

Back Up and Restore CA Directory

To ensure that data is coherent across your entire organization, regular backups should be done. Regular backups of CA Directory prevent data loss and damage caused by network disasters and failures. CA Directory provides an online backup utility (and the dxdumpdb and dxloaddb utilities for offline backup) to back up and restore CA Directory.

Note: For information about these utilities, see the CA *Directory Administrator Guide*.

Shut Down the Provisioning Server service

If the Provisioning Server service does not shut down, you can manually shut it down as follows:

1. Open a command prompt and enter the following command:

   ```
   net stop im_ps
   ```

2. If Services indicates that the Provisioning Server service is still in the stopping state, issue the following commands:

   ```
   net start im_ps
   net stop im_ps
   ```

   A similar procedure can be used to manually shut down the Provisioning Connector Server service, whose service name is im-ccs.

If the service still does not stop, open the Task Manager, select im_ps.exe (or im_ccs.exe) on the Processes tab, and click End Process.

View and Maintain Log Files

The provisioning components (Provisioning Server, Connector Servers, Provisioning Manager) can be configured to log information about all transactions that they process. You can use this information to predict and identify the sources of system or security problems. For example, if the warning messages in log files show that some accounts on an endpoint could not be explored, you can use the logged information to investigate those accounts and determine why they were not explored. Use a text editor to view and edit provisioning log files.

Server Event logs track messages generated by the Provisioning Server. You can log messages to several optional destinations, including CA Audit.

The provisioning components provide other types of logging to diagnose specific problems. Other than the provisioning server trace log, these logs are usually not enabled unless you need them to trace a particular event. They include provisioning server logs, slapd logs, and C++ Connector Server logs. You can also diagnose problems that occur when communicating with the provisioning server by enabling Provisioning Manager logging.

Messages from all logs are written to text files in the *PSHOME*\Logs directory and are named accordingly:

- Provisioning Server Event Log — eta*yyyymmdd*.log
- Provisioning Server Trace Log — etatrans*yyyymmdd-hhmm*.log
- Provisioning Server IMS Notification Log — etanotify*yyyymmdd-hhmm*.log
- Provisioning Server SLAPD Log — im_ps.log
- Provisioning Manager Log — etaclient*yyyymmdd*.log
- C++ Connector Server Endpoint Log — sa*yyyymmdd*.log
- C++ Connector Server Trace Log — satrans*yyyymmdd-hhmm*.log
- C++ Connector Server SLAPD Log — im_ccs.log

Server Event Logging

Server Event logs record important events generated from the Provisioning Server. These events consist of all *severity levels* (success, information, warning, fatal, and error). The logs record every client-initiated operation and its success or failure, including generated sub-operations.

In the System Task frame of the Provisioning Manager, under Global Properties, use the Logging tab to configure Server Event logging. Server Event logs typically only need to be configured once.

In some cases, you can turn logging on or off, or you can configure the severity levels of the messages logged. Thus, this Server Event logging can serve to audit the activities that are taking place within the Provisioning Server. However, the preferred auditing of provisioning activity is to enable the IMS Notifications features. The IMS Notifications feature sends detailed audit records to the IMS server for inclusion in the full audit record of CA Identity Manager activity. The notification records sent to the IMS can also trigger events for the additional CA Identity Manager Server processing.

Endpoint Logging

In the Endpoint Task frame, you can configure endpoint-specific logging. Endpoint logs track messages that a connector generates when it processes requests for objects residing in that endpoint. Each endpoint can be configured separately so you can turn logging on or off for just the endpoints where you need to learn additional information to diagnose problems.

You can also specify the severity (success, information, warning, fatal, and error) of the messages that get logged.

To turn logging on or off and to set the logging destinations and the severity levels of the messages logged for each endpoint, use the Logging tab of the endpoint's property sheet in the Provisioning Manager. For detailed instructions, see Setting Endpoint Logging in the Provisioning Manager help.

Endpoint logging is sent to a log file for the connector server in which the connector for the endpoint runs. For C++ connectors, the default log file name is *PSHOME*\Logs\saYYYYMMDD.log. The C++ connector server also adds some additional messages to this log. You control the log file name in the im_ccs.conf using the BaseLogFileName parameter. And you control which severities of these other messages are logged in the same conf file using the LogSeverities parameter.

Endpoint logging from connectors which run directly within the provisioning server (for example, the CA ACF2 connector) log to the provisioning server's event log which has the default name of *PSHOME*\Logs\etaYYYYMMDD.log.

Diagnostic Logging

To diagnose specific problems, you can enable the provisioning server trace log, slapd logs, or C++ Connector Server logs. These are typically not enabled unless you need them to trace a specific type of event. Provisioning Manager logging also is used for diagnosing problems in the Provisioning Manager or client utilities.

Provisioning Server Trace Log

Enable this logging component to generate a special transaction log file that records the details of every transaction processed by the Provisioning Server. You can choose from several logging levels to match the level of logging detail you prefer using the domain configuration parameter Transaction Log/Level.

The Provisioning Server trace log writes messages to *PSHOME*\Logs\etaTrans*yyyymmdd-hhmm*.log. To change the base part of the file name (the part before the date) or to relocate this log file to another drive, modify the domain configuration parameter Transaction Log/File name. For more information about the etaTrans*yyyymmdd*.log file, see the Provisioning Manager help.

Note: Unlike most logging which is turned off by default, Provisioning Server logging is fully enabled as the component is installed. If you choose not to run with the maximum trace logging of the provisioning server, you need to change the domain configuration parameters that control this logging. These parameter are located in the "Transaction Log" parameter folder in the Provisioning Manager on the System task under Domain Configuration.

Provisioning Server IMS Notification Log

The Provisioning Server is typically configured to send notifications (global user and other object change records) to the CA Identity Manager Server for integration with the IMS event system and audit data base. A notification thread running within the Provisioning Server reads notification records from the local notify DB and transmits them to the IMS. This activity is captured in the IMS Notification log, whose name is *PSHOME*\Logs\etanotify*yyyymmdd-hhmm*.log.

You configure the severity of log messages included in this log on the CA Identity Manager Setup screen in the System Task of Provisioning Manager.

The format of this log is similar to the Provisioning Server and Connector Server trace logs.

SLAPD and C++ Connector Server Logs

On Windows, you can enable SLAPD logging for advanced debugging tasks such as LDAP protocol packet handling and search-filter processing. You can set the log level in the Windows registry by assigning a value to the DebugLevel key. There are two registry keys, each controlling the logging for one of the services:

HKEY_LOCAL_MACHINE\SOFTWARE\ComputerAssociates\slapd\im_ps\CurrentVersion\DebugLevel

The im_ps registry key controls logging for im_ps.exe, run by the Provisioning Server service.

HKEY_LOCAL_MACHINE\SOFTWARE\ComputerAssociates\slapd\im_ccs\CurrentVersion\DebugLevel

The im_ccs registry key controls logging for im_ccs.exe, run by the Connector Server service.

Important! The preferred method for enabling SLAPD logging is by setting the loglevel parameter in im_ps.conf or im_ccs.conf, for both Windows and Solaris. Each file contains configuration instructions.

The DebugLevel registry key or loglevel configuration file parameter specifies the amount of information the server writes to its log file, which is one of the following, depending upon your type of slapd service:

- *PSHOME*\Logs\im_ps.log
- *PSHOME*\Logs\im_ccs.log

Note: A "TLS: can't accept" error message may appear in the im_ps.log file when running in FIPS mode due to a low-level initialization problem that clears up after the first connection from a client. Since clients retry connections, you can ignore this message.

You can select a debug level to match the type of debugging you want to perform. The debug levels are listed in the following table:

Value	Debug Information
1	Trace function calls
2	Debug packet handling
4	Heavy trace debugging
8	Connection management
16	Print out packets sent and received
32	Search filter processing

Value	Debug Information
64	Configuration file processing
128	Access control list processing
256	Stats log connections/operations/results
512	Stats log entries sent
1024	Print communication with shell back-ends
2048	Entry parsing
65535	All tracing

C++ Connector Server Trace Logging

C++ Connector Server Trace Logs record the activity of the C++ Connector Server, which is a module used to help manage many endpoint types. This log performs the following functions:

- Logs trace and debug messages for the C++ Connector Server.
- Monitors all statuses returned by its connectors. For example, if a connector returns fatal LDAP errors, the C++ Connector Server logs these errors with severity LOG_FATAL.

To set the log file name and logging levels in im_ccs.conf set the SATransLog and SATransLogLevel parameters. The supported logging levels are 0 (for off) and 1 (for on). The default is 0. These parameters must exist in the file after the database superagent line.

Provisioning Manager Logging

To diagnose problems communicating with the server, you can set logging to record events that transpire between the Provisioning Manager and the Provisioning Server to which it is connected. Use the Logging tab under File, Preferences to trace all requests sent to any server from the Provisioning Manager.

This logging is actually logging within the C/C++ client library used by the Provisioning Manager and some other clients (batch utility, password manager, csfconfig, bindeta, pingeta). Once logging is enabled and configured using Provisioning Manager, those log settings apply for these other clients as well. Each client logs its command name as it logs messages so you can identify which log messages are specific to which client.

However, for this to work the client being run must reside in the same file system folder as the Provisioning Manager's etadmin.exe program. When this is not the case (such as when running on Solaris where there is no Provisioning Manager install, or even on Windows when you run utilities from the Provisioning Server's installation), the client library consults registry settings specific to the Provisioning Server instead of specific to the Provisioning Manager. Set these other registry settings by running these eta-env commands using the eta-env program included in the Provisioning Server installation:

```
eta-env
    action=set
    name=Manager/LogMaster
    type=int
    value=1

eta-env
    action=set
    name=Manager/LogDestinations
    type=int
    value=16

eta-env
    action=set
    name=Manager/LogSevFile
    type=int
    value=31
```

These have the effect of configuring the C/C++ client library for the provisioning server's installation, setting the destination to "text file" and logging all message severities.

Finally, the csfconfig command has a "debug=yes" command-line parameter you can specify to turn this logging on for one command invocation overriding any registry settings configured with Provisioning Manager or eta-env.

Use AnalyzeLog

The command line utility, AnalyzeLog, takes as input a Provisioning Server trace log (etatrans*yyyymmdd-hhmm.*log) and produces different views of the information depending on what options you set. You can use this information to diagnose functional or performance problems reported by users.

Note: For more details on this utility, see the Provisioning Manager online help.

Log Files for High Availability

To ensure proper operation of your high-availability configuration, you should monitor the following log files:

- Alarm
- Warn
- Stats
- Diag
- Summary
- Trace

All logs can be flushed through the DXserver console. Only the SUMMARY and TRACE logs can be closed from the console.

Provisioning Directory Monitoring

Monitor Thresholds

Monitoring is one of the most significant activities as part of the system availability and security. Monitoring is needed to determine the source of performance problems, fault detection, and to take corrective action.

Important! Regular monitoring of the multiwrite queue is highly recommended. For more information on using multiwrite, see the information on replication in *CA Directory Administrator Guide.*

CA Directory can provide SNMP counters to an SNMP-aware Enterprise Management Application. CA Directory can also provide SNMP traps under the following conditions:

- Authentication Failure
- Alarms
- Directory Updates

You can also use the CA Directory Statistics logs to gauge application load over time. This can be invaluable in providing data to both measure and show the growth of the new service.

Observe Router Traffic

CA Directory includes the DXconsole utilility, which you use to monitor the network traffic going through a CA Directory router.

The router knowledge files, such as imps_router.dxc, define the console port for a router. You can adjust the console port number to satisfy your enterprise needs. To monitor traffic sent through a router, start up DXconsole and connect it to the localhost with the port number defined in the router knowledge file.

Note: Because the Provisioning Server uses LDAP for communication between various components, you can set the trace level to ldap. For more information, see the *CA Directory Administrator Guide.*

Enable SSL Encryption

CA Directory includes a dxcertgen.exe utility, which you can use to generate certificates and personality files. Using this utility is the recommended method for enabling encryption for high availability solutions.

For more information, see the *CA Directory Administrator Guide.*

Downsize a DSA

If the DSA becomes too large, you can downsize it.

Follow these steps:

1. Stop the DSA.

   dxserver stop *dsa_name*

2. Dump the datastore to LDIF:

   dxdumpdb -f *filename dsaName*

   For example: dxdumpdb -f democorp.ldif democorp

3. Edit the following DSAs initialization file to change dxgrid-db-size to the required size:

   dxserver\config\servers*dsaname*.dxi

4. Reload the data:

   dxloaddb -s *dsaName* ldif-*filename*.ldif

   For example: dxloaddb -s democorp democorp.ldif

Chapter 8: Pluggable Authentication Module (PAM)

This section contains the following topics:

PAM Overview

CA Identity Manager provides a Pluggable Authentication Module (PAM) that allows the Provisioning Server to authenticate against external security systems, such as a Primary Domain Controller (PDC).

Note: An external security system need not be a Windows system. But if it is, you can use one of the predefined PAM modules provided with the Provisioning Server (two for Active Directory and one for Windows NT). Otherwise, you must write your own PAM module.

When PAM is enabled, global users can log on to any Provisioning Manager or etautil using the user's password in the external security system. It is only the password check that the Provisioning Server defers to the PAM module. All additional user information, such as suspension state, Self-Administration check box, and administrative privileges, that controls what actions you may perform still reside in the provisioning directory as properties of the global user or related objects.

How PAM Works

The PAM authenticates passwords using the following process:

1. Global users log in to a CA Identity Manager interface with their native system passwords.
2. CA Identity Manager sends the authentication information to the PAM loader.
3. The PAM loader sends the authentication information to PAM where it is authenticated against the system. The CA Identity Manager server grants or fails login requests based on the results returned from the PAM DLL.

Sample PAM Programs

CA Identity Manager provides the following sample PAM programs:

- Windows NT
- ADS
- ADSMultiDomain
- Generic PAM sample

Note: The Generic PAM sample is included in \Provisioning Server SDK\admin\samples\SDK\PAM.

Each sample program includes the etaPAM.dll binary, which lets you authenticate to a local Windows system. Sample source code is provided in the Provisioning SDK for reference purposes only. Customization may be needed for a production deployment.

Any global user who needs to authenticate to CA Identity Manager must exist in the Windows domain. This includes the special global user, etapwsad. This user must be created with the PasswordAdministrator profile. It does not exist by default.

If these users do not exist, the CA Identity Manager clients will not function.

The Generic PAM Sample shows an example implementation of the PAM interface that does not actually communicate with an authentication system. This sample demonstrates how to implement the PAM interface on Windows or Solaris, without being tied to a particular platform for carrying out the user authentication.

Use PAM

The PAM program is called etaPam.dll. The etaPam.dll is in these folders:

- PSHOME/PAM/ADS
- PSHOME/PAM/ADSMultiDomain
- PSHOME/PAM/NT

The ADS version authenticates to a Windows domain controller in a single-domain or multi-domain ADS environment. To ensure that ADS domains are trusted and can be reached, the Provisioning Server must be in the ADS domain. The NT version authenticates to the local accounts database. By default, these programs authenticate passwords on Windows systems only.

To use PAM with other systems, write another PAM program for the system. Sample source for the Windows or UNIX PAM modules provided with CA Identity Manager can be obtained by installing the Provisioning SDK. The generic PAM runs on UNIX and the NT and ADS samples run on Windows.

Using PAM and Password Expiration

If your environment is configured for PAM, you must inform PAM through the etapam_id.conf configuration file if the endpoint PAM is to consult has been acquired in CA Identity Manager as a managed endpoint. This allows CA Identity Manager to update a user's account password on this managed endpoint every time users or administrators update their global user passwords. For more information, see Update Global User Passwords.

If you configure PAM to consult an endpoint that has not been acquired in CA Identity Manager as a managed endpoint, disable the password expiration features. If you do not disable the password expiration feature, your users will be confused as they are asked to change their global user password although that password is not used for authentication.

To disable password expiration

1. Click the System Task button in the Provisioning Manager.
2. Click Password Profile.
3. Click the Expiration/Locking tab.
4. Clear the Expiration Period and click Apply.

Update Global User Passwords

You can configure PAM so that any update to a global user password also updates the account in the external security system that PAM uses for password verification. To do this, the external security system needs to be an acquired endpoint in CA Identity Manager, that is, a managed endpoint.

To configure automatic updates to external security systems, perform the following steps:

1. Add the following lines to your etapam_id.conf file:
   - For Active Directory:

     ```
     endpoint-type=ActiveDirectory
     endpoint-domain=YOUR_DOMAIN
     endpoint-name=YOUR_ENDPOINT_NAME
     ```

   - For Windows NT:

     ```
     endpoint-type=Windows NT
     endpoint-domain=YOUR_DOMAIN
     endpoint-name=YOUR_ENDPOINT_NAME
     ```

2. Replace *YOUR_DOMAIN* and *YOUR_ENDPOINT_NAME* with the domain and endpoint names that identify this managed endpoint in CA Identity Manager. You can omit endpoint-domain if the endpoint is in the server's local domain.
3. Start or restart the Provisioning Server service to have this change take effect.

When a managed endpoint is enabled in this way, any change to a global user password is also applied to the password of the matching account on the indicated endpoint if that account is correlated to the global user. The account password update occurs whether or not password propagation to accounts is requested. It occurs even in cases where password propagation would not have occurred. For example, the update occurs even if password propagation to the endpoint has been disabled or if the global user is marked as restricted. If password propagation to accounts is requested, the global user's other accounts are updated as well.

For Active Directory, a matching account is one whose samAccountName attribute is equal to the global user's name. For Windows NT, a matching account is one whose account name is the same as the global user's name.

Use PAM with Multiple ADS Domains

PAM must be configured on each Provisioning Server individually. Each PAM module only handles the authentication for the Provisioning Server where it is installed.

PAM allows a comma-delimited list of ADS domains to be added to the etapam_id.conf file.

To use PAM with multiple ADS domains

1. Ensure that the user account, specified as the user the Provisioning Server service logs on with, has the Act as Part of the Operating System privilege. If you need to add this privilege to the user, you must restart this service.
2. Ensure that enable=yes appears in the etapam_id.conf file.
3. Add a comma-delimited list of ADS domains to be added to the domain= setting in the etapam_id.conf file. These domains must be trusted by the domain in which Provisioning Server is installed, or be its own local domain. The following is an example:

   ```
   domain=LocalDomain,Trusted1,Trusted2
   ```

   CA Identity Manager attempts to authenticate to each listed domain until it is successful.

   Copy etapam_id.conf from the PSHOME\PAM\ADSMultiDomain directory to PSHOME\PAM.
4. Copy etapam.dll from the PSHOME\PAM\ADSMultiDomain directory to PSHOME\bin.
5. Restart the Provisioning Server service to make the Provisioning Server aware of any changes that were made to etapam_id.conf.
6. Log in with a user name that is both a CA Identity Manager Global User name and an ADS account name in one of the ADS domains specified in etapam_id.conf. Use the account's ADS password, not its Global User password. The Global User must have the Provisioning Server administrative privileges that are necessary for the user's purpose.

Troubleshooting PAM

The Provisioning Server Trace log (regardless of the configured log level), upon start up of the Provisioning Server service, will contain information about whether PAM is enabled or disabled and whether there is a valid configured PAM managed endpoint. To see this information, view the PSHOME\Logs\etatransYYYYMMDD.log and look for lines containing the PAM: string.

Common messages include the following:

PAM: Initialization started

Signals the start of PAM processing.

PAM: Not enabled

Indicates that PAM is not being used and could mean any of the following:

- The etapam_id.conf file was not found or invalid.
- The etapam_id.conf file was found but the enabled parameter was not set to yes.
- The etapam.dll file was not found on the execution search path or that the supplied etapam.dll file could not be successfully loaded.

PAM: No PAM managed endpoint

Indicates that no managed endpoint was specified using endpoint-type, endpoint-domain and endpoint-name parameters in etapam_id.conf.

PAM: Missing EPType or EPName

PAM: Unable to find specified domain

PAM: Unable to find specified endpoint

These messages indicate that there was a problem identifying the managed endpoint using the supplied endpoint-type, endpoint-domain, and endpoint-name parameters in etapam_id.conf.

PAM: Managed endpoint configured

Indicates that the managed endpoint identified by endpoint-type, endpoint-domain, and endpoint-name was valid. The next line logged is the full LDAP distinguished name of the managed endpoint.

Activate System PAM Debug Mode

The PAM library can provide debug information during execution. After enabling the system to collect debug output, you can use the gathered information to track PAM-API invocations and determine failure points in the current PAM setup. To enable PAM debug output, create an empty /etc/pam_debug file. The PAM library checks for existence of the /etc/pam_debug file and if found, enables syslog output.

Index

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W