ENOVIA DesignSync

Access Control Guide

3DEXPERIENCE 2022

Table Of Contents

Guide Organization

The Access Control guide provides the access controls that allow and deny access to commands and GUI interfaces.

ACAdmin

The DesignSync Web Interface features a GUI interface to create and manage access controls. This section of the guide documents the ACAdmin interface. For information on getting started with the ACAdmin interface, see Access Administrator Overview.

Access Controls

The documentation for Access Controls is organized by major topic area, for example, Mirrors, Data Replication, Modules, etc. and shows you how the Access Controls are defined and used, including examples and usage tips. Before you begin to customize the Access Controls, familiarize yourself with the Access Control Basics by starting with Introduction to Access Control.

Note: References from the *ENOVIA Synchronicity Access Control Guide* to the *ENOVIA Synchronicity Command Reference* guide always link to the ALL version of the guide, which contain information about all working methodologies for DesignSync. For more information about the available working methodologies, see ENOVIA Synchronicity Command Reference.

Access Control Basics

Introduction to Access Control

The access control system determines which users can access specific functionality in ENOVIA Synchronicity DesignSync Data Manager products. You can customize access controls to meet the needs of your organization.

How Access Controls Work

Access controls let you limit access to certain DesignSync, ProjectSync, and module operations, or **actions**:

- For DesignSync, you can control access to most DesignSync actions, such as checking in files, checking out files, removing locks, tagging files, retiring files, and deleting versions from a vault. In addition, you can specify the level of user authentication required when accessing DesignSync data.
- For modules, you can control access to most module actions, including creating and removing modules and hierarchical references, and upgrading DesignSync vaults or legacy modules to the current module format.
- For ProjectSync, you can use access controls to prevent users from performing certain operations or allow only specified users to perform operations. For example, you can use access controls to ensure that users can edit only their own user profiles.

  In most cases, operations that the user does not have access to perform are removed from the ProjectSync GUI. For instance, if you do not have permission to edit projects, the ProjectSync menu does not list that option.

  Options are removed from the GUI only when an operation is completely inaccessible to a user. For example, if the user can edit some projects, but not all of them, the menu option for modifying a project remains available. However, if a user attempts an operation for which permission is denied, a standard error panel displays.

How the Access Controls Are Implemented

DesignSync implements access control using `AccessControl` files, which contain stcl commands that define actions, as well as stcl commands that allow or deny these actions to particular users.

DesignSync also provides a web-based interface to control the Access permissions, users and groups who have access to design data and commands. The web-based interface, Access Administrator is described in Access Administrator Overview.

By default, all users have access to all ProjectSync and DesignSync actions except for deleting vaults (`rmvault` command), deleting versions (`rmversion` command), rolling back a module version (`rollback`), and changing lock owners (`switchlocker` command). All users have access to all module actions except for deleting modules (`rmmod` command). For these operations, access is denied to everyone, by default.

A primary `AccessControl` file is shipped with DesignSync: `$SYNC_DIR/share/AccessControl`. This file contains code that sources additional Access Controls Files:

- System provided TCL utility procedures and variable definitions for Access Control. This file is located `$SYNC_DIR/share/tcl/AC.tcl`. This file that should not be modified.
- System `AccessControl.*` files that define actions for each subset of the DesignSync products. These individual files are in contained in the following directory `$SYNC_DIR/share/AC_Components`. These files should not be modified.
- Custom TCL utility procedures and variable definitions for Access Control. These files are applicable to the server, site, or enterprise-wide. For information on creating and maintaining the custom Access Control TCL file, see Setting Up Access Controls.
- Custom `AccessControl.*` files that define rules for each subset of the DesignSync products. For information on creating and maintaining the custom Access Control files, see Setting Up Access Controls.

Access Control Definition Files

The access control definitions are divided into subsets for ease of use. DesignSync provides the following Access Control definition files:

- `AccessControl.auth` - Contains server-level access control definitions, such as the access controls that govern server security.
- `AccessControl.ds` - Contains DesignSync access control definitions.
- `AccessControl.hcm` - Contains module access control definitions.
- `AccessControl.ps` - Contains ProjectSync access control definitions, such as the access controls that govern projects and configurations.
- `AccessControl.psipg` - Contains ProjectSync access control definitions that are shared with IP Gear. (Customization of IP Gear access controls is not yet supported.)

Important: Do not edit any of the default `AccessControl` files supplied with DesignSync; any changes to these files will be lost upon upgrading. Instead, edit the site or server custom `AccessControl` file (see Setting Up Access Controls).

`AccessControl` files are read by the SyncServer, not by client applications (dss, dssc, stcl, stclc, DesignSync), and are used to control access to team data in a remote vault. Local vaults are unaffected by `AccessControl` files.

Note: When ACAdmin is used, there are additional files that are automatically or manually created to support ACAdmin. For more information, see Access Administrator Overview.

Related Topics

Setting Up Access Controls

Using Access Commands

Access Administrator Overview

Using Secure Access Mode

When the Access Control files are read into the DesignSync system, either at server startup or after an Access Reset, the access control definitions and rules, in the appropriate processing order are read into the system. When using Access Control in secure mode, an additional protection is added to the process; the access control definitions and rules are only updated by running the Access Reset command explicitly.

How Secure Access Control Works

When a server is started for the very first time, or an ‘access reset’ is performed, all the AccessControl files included with DesignSync and maintained in the custom ares are processed and the entire set of definitions, and rules are captured into a single consolidated file in the correct processing order. This single consolidated file is checked into the ‘AccessControlVault’.

Note: When a new version of the secure revision control file is created, a RevisionControl note is created which can be subscribed to as usual, so notification of the update can be sent out to all interested users. For more information on subscribing to revision control notes, see

About the Access Control Vault

The consolidated access control file is maintained in an access controlled vault within DesignSync. Every time the Access Reset command is performed, the DesignSync vault records any changes to the access control vault, providing a full history of the Access Reset usage. If there are no changes, DesignSync still creates a new version in the vault (though there are no content changes to the rules captured in the vault.)

The Access Control Vault cannot be operated on outside of the Access Reset command. For example a user could not populate the vault into a workspace, modify the files, and check them in. Changes are made to the custom access control files

ONLY by performing the modifications to the files and then running the Access Reset. This provides complete traceability by:

- restricting access to the Access Control files and the Access Reset command ensuring that no unauthorized changes are made.
- capturing the full history of changes, including who authorized the reset and when.
- capturing the full of set of rules associated with the access control changes.

The Access Control Vault is located on the server at the following SYNC URL:

```
sync://<host>:<port>/SYNC/AccessControl/RuleSet;
```

where:

- host - name of the machine hosting the server
- port - port number used by the server.

This is a regular files-based vault, but access is restricted by the following access control:

```
access deny $DesignSyncWriteActions everyone when Object
"sync:///SYNC/AccessControl/*"
```

The 'read' actions are allowed, and, by default, all users are allowed populate the vault or request a vhistory. The administrator can restrict that by adding an action to restrict read actions, for example:

```
access deny $DesignSyncReadActions everyone when Object
"sync:///SYNC/AccessControl/*"
```

How Access Commands Work with Secure Access Control Enabled

The custom access rules are defined using 'access deny', 'access allow', and 'access filter' commands. These commands may have been used in server-side tcl scripts to bypass Access Reset and change the access rules on the server. None of the actions are permitted in general-purpose TCL scripts when Secure Access Control is enabled. These commands are available only when read from the custom Access Control files by an Access Reset. The list of restricted commands are:

- access define
- access allow
- access deny
- access decline
- access filter
- access db_filter

Notes::

- During Access Reset, when the rules are re-read into the access control vault, these commands are active.
- Any access sub-commands commands not explicitly listed as denied are allowed. These are the commands that allow you to view access controls.

Secure Access Control Revision Control Notes

Whenever an Access Reset is performed, a new version of the consolidated rules set is checked in to the Access Control Vault, and 'ci' Revision Control (RC) Note will be created. This note is always created, even if checkins on your system don't normally create RC notes. As with any other revision control notes, the administrator can create subscriptions to get email updates whenever the Access Control Vault is updated.

For information on subscribing to Secure Access Control Revision Notes, see Subscribing to Secure Access Control Revision Notes.

Related Topics

Enabling/Disabling Secure Access Control

Format of Consolidated Rule Set

Sample Consolidated Rule Set

Setting Up Access Controls

To control access to an operation, you add stcl access control commands to a custom `AccessControl` file. Access controls are applied using a predefined set of `access` stcl commands. Each `access` command includes a keyword representing a particular DesignSync, ProjectSync, or module action. See Using Access Commands for a list of the `access` commands and examples of how to use them.

By default, the default access control files, included with DesignSync, set up open permissions for all users. These files contain the definitions for all operations that you can control and set the default access controls. See Introduction to Access Control for details on the individual files used to define access controls.

Important: Do not edit the default `AccessControl.*` files; changes will be lost upon upgrading. Instead, edit your site or server custom `AccessControl` file.

Customizing AccessControl Files

During installation, the DesignSync `sync_setup` script copies the `$SYNC_DIR/share/examples/ExampleAccessControl` file to the following locations:

- `$SYNC_SITE_CUSTOM/share/AccessControl`

  where `$SYNC_SITE_CUSTOM` defaults to `$SYNC_CUSTOM_DIR/site`.

  The site `AccessControl` file sets access controls for an entire site.

- `$SYNC_CUSTOM_DIR/servers/<host>/<port>/share/AccessControl`

  The server `AccessControl` file sets access controls for a specific SyncServer.

You use one of the custom `AccessControl` files to make your customizations. These files contain commented-out access commands that you can use as examples for creating your own access controls.

To create custom access controls:

1. Make your access control changes in the site-wide or server-specific `AccessControl` file in the custom area.

   Using the custom area ensures that your `AccessControl` files are not overwritten when you reinstall or upgrade DesignSync software and guarantees that the default `AccessControl` file correctly implements the baseline access control behavior. However, because your custom `AccessControl` files are not overwritten by new installations, you should check the `ExampleAccessControl` file for new additions after each upgrade.

2. Force the SyncServer to reread the `AccessControl` file using one of the following methods:
   - Execute the `access reset` command in a server-side Tcl script.
   - Click **Access Reset** on your ProjectSync menu.

     This option lets you update access controls without having to reset the SyncServer. The **Access Reset** option appears on the menu only when you have permission to reset the access controls. (See Access Controls for Server Administration for details.)

   - Click **Reset Server** on your ProjectSync menu.

     This option gracefully resets the SyncServer, causing the server to refresh its in-memory cache with information about the current server files on disk, including `AccessControl` files. The Reset Server option appears on the menu only when you have administration permission for the server. (See Access Controls for Server Administration for details.)

Important: In Secure Access Mode, you must perform an Access Reset in order for changes to the access control to take effect

Note: DesignSync automatically updates the consolidated Access Controls and records any changes to the access control vault, providing a full history of the Access Control changes. The consolidated Access Controls are are stored in a vault located on the server at the following SYNC URL `sync://<host>:<port>/SYNC/AccessControl/RuleSet.` When using secure mode, this update only occurs when performing an Access Reset. Otherwise it occurs when resetting or restarting the server as well.

Errors in an `AccessControl` file can cause your server to become unresponsive if the errors are not corrected quickly. To avoid this problem, correct access control errors immediately and reset the access controls. See Troubleshooting Access Controls for more information.

Note: When you are using ACAdmin and creating new access control definitions, those definitions are stored in a special custom location. For more information, see Creating a New Access Definition.

Access Control Search Order

On startup, the SyncServer loads `AccessControl` files in the following order:

1. The default access control files for the individual products in:

   ```
   $SYNC_DIR/share/AC_Components
   ```

2. The enterprise-wide file in:

   ```
   $SYNC_ENT_CUSTOM/share
   ```

   where `$SYNC_ENT_CUSTOM` defaults to `$SYNC_CUSTOM_DIR/enterprise`.

   The custom enterprise area is a placeholder for future development of enterprise-wide customizations.

3. The site-wide file in:

   ```
   $SYNC_SITE_CUSTOM/share
   ```

4. The server-specific file in:

```
$SYNC_CUSTOM_DIR/servers/<host>/<port>/share
```

Conceptually, these files are concatenated together, in the order the files are listed above.

When a user attempts an action to which more than one access rule is applicable, the order in which the rules are defined becomes important. The central principle to keep in mind is that the last applicable rule within this conceptually concatenated file wins.

For example, consider a sequence of rules, which may have been spread across the various AccessControl files, but which conceptually concatenate as follows:

```
1. access allow Checkin everyone

2. access deny Checkin users {joe}

3. access filter Checkin everyone  {
          if {[string match "guest-*" $user]} {
                  return DENY
             } else {
                  return UNKNOWN
             }
     }

4. access filter Checkin everyone when url
sync:///Projects/Public/* {
         if {[string match "*.txt" $url] {
                 return ALLOW
             } else {
                 return DENY
         }
      }
```

The rules are processed from last to first. Let's look at a few examples.

If user `mike` attempts to check in a file named `foo.txt` inside project `Ajax:`

- The fourth rule does not apply, since the project name `Ajax` does not match the `when` clause; processing continues.
- The third rule does apply since its `user` and `when` clauses match the parameters of the action being attempted. Thus, the tcl filter needs to be evaluated. In the filter script, `$user (joe)` is found not to match the pattern, and the filter script returns `UNKNOWN`. This is not a definitive allow or deny result, so processing continues.
- The second rule does not apply, because this `user` is not `joe` ; processing continues.

- The first rule does apply, and it allows the action.

Therefore, user `mike` is ultimately allowed to check in to project `Ajax`.

Next, user `mike` attempts to check in the file `x.txt` inside project `Public`,

- The fourth rule is applicable this time, and the filter script runs. In the script, the file name is found to match the pattern, and the script returns `ALLOW`.

Since this last applicable rule returned a definitive result (that is, to allow the action) there is no need for any further rules to be evaluated. The result is once again to allow the checkin.

Now user `joe` attempts to check in a file named `foo.txt` in the `Ajax` project.

- The fourth rule does not apply, since the project name `Ajax` does not match the `when` clause; processing continues.
- The third rule applies, the filter runs, and it returns `UNKNOWN`. Processing continues.
- The second rule does apply to `joe`, and it denies him the checkin.

Since this last applicable rule returned a definitive result (that is, to deny the action) there is no need for any further rules to be evaluated. Therefore, `joe` is not allowed to check in to project `Ajax`.

Now, `joe` attempts to check in a file named `foo.exe` into the `Public` project.

- The fourth rule does apply, and the filter script runs. In the `if` statement, the pattern is found not to match; the `else` clause runs, and returns `DENY`.

Since this last applicable rule returned a definitive result (that is, to deny the action) there is no need for any further rules to be evaluated. Therefore, joe is allowed to check the file in to project `Public`.

Finally, `joe` attempts to check in a file named `foo.txt` into the `Public` project.

- Once again, the fourth rule applies, but this time when its tcl script is evaluated, it returns `ALLOW`.

Since this last applicable rule returned a definitive result (that is, to allow the action) there is no need for any further rules to be evaluated. Therefore, `joe` is allowed to check the file in to project `Public`.

Note that in each case, rules keep getting evaluated, in reverse order, until an applicable rule returns a "definitive" result (`ALLOW` or `DENY`), and that a filter that returns `UNKNOWN` is treated the same way as if the rule were not applicable at all.

Also note that if there are multiple applicable filters, not all of them will necessarily have their tcl executed. Knowing this can be helpful in optimizing your custom access control files by strategically ordering the filters. It is also worth mentioning that filters that have side effects, such as logging, will not always have their tcl executed if they are "eclipsed" by another rule.

A return of DECLINE causes subsequent access rules to be applied. Return values from access rules are defined in Access Rule Commands.

If no custom `access filter` or custom access rule results in an explicit ALLOW, DENY, or DECLINE, the default access for that action is returned.

Out-of-the-box, most actions are allowed. See Introduction to Access Control for the exceptions.

Related Topics

Introduction to Access Control

Using Access Commands

Working with Server stcl Scripts in the stcl Programmer's Guide

Using Access Commands

You use the stcl programming language (a combination of the DesignSync command set and the Tcl language constructs) to define access controls. You can use these commands in your custom `AccessControl` file to define actions to be controlled and to create rules that control access.

The majority of your needs should be satisfied with simple one- or two-line commands. However, you also can use access commands in Tcl scripts outside an `AccessControl` file.

See Setting Up Access Controls for information on creating your own custom `AccessControl` files.

The access Commands

The most important concept for using the access control system is **action**, which is an operation performed by a user. An action is verified by the access control system, together with all the information needed for the verification.

Two types of commands are used in `AccessControl` files:

- **Action definition commands**, which define the actions that can be controlled.
- **Access rule commands**, which specify who can perform the actions and under what circumstances.

You use the action definitions with access rule commands to specify the access limits you need. For example, the following action definition command defines the action of adding a project.

```
access define AddProject
```

In your custom `AccessControl` file, you can use this action definition within an access rule command to control who can add a project:

```
access allow AddProject only users chris
```

This access rule command states that only Chris can add a project.

Most action definition commands include optional parameters that you can use in `when` clauses in your access rule commands. For example, the following action definition defines the action of editing a project:

```
access define ModifyProject <project>
```

When you create an access rule that uses the `ModifyProject` action, you can specify a project name as part of your rule. For example:

```
access deny ModifyProject everyone
access allow ModifyProject only users chris \
  when project "sync:///Projects/ASIC"
```

The first access rule command prevents all users from editing projects. The second access rule command, which builds on the first, states that only Chris can modify the project ASIC.

Note: When you are using ACAdmin and creating new access control definitions, those definitions are stored in a special custom location. For more information, see Creating a New Access Definition.

Action Definition Commands

An `AccessControl.*` file contains three kinds of action definitions:

- The `access define` statements, which define individual actions and optional parameters that can be passed to the action.
- The `set` statements, which define variables to represent multiple actions that can be access-controlled as a group.
- The `access init` statements, which let you create your own variables in a custom `AccessControl` file.

You use these action and variable definitions in access rule commands to control access to operations. (See Introduction to Access Control for details on the individual files used to define access controls.)

The action keywords (such as `AddProject` and `Checkout`) are case sensitive.

Do not redefine the already defined actions using `access define` statements in your site and server `AccessControl` files. You will not be able to access the server if it detects duplicate `access define` statements. You can, however, define new actions within your site or server custom `AccessControl` files.

See the Access Command Details for complete information on using these commands.

access define Commands for Individual Actions

The `AccessControl.*` files define most of the actions that you need to control access to DesignSync, ProjectSync, and module operations. (See Introduction to Access Control for details on the individual files used to define access controls.) For example, the ProjectSync `AccessControl.ps` file contains the following definition for the action of editing a user profile:

```
access define EditUser {username isSelf}
```

In this example, `EditUser` is the action of editing a user profile. The optional parameters are:

- `username`, which stands for the name of a user, as defined at `sync:///Users`.
- `isSelf`, a Boolean that takes the value `1` for the current user or `0` for any other user.

Like the `username` argument in the preceding example, the value of some parameters is assumed to be relative to a Sync URL. For example:

```
access define EditNote {system type id}
```

This definition controls who can edit notes. The `system` argument is always `sync:///Note/SyncNotes`. This parameter is ignored, but is included for future extensibility. The `type` argument takes a note type name relative to `sync:///Note/SyncNotes`. The `id` argument stands for the note ID number.

set Commands for Groups of Actions

The `AccessControl`.* files also define sets of actions that control related groups of commands. (See Introduction to Access Control for details on the individual files used to define access controls.) You can use the variable you set for these groups of actions in access rule commands to control access to all the defined commands.

For example, the `AccessControl` file contains:

```
set UserActions {AddUser EditUser DeleteUser EmailSubscribe}
```

`AddUser`, `EditUser`, `DeleteUser`, and `EmailSubscribe` are individual actions defined with the `access define` command. You use the defined variable `$UserActions` in `access allow` or `access define` statements to control access to adding, editing, and deleting users and for creating email subscriptions for users.

For example, to ensure that only your project leaders (Chan, Lynch, and Kapoor) can perform these actions, you could create the following access rule:

```
access allow $UserActions only users {chan lynch kapoor}
```

In your custom `AccessControl` files, you also can use this syntax to create variables for your own sets of user actions.

See Access Controls for Groups of Commands (ProjectSync) and Access Controls for Groups of Commands (DesignSync) for details on the predefined sets of commands for these applications.

access init Commands for Custom Variables

You can define your own variables for use in access rule commands by specifying the variables inside `access init` blocks in your custom `AccessControl` file. For example:

```
access init{
  set admin {syncmgr}
  set projectLeaders {chan lynch kapoor}
}
```

You now can use the variables `$admin` and `$projectLeaders` in your access rule commands. Using the preceding example, it makes more sense to create your access rule command using the `$projectLeaders` variable than using the individual user names:

```
access allow $UserActions only users $projectLeaders
```

You also can use `access init` to create variables and procs for use in access filters. However, when `access init` is used in filters, the `access init` statement is sourced each time the filter is run. This behavior can introduce performance penalties for operations such as viewing a note because the statement is sourced for each note. The `access global` command is used inside filter scripts and is sourced only once, when the access control system is initialized. See the `access init` and `access global` command descriptions and Sample Access Controls (DesignSync) for more information.

Access Rule Commands

There are four possible outcomes of access rules checking:

- ALLOW

  Access is explicitly granted.
- DENY

  Access is explicitly denied.
- DECLINE

  Access to perform the requested operation on an entire module is not granted nor denied. Subsequent access rules will be applied to the module's individual members. See the Declining Access section below for details.
- UNKNOWN

  Access is not explicitly allowed, denied, or declined. The custom `access filter` that returned the UNKNOWN value had no effect. The result is delegated to other applicable access rules that will be evaluated next. This delegation will continue in order until an access filter explicitly allows, denies, or declines access. If no custom `access filter` or custom access rule results in an explicit ALLOW, DENY, or DECLINE, the default access for that action is returned.

  Out-of-the-box, most actions are allowed. See Introduction to Access Control for the exceptions.

Corresponding Tcl return codes are provided in the `access verify` command documentation.

Allowing and Denying Access

You create access rules using the `access allow` and `access deny` commands to specify the access granted to users. You can allow or deny access to:

- `everyone` - Applies to all users on the server.
- `users` - Applies to the users specified in the list.
- `only users` - Applies only to the users in the list; people not in the list are excluded.

For example:

```
access allow {AddTrigger EditTrigger} only users $admin
```

In this example, only users defined as administrators can add or edit triggers; all other users are denied permission.

You optionally can include parameters with `when` clauses to further refine control. For example:

```
access allow $NoteActions only users $admin
  when type "AdminNote"
```

In this example, only administrators can perform the set of actions described by the variable `$NoteActions` (a group of methods for interacting with notes) when the note type is AdminNote.

The parameters specified in `when` clauses are the parameters declared in action definitions. The `NoteActions` actions have `Type` parameters that store the type of note. See ProjectSync Action Definitions to find out the parameters for each ProjectSync action; see DesignSync Action Definitions to find parameters for each DesignSync action. See User Authentication Action Definitions for the parameters defined for user authentication access controls.

You can use multiple `when` statements. For example:

```
access deny ModifyNoteProperty everyone \
  when type "SyncDefect" \
  when field "AuditTrail"
```

In this example, no one can edit the **Audit Trail** field of the SyncDefect note type.

Access control rules are order-dependent, so later rules qualify earlier ones. You can use this behavior to refine your access controls. For example:

```
access allow $UserActions only users $admin
access allow EditUser everyone when isSelf 1
```

In this example, only administrators can perform the set of actions described by the variable `$UserActions` (working with user profiles and email subscriptions); however, users can edit their own user profiles.

See Using Access Command Qualifiers for more information on the qualifiers you can use with access rules.

Declining Access

The DECLINE outcome defined above is used with module data. When you operate on a non-legacy module, you are operating on the module as well as on the module's individual members. Access control rules can apply at the module level, and also at the individual module member level. This is accomplished by using the `access decline` command. The `access decline` command documentation includes sample access controls showing how to use the `access decline` command.

An operation on module data first checks module-level access. If the module-level access is allowed or denied, then no further checks are necessary. However, if the module-level access is declined, then an access check is needed for each individual module member participating in the operation.

For example, `ci` of a module first checks module-level access via the `Checkin` access control. Permission to add or remove hierarchical references may also be checked. See Creating a New Version of a Module for details. If module-level `Checkin` access is declined, then `ci` permission of each module member to be included in the new version of the module is checked, via the `MemberCheckin` access control. If `Checkin` access for the module is declined, and there isn't an applicable custom access statement for `MemberCheckin`, all of the module's members will be granted `MemberCheckin` access. That is because out-of-the-box, most actions are allowed. See Introduction to Access Control for the exceptions.

Access controls for DesignSync commands that operate on module data are described in the sub-book "Using DesignSync Access Controls with Modules", within the book "Access Controls for Modules". Attempting to decline access for a command that does not support the DECLINE outcome returns DENY.

Filtering Access Controls

You can put optional arguments into `access filter` scripts and combine them with Tcl scripts to control specific events. For example:

```
access filter ModifyNoteProperty \
  when type "HW-Defect-1" \
  when field "Status" \
  when newval "Closed" \
{
  if {[lsearch $projectleaders $user] != -1} {
    return ALLOW
  }
  return DENY
}
```

In this example, the access control allows only a project leader to change the state of a HW-Defect-1 note to "Closed." The Tcl script in the curly braces searches a list of project leaders for `$user`. If the user name is found, the operation is allowed; if not, the operation is denied.

Access Command Details

A list of access commands and their descriptions is in Access Command Details.

Related Topics

Modules Action Definitions

Introduction to Access Control

ProjectSync Action Definitions

DesignSync Action Definitions

Setting Up Access Controls

User Authentication Action Definitions

Using Access Command Qualifiers

Using Access Command Qualifiers

When creating access control rules, you use qualifiers to specify:

- The user who performs the action governed by the rule.
- The parameters that limit the scope of the rule (`when` clauses).
- The visibility of ProjectSync GUI options when an action is partially restricted.
- The explanatory message that appears when a DesignSync access fails (`-because` clauses).

See Setting Up Access Controls for information on creating your own custom `AccessControl` files.

User Qualifiers

All access control rules require information on the user performing the action. The user qualifiers follow the action keyword in an access rule. You can use one of three user qualifiers in an access rule:

- `users` - Lets you allow or deny access to the specified users but does not exclude others from the same type of access. For example, suppose you have the following access rules:

  ```
  access deny Delete users {george martha}
  access deny Delete users {bonnie clyde}
  ```

  In this case, none of the four specified users is allowed to delete objects.

- `only users` - Lets you allow or deny access only to the specified users and excludes all others from the rule. For example, suppose you have the following access rules:

  ```
  access deny Delete only users {george martha}
  access deny Delete only users {bonnie clyde}
  ```

  In this case, the second rule overrides the first and only Bonnie and Clyde are prevented from deleting objects.

- `everyone` - Lets you allow or deny access to all users. For example:

  ```
  access deny Delete everyone
  ```

  This rule prevents everyone from deleting objects.

Using access allow|deny without the only Qualifier

Many `access allow` and `access deny` examples use the `only` qualifier to allow or deny a user or group of users access to an operation. A common use of the `only` qualifier is to deny access to everyone while allowing access only for specified users.

Access control qualifiers such as `everyone` and `only users` are very useful for defining the basic access rights, but are not useful for handling special cases. In many scenarios it is more convenient to modify the access rights incrementally than to set them for everyone in a single rule.

Here are some scenarios that illustrate when you might use `access allow` and `access deny` statements without the `only` qualifier:

- Ensuring that access controls of separate teams do not conflict.

  Suppose that your server supports multiple teams of users and each team has its own section of the `AccessControl` file. Another likely scenario is that each team has its own file that is sourced by the central `AccessControl` file. If each team uses the `only` qualifier to modify the access rights, only the team whose file is sourced last has any rights to access the server. Thus, using `access allow` and `access deny` statements without the `only` qualifier lets users on one team add their own rights without interfering with the access rights of other teams.

- Ensuring that an access filter is not overridden.

  Suppose the access rights are determined by a chain of two rules: the first is an `access filter` script and the second is an `access allow|deny` statement. Whether a user is granted access by the filter depends on the filter code and the values of the filter's parameters; therefore, the filter's result is variable and not fixed. Yet, if the subsequent `access allow|deny` statement uses an `only` qualifier, then the result of the filter is always ignored. Specifying the subsequent `access allow|deny` statement without the `only` qualifier lets you add rights for a single user without overriding the previous rule.

- Adding a 'superuser' without modifying existing rules.

  The easiest way to add a 'superuser' to your server is to add a rule at the end of the `AccessControl` file that grants access to the actions you want to allow the 'superuser' the right to perform:

  ```
  access allow $NoteActions users jackson
  ```

  This additional access control rule has no effect on previous access controls.

- Adding a rule to prevent a user from performing operations.

  Just as you can add a single statement at the end of the `AccessControl` file to allow permissions for a 'superuser', you can add a single `access deny` statement to the end of the `AccessControl` file to prevent a user from performing particular operations:

  ```
  access deny $DesignSyncActions users delano
  ```

The alternative to this single `access deny` statement is to edit every rule that uses an `only` qualifier.

See Sample Access Controls for other examples of using user qualifiers. See Access Control Scripting for examples of using qualifiers in scripts.

Parameter Qualifiers

Most action definitions include one or more parameters. You can use these parameters in access rules by putting them in `when` clauses in `access allow` and `access deny` commands.

For example, the action definition for deleting a note is:

```
access define DeleteNote {system type id isAuthor}
```

You can use the `<isAuthor>` parameter in a `when` clause in an access rule to ensure that only the authors of a note can delete the note:

```
access deny DeleteNote everyone

access allow DeleteNote everyone when isAuthor 1
```

In this example, the first rule prevents everyone from deleting notes. The second rule builds on the first and uses the Boolean `<isAuthor>` parameter to allow the author of the note to delete it.

You can use multiple `when` clauses in an access rule. For example:

```
access allow DeleteNote everyone \
when type SyncDefect \
when isAuthor 1
```

In this example, only the note author can delete a note and only if it is a SyncDefect.

See ProjectSync Action Definitions to find out the parameters for each ProjectSync action; see Revision Control Action Definitions to find parameters for each DesignSync action; and see User Authentication Action Definitions for the parameters defined for user authentication access controls.

See Sample Access Controls (DesignSync) for other examples of using `when` clauses with DesignSync action definition parameters. See Sample Access Controls (ProjectSync) for other examples of using `when` clauses with ProjectSync action definition parameters.

You also can access parameters within filter scripts you create using `access filter` statements. For each action, implied parameters, `<user>` and `<action>`, also are passed to filter scripts. The `<user>` parameter contains the user name of the user attempting the action. The `<action>` parameter contains the type of action that has triggered the filter script. See Access Control Scripting for examples of access filter scripts that include these action parameters.

The * Parameter Qualifier

In ProjectSync, selections for an action do not appear in the GUI when the user does not have permission to perform the action. However, when the action is partially accessible to the user, the selection is displayed. For example, if a user cannot modify any projects, the **Project - Edit** menu selection is removed from the user's version of the ProjectSync menu. But if the user can modify any project, the **Project Edit** is displayed.

To support this behavior, an asterisk (`*`) is used as a special parameter qualifier in ProjectSync access rules. If access to an action is allowed for any value of a parameter, it is allowed for the special value `*` of that parameter.

For example, you might create the following access rules to specify that test2 is the only configuration that users are allowed to edit:

```
access deny ModifyConfig everyone

access allow ModifyConfig everyone when config test2

access allow ModifyConfig everyone when config {[*]}
```

In this example, if users are allowed to edit the test2 `config` parameter value, you also need to allow edit access for the * value of that parameter. Using the `*` parameter ensures that the GUI displays the selections that allow users to edit configurations. However, if users attempt to edit any configuration other than test2, they see a message telling them that they do not have permission to edit the configuration.

When using `*` in a `when` clause, you must escape the `*` character as shown in the preceding example: `when config {[*]}`. If you do not escape the asterisk, it is treated as a wildcard and the `when` clause matches all configurations.

The `*` qualifier is most useful in access control scripts that use the `access verify` command. When you use the `*` qualifier with the `access verify` command, ProjectSync verifies the `*` and does not need to verify each project separately to see whether it can be modified:

```
access verify ModifyProject *
```

When you write filters, follow this convention to prevent parts of the ProjectSync GUI from becoming inaccessible to users who have permission to perform actions under certain circumstances.

The following sample filter allows any user to modify only configurations named `scratch` (of any project):

```
access filter ModifyConfig {
  if {$config == "*"} {
  return ALLOW
  } elseif {$config == "scratch"} {
  return ALLOW
  } else {
  return DENY
  }
}
```

In this example, if the check for `$config == "*"` is removed, the GUI selections for modifying configurations are removed, even though the user has permission to modify some configurations.

You also can use the `*` mechanism in the Edit note panel. Here, each field is verified as follows:

```
access verify ModifyNoteProperty <sys> <type> <id> <field>
<oldval> *
```

This example can be paraphrased as "May I change this field at all?" If the verification returns DENY, the field displays as non-editable text on the panel.

The -because Qualifier

When you create an access rule for a DesignSync operation, you optionally can include a `-because` qualifier to give users information on the operation. For example:

```
access deny Unlock everyone \
-because "To unlock, you must be the lock owner or an admin."

access allow Unlock everyone when IsLockOwner "yes"

access allow Unlock users $admin
```

The first rule prevents all users from unlocking an object. The `-because` clause generates an explanatory message to users who are denied permission to unlock a file. The second and third rules build on the first rule to allow the lock owner and administrators to unlock files.

A `-because` clause generates the message in the command area of the DesignSync GUI or on the command line when you run DesignSync from a shell. You do not need to add `-because` clauses to ProjectSync, which has its own built-in set of messages.

Related Topics

Sample Access Controls (DesignSync)

Sample Access Controls (ProjectSync)

Setting Up Access Controls

Using Access Commands

Using Access Command Qualifiers

Creating New Access Definitions

In addition to assigning existing access definitions to users and groups and creating categories of commands by grouping existing access definitions, you can also create your own access definitions which can then be used individually or grouped into a category and assigned to users or groups for their use. This allows you to extend the functionality of access controls beyond what is provided by DesignSync.

Where you define the access definitions depends on whether you are using ACAdmin or manually creating your access controls with the included Access Controls files.

Note: This file is only used when ACAdmin is in use.

Creating Access Definitions while Using DesignSync ACAdmin

When you create access definitions while using ACAdmin, you store the custom access control definition in the AccessControl.aca_definitions file. This file is located in the $SYNC_CUSTOM_DIR/servers/<ServerName>/<PortNumber>/share/AC_Components

Creating Access Definitions:

1. Create or edit the access control definitions file:
   `$SYNC_CUSTOM_DIR/servers/<ServerName>/<PortNumber>/share/AC_Components/AccessControl.aca_definitions`
2. Add the definition to the file in the folowing format:
   `access define <CommandName> [<Arguments>]`
3. Reset Access Controls.

Creating Access Definitions while using DesignSync without ACAdmin

When you create access definitions while not using ACAdmin, you store the custom access control definition in the access controls file. This file is located in one of two possible locations depending on the applicable of the access controls:

```
$SYNC_CUSTOM_DIR/site/share/AcessControl
 $SYNC_CUSTOM_DIR/server/<host>/<port>/share/AcessControl
```

Creating Access Definitions:

1. Select the appropriate Access Control file to store change as described in Setting Up Access Controls
2. Add the definition to the file in format:
   `access define <CommandName> [<Arguments>]`
3. Reset Access Controls

Related Topics

Setting Up Access Controls

Access Administrator Overview

Enterprise DesignSync Access Maps

When DesignSync is used as part of an enterprise system, DesignSync and Enterprise Development objects are maintained, in synchronization, across both platforms. In order to best support this, DesignSync provides access maps to map from the DesignSync to the ENOVIA Enterprise commands.

Note: The platform server must have Access Control mapping delegations enabled. For more information see the *DesignSync Administrator's Guide*: SyncAdmin Enterprise Servers.

Mapping DesignSync Commands for use with the Enterprise System

When using DesignSync as part of your comprehensive Enterprise solution, and the Enterprise server is defined as managing access control, DesignSync provides some default mappings and some sample mapping to support the delegation functionality.

The DesignSync server platform access maps are included in the default `AccessControl.hcm` file.

By default, the following access maps are enabled when access control delegation is enabledl.

- Access Controls for Checking In
- Access Controls for Adding Hierarchical References
- Access Controls for Removing Hierarchical References

Additional access maps are provided as samples and located in the custom server directory access controls file:
`$SYNC_CUSTOM_DIR/servers/<`*`serverName`*`>/<`*`portNumber`*`>/share/AccessControl`

Examples of Access Controls

All custom access controls should be placed in custom AccessControl files. See Customizing AccessControl Files for details.

For examples of access controls, see

- Sample Access Controls for DesignSync
- Sample Access Controls for ProjectSync
- Sample Access Controls for SyncServer access

There are also sample access controls in the installed `<SYNC_DIR>/share/examples/ExampleAccessControl` file. During DesignSync installation, the `sync_setup` script creates site-wide and server-specific `AccessControl` files by copying the installed `ExampleAccessControl` file to a site or server `AccessControl` file, if an `AccessControl` file does not already exist in those locations.

Access Administrator (ACAdmin)

Access Administrator Overview

The Access Control Admin provides a graphical web interface to create, remove, maintain, and manage access controls. This provides a simpler, more intuitive interface to customizing the access controls for DesignSync. By default, the Access Administrator functionality is disabled. In order to use the Access Control Administrator (ACAdmin), you must enable it using the Access Control registry setting: Enable ACAdmin described in the *ENOVIA Synchronicity DeisgnSync Data Manager Administrator's Guide*.

Important: Access Control processing order (precedence) is determined by the priority set on the command category. The higher the priority, the higher the precedence.

When you enter commands through the Access Administrator tool, you have full control over the priority settings in the command category. If you need to change the processing order, change the priority setting for the command category and reset ACAdmin.

Using ACAdmin provides a graphical interface for organizing existing definitions into command categories, and assigning existing definitions to users and groups. Any new definitions must be created as described in Creating a New Access Definition before they can be used by ACAdmin.

IMPORTANT: In order to use the Access Administrator tool, you must enable the functionality. For more information on enabling the Access Administrator tool, see the Access Control Registry Settings.

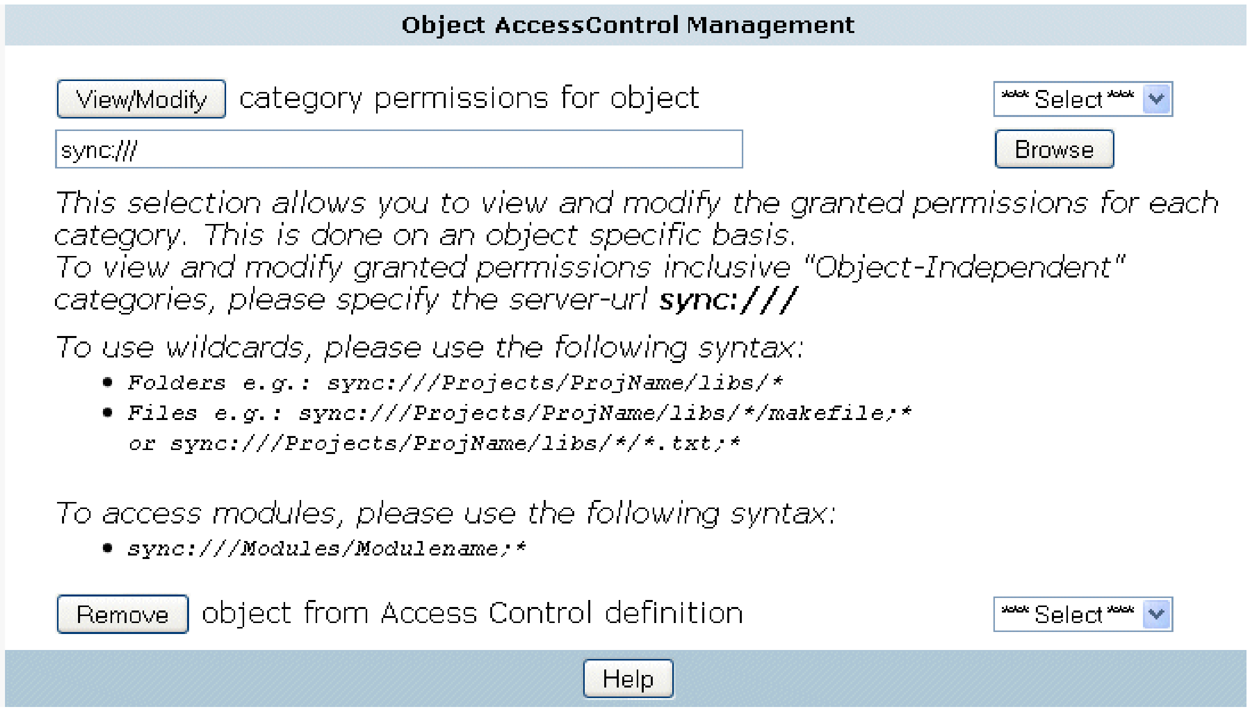
Permission Management

Object-Oriented AccessControl Management

This section controls access policies for objects on the server. Permissions are the access rights granted for users at various objects or areas of the vault. The access rights granted at each object or area of the vault are expressed using a set of users and user groups for each category.

Once the changes have been made, they are immediately available for DesignSync.

Click on the fields in the following illustration for information.


Object AccessControl Management
View/Modify
category permissions for object
*** Select ***
sync:///
Browse

This selection allows you to view and modify the granted permissions for each category. This is done on an object specific basis.
To view and modify granted permissions inclusive "Object-Independent" categories, please specify the server-url **sync:///**

To use wildcards, please use the following syntax:

- *Folders e.g.: sync:///Projects/ProjName/libs/**
- *Files e.g.: sync:///Projects/ProjName/libs/*/makefile;**
  or sync:///Projects/ProjName/libs//*.txt;**

To access modules, please use the following syntax:

- *sync:///Modules/Modulename;**

View/Modify

Launches the Permissions Definitions page for the selected object.

Object Selection

Enter the Sync URL of the object on which to set the permissions. You can set the permissions for any object on the server. You may specify objects as follows:

Object URL	Description
`sync:///`	Controls the default permissions for all object-independent actions on the server. For example, user creation and removal actions do not depend on specific server objects. (Default)
`sync:///*`	Controls the default permissions for all object-dependent actions on the server. For example, browsing objects on the server, creating or modifying modules, etc. depend on having access to related server objects.
`sync:///Projects[/`*`ProjectName`*`]/[`*`pathtoObject`*`]`	Controls the permission for the specified project or the entire project area on the server.
`sync:///Modules[/`*`Category`*`[...]][/`*`ModName`*`]`	Controls the permission for the specified module, category, or entire module area

	on the server.

Notes:

- You can use wildcards, such as * to specify all matching objects within the specified path, for example:

  ```
  sync:///Modules/Chip/*.c
  ```

  controls access to all .c files within the module Chip. If Chip was a category and you wanted to specify all modules within the category, you could specify the URL either of the following ways:

  ```
  sync:///Modules/Chip/*
  ```

  ```
  sync:///Modules/Chip
  ```

- The first two URLs in the table are generic Sync URLs that can be used to provide default access for the server.

Select

Contains a drop-down list of all Sync URLs with defined access permissions. Select a Sync URL from the list to populate the Object Selection box.

Note: Use the Select drop-down to select the object definitions to modify or remove. To create a new access definition, use the Browse button, or manually enter the Sync URL.

Browse

Allows you to select specific objects, including files, folders, vaults, or modules. For more information on using the Browse to navigate your server, or a different server, see *ENOVIA Synchronicity ProjectSync User's Guide*: Using the Object Browser.

Remove

Removes the permission definitions for the selected object.

Note: You may only remove one object definition at a time using this interface.

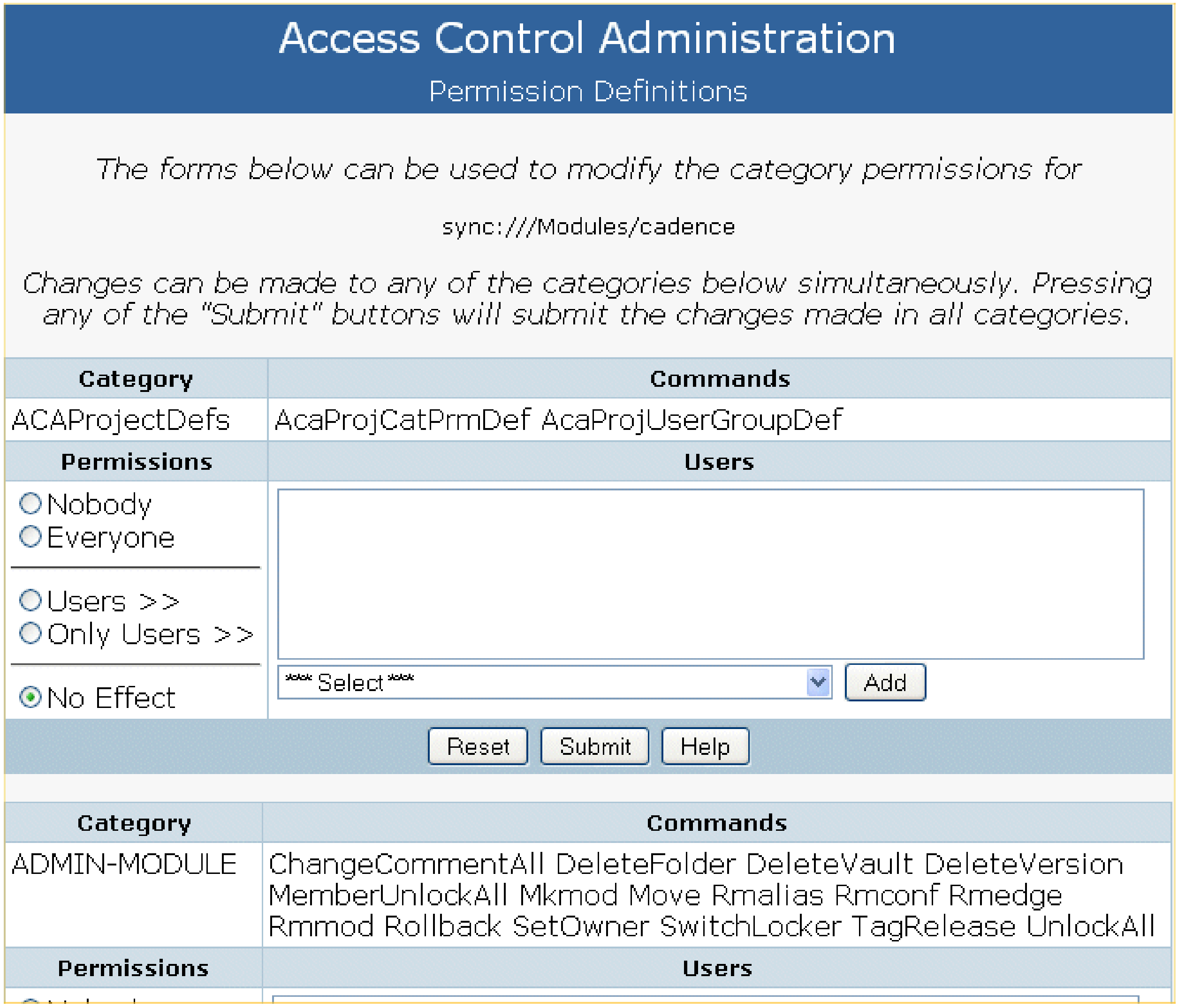
Permission Definitions

This page allows you to create or modify access policies for specified objects on the server.

To define permissions:

1. Enter the object in the Object AccessControl Management section of the Access Administrator and click on the **View/Modify** button. The Permission Definitions page launches and displays a list of the defined categories containing the permissions to associate with the object.
2. Select the appropriate categories and users or user groups to apply to the object.
3. Press **Submit** to accept the changes or **Reset** to clear any changes made. **Submit** sends all modifications made on the page to the server, not just the ones for the particular section containing the selected **Submit** button. Selecting **Submit** launches the preview pending updates page.
4. On the preview pending updates page, you can review the list of changes. To commit the changes, select **Submit**. To refine the permissions further, select **Back** to return to the Permission Definitions page.

Click on the fields in the following illustration for information.



Access Control Administration

Permission Definitions

The forms below can be used to modify the category permissions for

sync:///Modules/cadence

Changes can be made to any of the categories below simultaneously. Pressing any of the "Submit" buttons will submit the changes made in all categories.

Category	Commands
ACAProjectDefs	AcaProjCatPrmDef AcaProjUserGroupDef
Permissions	**Users**
○ Nobody ○ Everyone ○ Users >> ○ Only Users >> [✓ No Effect]	*** Select *** Add

Reset Submit Help

Category	Commands
ADMIN-MODULE	ChangeCommentAll DeleteFolder DeleteVault DeleteVersion MemberUnlockAll Mkmod Move Rmalias Rmconf Rmedge Rmmod Rollback SetOwner SwitchLocker TagRelease UnlockAll
Permissions	**Users**

Category

The name of a defined command category. You may apply any command category to the selected object.

Permissions

Defines the grant level of this permissions definition. The grant level is one of the following options:

- Everyone - Grant all users the defined access to the commands included in the command category for the specified object.
- Nobody - Grant no users the defined access to the commands included in the command category for the specified object.
- Users - Grant specified user groups and users access to defined commands included in the command category for the specified object. If the user group or

user is on the list, then operation is allowed. If not, the AccessControl system continues processing remaining AccessControl directives to see if any other rule grants the permission.

- Only Users - Grant only the specified user groups or user access to defined commands included in the command category for the specified object. If the user is not on this list, the AccessControl system immediately denies access to the user and does not process additional AccessControl directives.
- No Effect - Do not apply this command category to the specified object. (Default)

Important: How access controls are defined can affect processing order. Use the command categories priorities to set processing order. Commands with higher priority are processed first.

Commands

A list of the commands contained in the command category. To change the commands associated with the command category, you must modify the command category.

Users

The user group or specific users associated with the command category for the specified object. You may specify as many users or user groups as desired.

Using the pick list beneath the text box, you may view the entire list of defined users and groups. You can select one and press **Add** to add it to the text box, or type a user or group name in the text box to add a user.

To remove a user, highlight the user or group name and press the **Delete** key.

To create user groups, see Create User Groups. To modify existing user groups, see Modify User Groups. To create user profiles, see *ENVOIA Synchronicity DesignSync Data Manager Administration Guide*: Creating User Profiles.

Update Permission Definitions

This page allows you to review the changes you've made to the permissions definitions.

Accept the listed changes by pressing the Submit button, or modify the changes by pressing the back button to return to the Permissions Definitions page.

User Group Management

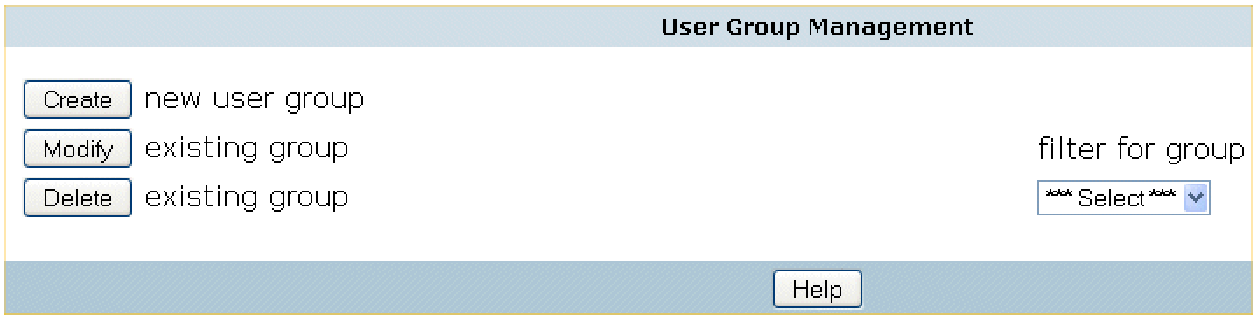
User Group Management

User groups allow you to group users will similar roles into one element that can be used in an access definition. These defined groups can now be used when defining the access to categories on objects.

Once the users are defined, the Project Managers use the Object-Oriented AccessControl Management screens to assign permissions to the user groups.

User Groups are used in access definitions for objects and command categories like any other user on the system. User groups can be associated with an organizing "filter" tag that allows you to group users and groups into a logical group, for example by project team, or department.

Click on the fields in the following illustration for information.



Create

Select this option to create a new user group.

Modify

Click the **Modify** button to modify user groups from the list of available user groups.

Delete

Click the **Delete** button to delete user groups from the list of available user groups.

filter for group

Select one of the available filters. By default, there are no filters available in this list. Filters become available when associated with a user group. When you select a filter, only user groups associated with that filter show up

When you have a large number of defined users and groups, it can be difficult to locate the specific users to modify in the list. The filter organization provides a way to identify users who are associated with a project or module so that you can restrict the view to show you only possible target user groups.

Note: The filter does not restrict the user group to, for example, a specific module, category, or project. It is strictly a display filter.

Create User Groups

This page allows you to create user groups. These user groups are then available to be assigned to command categories. Grouping users into groups allows you to define and assign roles and maintain the roles and functions easily even when individual group members change.

IMPORTANT: External group names cannot collide with existing group names. To avoiding naming collisions, it is highly recommended that you use a consistent naming convention for both internal and external names. One example of a naming convention would be:

- Internal users - username in all lower case (ie: rsmith)
- Internal groups - group name begins with capital letter, or each significant piece of the group names begins with a capital letter: (Admins, ModuleAdmins)
- External groups - being with a fixed prefix (SITE_Admins, SITE_ProjectLeaders)

This will prevent a collisions as well as being obvious visually to the user whether the user is seeing a group or user, and whether a group is internal to the ACadmin system on the server or available across servers.

Notes:

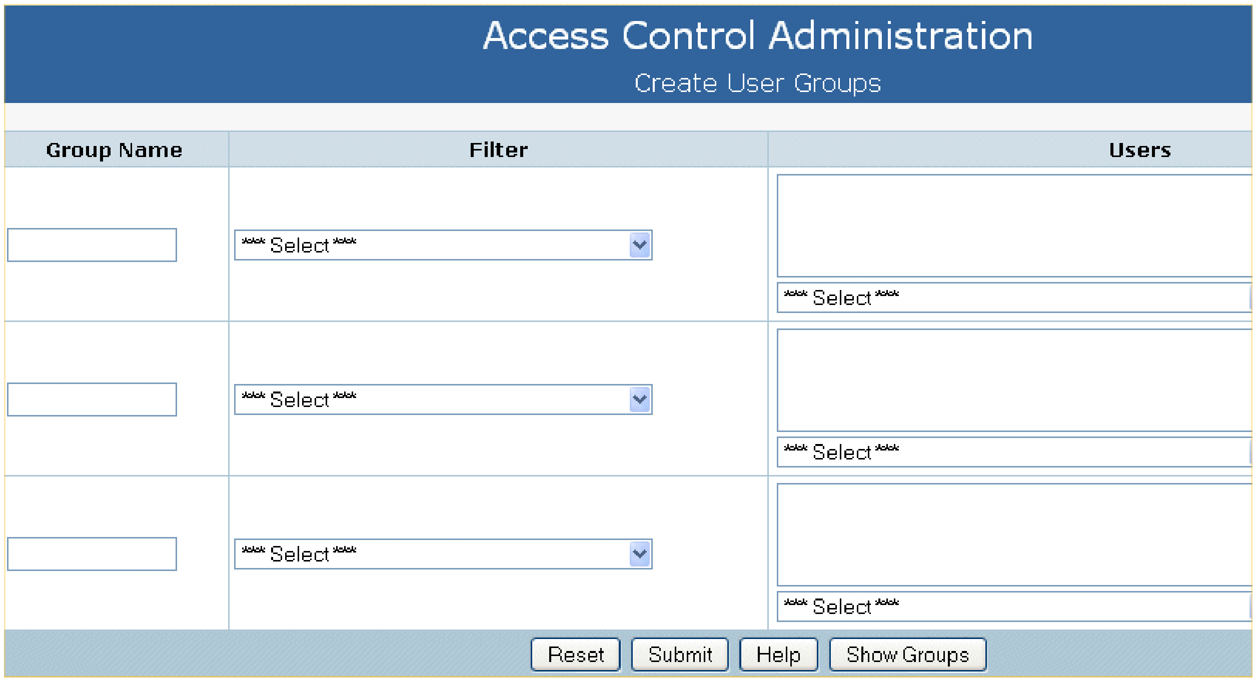
- Usually only the DesignSync administrators should have permission to create or modify user group definitions. The access control that determines whether a user can create or modify user group definitions is `AcaProjUserGroupDef`, This is included in the command group ACAactions assigned to the SVR_ADMIN category. For more information on ACActions, see Access Controls for Groups of Commands.
- There are three dynamic (or virtual) user groups that should never be manipulated manually. They are All-Module-Owners, All-Project-Owners, and All-Server-Users. These groups are automatically generated when ACAdmin is reset. If you have made changes that affect these groups, you should perform an ACAdmin Reset. For more information on these dynamic user groups, see Special User Groups.

To create User Groups:

1. Select Create in the User Management section of the Access Administrator.
2. Name the user group and select the desired options on the Create User Category page.
3. Select Submit to review your selections on the Preview Pending Command Category Update page.
4. To accept the changes, press **Submit**. If the command is successful, you see a confirmation page. If the command was not successful, you see an error page

explaining the failure. If you do not accept the changes, you may click **Reset** to cancel the changes.

Click on the fields in the following illustration for information.



Group Name

Name of the user group. This name should correspond to DesignSync naming conventions. For a list of reserved characters that should not be used in the user group name, see the *ENOVIA Synchronicity DesignSync Data Manager User's Guide*: URL Syntax.

Tip: To allow you to easily differentiate between user groups and individual users, you should implement a naming convention such as prefixing the user group with a standard prefix like Group- or by creating group names in all capital letters. For example:

```
DEVELOPERS
```

or

```
GROUP-Developers
```

Filter

An identifier that corresponds to a project, group of projects, category, module, or group of modules. By default, the filter list contains all projects and modules defined on the server. This list can be changed by providing a custom method for defining available filters. For information on defining custom filters, see Customizing Adaptable Functions using the acaCallbacks.tcl File.

The filter is used to provide a meaningful association for the user group.

Users

Lists the users associated with the user group. You may associate individual users or other user groups. For example, you may have a group called ProjectManagers that contains defined project manager user groups from different product groups.

Existing User Group

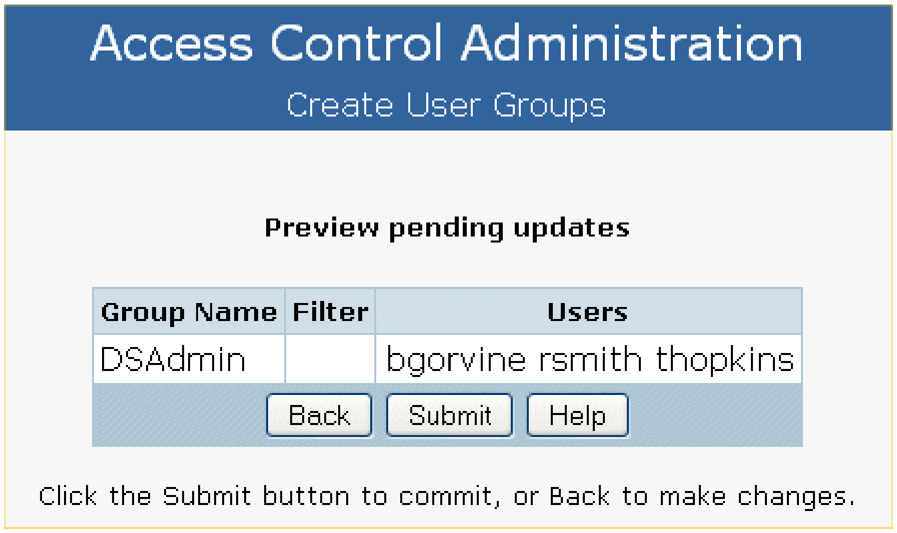
This list displays the existing user groups and their properties.

Select **Dismiss** to close the window.

Preview Pending User Group Updates

This page allows you to review the changes you've made to user groups by creating, modifying, or deleting user groups and confirm the changes before committing them to the server.

Accept the listed changes by pressing the **Submit** button, or modify the changes by pressing the **Back** button to return to the Create User Groups, Modify User Groups or Delete User Groups page.



Preview pending updates

Group Name	Filter	Users
DSAdmin		bgorvine rsmith thopkins

Click the Submit button to commit, or Back to make changes.

Modify User Groups

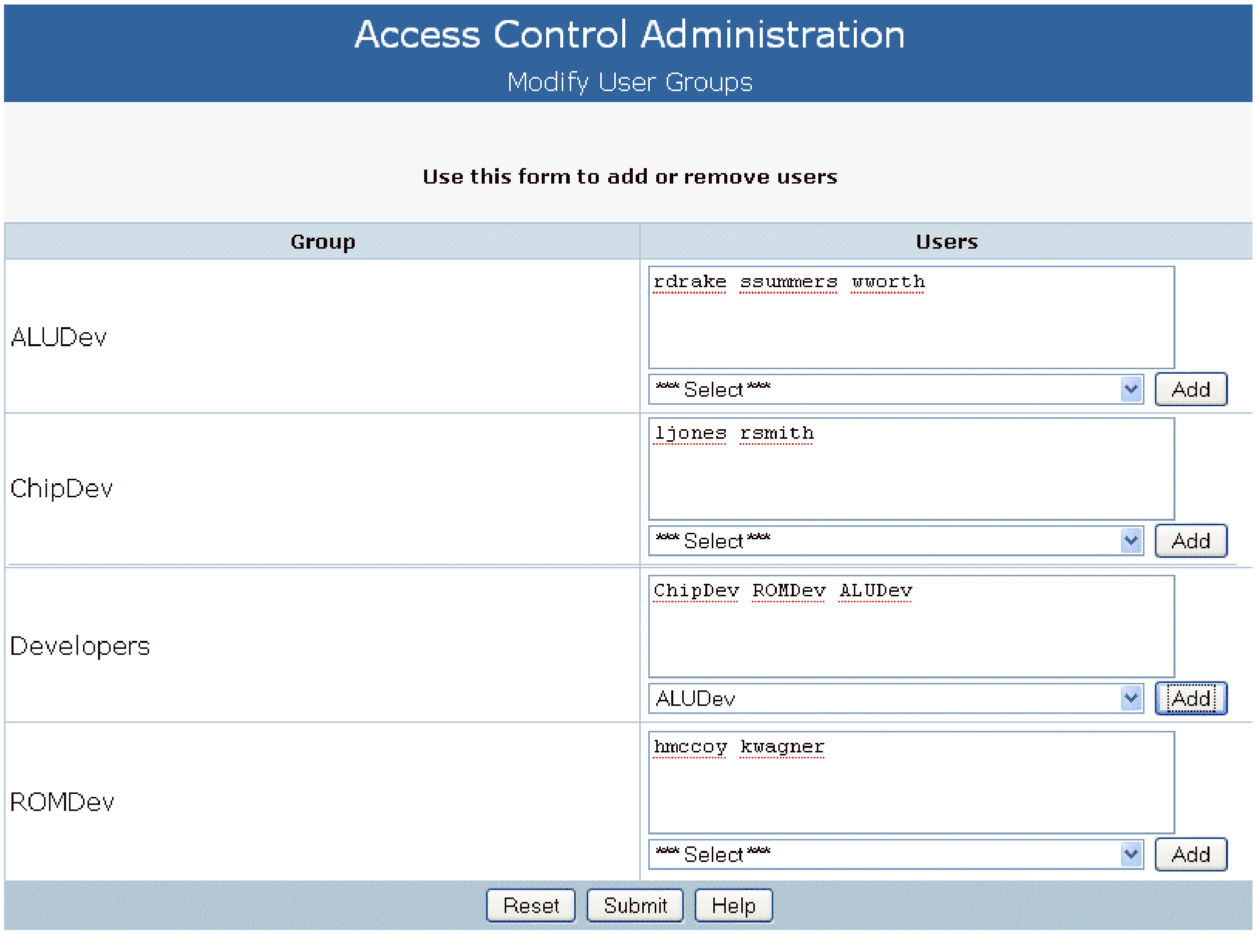
This page allows you to modify existing user groups. Grouping users into groups allows you to define and assign roles and maintain the roles and functions easily even when specific group members change.

Note: Usually only the DesignSync administrators should have permission to create or modify user group definitions. The access control that determines whether a user can create or modify user group definitions is `AcaProjUserGroupDef`, This is included in the command group ACAactions assigned to the SVR_ADMIN category. For more information on ACActions, see Access Controls for Groups of Commands.

To modify User Groups:

1. Select **Modify** in the User Group Management section of the Access Administrator. This loads the Modify User Groups page with a list of all the defined users and groups.
2. Select the desired options on the Modify User Groups page.
3. Select **Submit** to review your selections on the Preview Pending User Group Update page.
4. To accept the changes, press **Submit**. If the command is successful, you see a confirmation page. If the command was not successful, you see an error page explaining the failure. If you do not accept the changes, you may click **Reset** to refresh the form from the server, erasing any unsaved modifications.

Click on the fields in the following illustration for information.


Access Control Administration

Modify User Groups

Use this form to add or remove users

Group	Users
ALUDev	rdrake ssummers wworth
ChipDev	ljones rsmith
Developers	ChipDev ROMDev ALUDev
ROMDev	hmccoy kwagner

Group Name

Name of the user group. You cannot modify the user group name.

Users

Lists the users associated with the user group. You may associate individual users or other user groups. For example, you may have a group called ProjectManagers that contains defined project manager user groups from different product groups.

To add a user to the list, you may type the user or user group name in the text box, or select it from the drop-down list and click the **Add** button.

To remove a user from the list, highlight the username and press the **Delete** key.

Delete User Groups

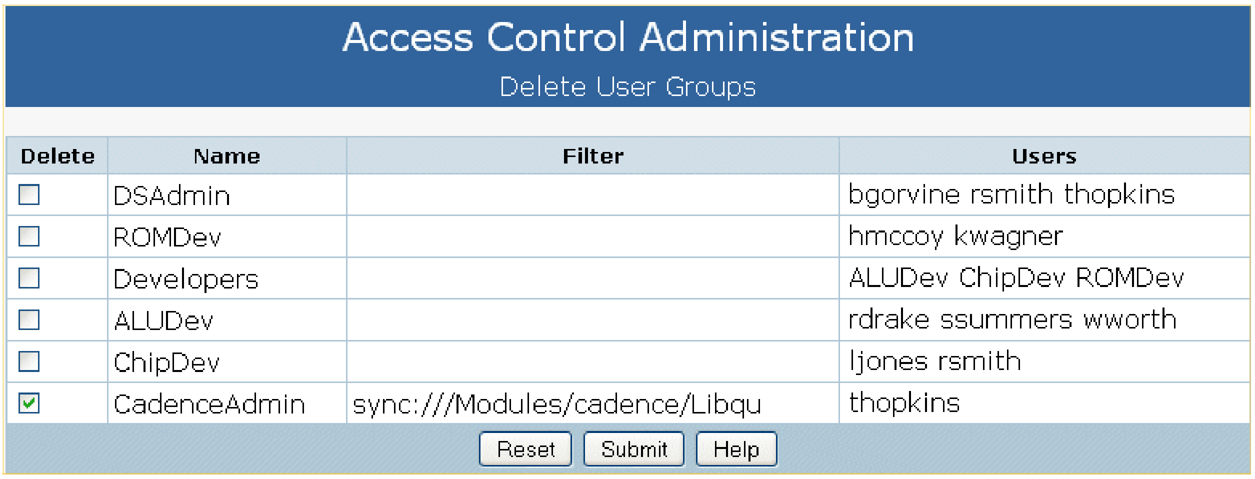
This page allows you to delete existing user groups. This deletes the group definition and removes it from any access controls. The access controls remain in place in the same processing order.

Note: Usually only the DesignSync administrators should have permission to create, modify, or delete user group definitions. The access control that determines whether a user can create or modify user group definitions is `AcaProjUserGroupDef`, This is included in the command group ACAactions assigned to the SVR_ADMIN category. For more information on ACActions, see Access Controls for Groups of Commands.

To delete User Groups:

1. Select **Delete** in the User Management section of the Access Administrator. This loads the Delete User Groups page with a list of all the defined users.
2. Click the Delete checkbox to mark the user group for deletion.
3. Select **Submit** to review your selections on the Preview Pending User Group Update page.
4. To accept the changes, press **Submit**. If the command is successful, you see a confirmation page. If the command was not successful, you see an error page explaining the failure. If you do not accept the changes, you may click **Back** to return to the Delete User Categories page.

Click on the fields in the following illustration for information.



Access Control Administration

Delete User Groups

Delete	Name	Filter	Users
☐	DSAdmin		bgorvine rsmith thopkins
☐	ROMDev		hmccoy kwagner
☐	Developers		ALUDev ChipDev ROMDev
☐	ALUDev		rdrake ssummers wworth
☐	ChipDev		ljones rsmith
✓	CadenceAdmin	sync:///Modules/cadence/Libqu	thopkins

Delete

Click this checkbox to delete this user group. Unclick the checkbox to leave this user group unmodified.

Group Name

Name of the user group.

Filter

An identifier that corresponds to a project, group of projects, category, module, or group of modules.

Users

Lists the users associated with the user group.

Special User Groups

DesignSync provides some predefined virtual user groups for use. These groups are dynamically maintained when AC Admin is reset. These groups can be used when defining the access to categories on objects, like other user groups, but they can not be modified, renamed, or removed.

You should never manually add or remove users from these groups. They are dynamically maintained. If you want to exclude an included user from permissions assigned to one of these groups, create a new access rule with a higher priority that uses the desired permissions.

Note: You can hide special user groups by using tuning the `acaCreateDynamicGroups` parameter. For more information on tuning ACAdmin parameters, see Setting Tunable Parameters in the acaConfigCustom.tcl File.

The predefined virtual user groups are:

- All-Module-Owners - a list of all the module owners' usernames. You can use this group to create a set of minimal permissions for all module owners, who are usually the module administrators.
- All-Project-Owners - a list of all the project owners' usernames. You can use this group to create a set of minimal permissions for all project owners, who are usually the project administrators.
- All-Server-Users - a list of all the user accounts on the server. You can use this group to create a set of minimal permissions for all defined users on a server.

These groups are automatically generated when ACAdmin is reset. If you have made changes that affect these groups, such as adding or removing user accounts on the server, you should perform an ACAdmin Reset.

Command Category Management

Category Management

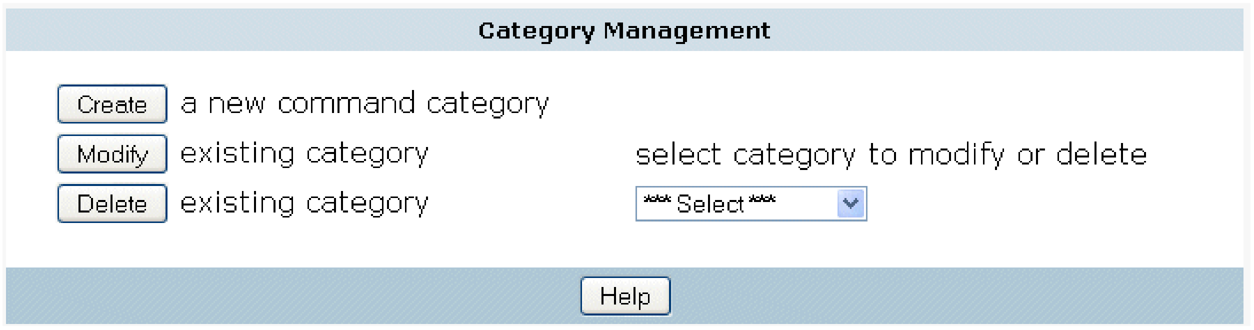
This section controls command category management. Access Control commands are grouped into command categories. Usually only the DesignSync Administrators are granted permission to create or modify command category definitions.

Once the categories are defined, the Project Managers use the Object AccessControl Management screens to assign permissions to users or groups by category.

AcaProjCatPrmDef

Changes to the command categories affect access control settings for all objects hosted by the server.

Click on the fields in the following illustration for information.



Create

Select this option to create a new command category.

Modify

Select a command category using the Select drop-down box, and click the **Modify** button to edit the command category. If you do not specify a command, the Modify page displays all command categories.

Delete

Select a command category using the Select drop-down box, and click the **Delete** button to delete the command category. If you do not specify a command, the Delete page displays all command categories.

Select

Contains a drop-down list of all existing command categories.

Note: Use the Select drop-down to select a command category definition to modify or delete.

Create Command Category

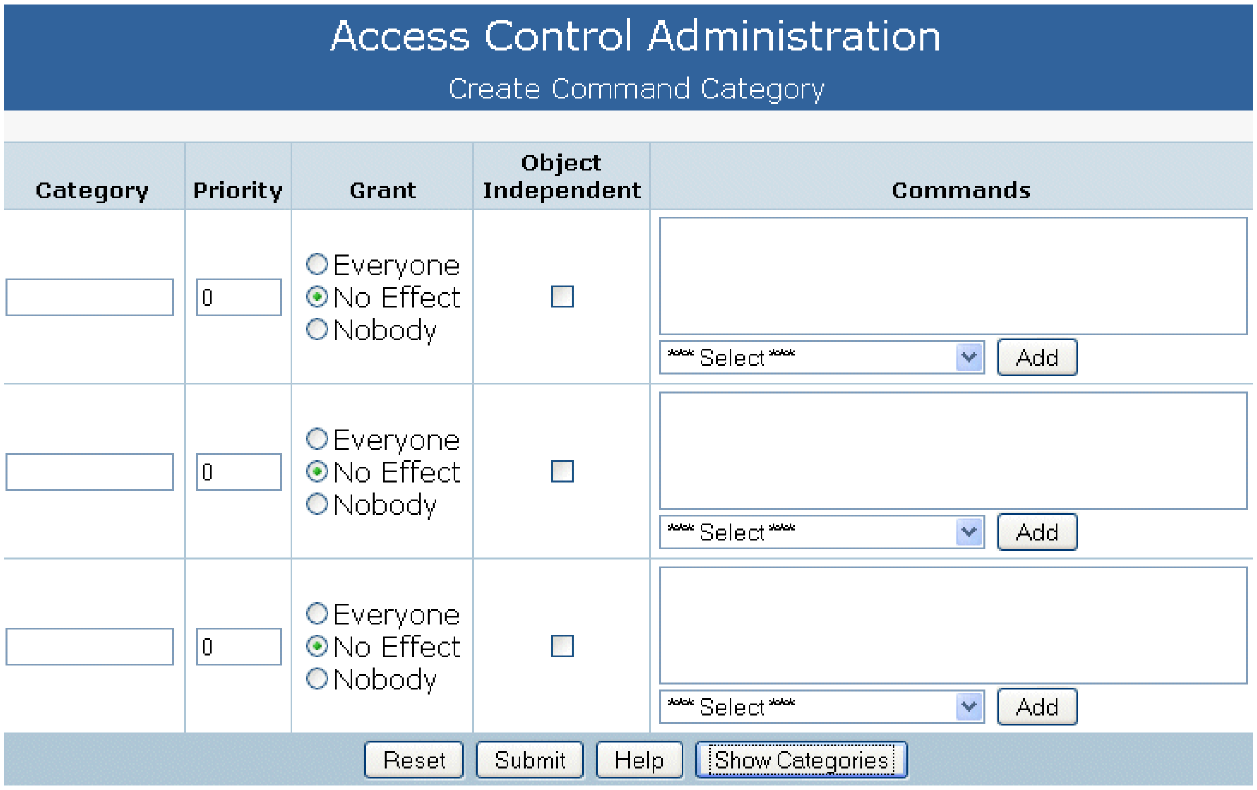
This page allows you to create command categories. Grouping the rights into a category allows you to grant permissions on a per-function basis.

Note: Usually only the DesignSync administrators should have permission to create or modify command category definitions. The access control that determines whether a user can create or modify category definitions is `AcaProjCatPrmDef`, This is included in the command group ACAactions assigned to the SVR_ADMIN category. For more information on ACActions, see Access Controls for Groups of Commands.

To create Command Categories:

1. Select **Create** in the Category Management section of the Access Administrator.
2. Name the command category and select the desired options on the Create Command Category page.
3. Select **Submit** to accept the options. If the command is successful, you see a success page showing that the category was created. If the command was not successful, you see an error page explaining the failure. The success page also contains a link to the Modify Command Category which allows you to view all existing categories and make any desired modifications.

Click on the fields in the following illustration for information.


Access Control Administration

Create Command Category

Category	Priority	Grant	Object Independent	Commands
	0	Everyone [✓ No Effect] Nobody	☐	*** Select *** Add
	0	Everyone [✓ No Effect] Nobody	☐	*** Select *** Add
	0	Everyone [✓ No Effect] Nobody	☐	*** Select *** Add

Reset Submit Help Show Categories

Category

Name of the command category. This name should correspond to DesignSync naming conventions. For a list of reserved characters that should not be used in the command category name, see the *ENOVIA Synchronicity DesignSync Data Manager User's Guide*: URL Syntax.

Priority

Priority of the command category within the processing order. The higher the priority, the greater the priority given to the command category.

Note: Access restrictions need to be positioned with high priority in order to be enforced.

Grant

Permissions grant for a category on a specific object is one of five types:

- **Everybody** - Grants permission for all defined users to access the specified command.
- **No Effect** - Does not create an access-control for the specified command, but allows the server to dictate the access control.
- **Nobody** - Denies permissions for all defined users to access the specified command. If certain users or groups are permitted to access the command, that access priority must be higher than this one.

Object Independent

Indicates whether the group of actions is server-specific or object-dependent.

- Object dependent - These actions apply to specific objects on the server, for example, a project or module. (Default) To specify object-dependant, do not check the **Object Independent** box.
- Object independent - These actions are server-wide, not applicable to specific objects, for example, user creation. To specify object-independent, check the **Object Independent** box.

Commands

Lists the access controls associated with the category. You may either type the command name or select the command from the drop-down list and press **Add** to add the command to the list. To remove a command you added, highlight the command and delete it.

Existing Command Categories

This list displays the existing command categories, including the default category definitions created when Access Administrator is activated, and the category properties.

Select **Dismiss** to close the window.

Modify Command Category

This page allows you to modify the existing command categories. You may re-prioritize, change the grant assignment or change the list of commands in the category.

By default, the list displayed is the existing command categories. Or, to limit the list, you can select the command category to modify.

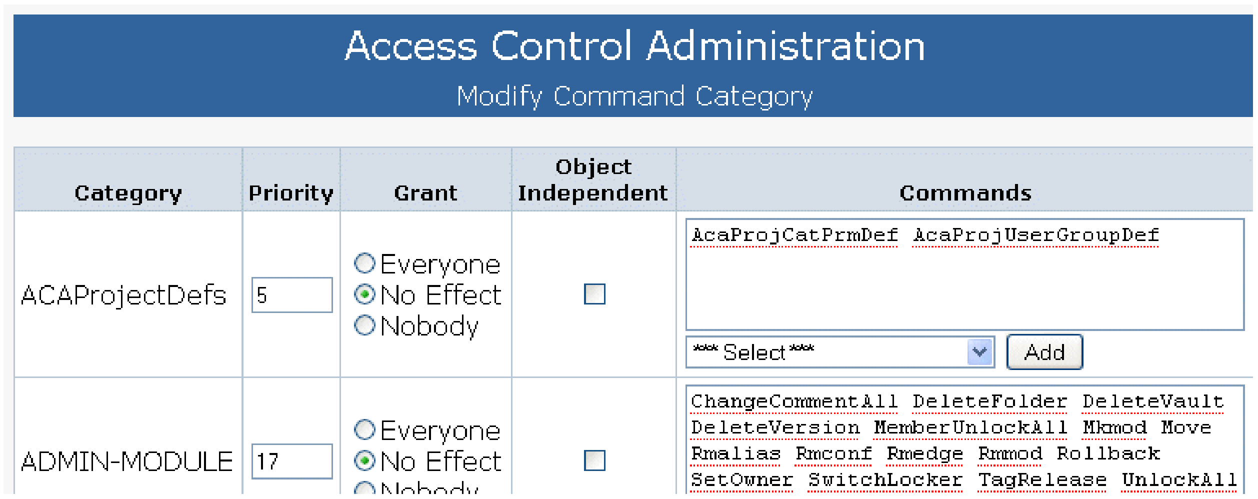
Note: Usually only the DesignSync administrators should have permission to create or modify command category definitions. The access control that determines whether a user can create or modify category definitions is `AcaProjCatPrmDef`, This is included in the command group ACAactions assigned to the SVR_ADMIN category. For more information on ACActions, see Access Controls for Groups of Commands.

To modify Command Categories:

1. Select **Modify** in the Category Management section of the Access Administrator. If you have selected a specific category, the Modify Command Category page displays only that category. If you have not selected a category, the page displays all the defined categories
2. Select the desired options on the Modify Command Category page.
3. Select **Submit** to accept the options. DesignSync allows you to review your selections on the Preview Pending Command Category Update page.
4. To accept the changes, press **Submit**. If the command is successful, you see a confirmation page. If the command was not successful, you see an error page explaining the failure. If you do not want to accept the changes, you may click **Reset** to cancel the changes.

Click on the fields in the following illustration for information.

Note: The page displays all defined categories. The image below is a small sampling, and not intended to show the complete page.



Access Control Administration

Modify Command Category

Category	Priority	Grant	Object Independent	Commands
ACAProjectDefs	5	Everyone [✓ No Effect] Nobody	☐	AcaProjCatPrmDef AcaProjUserGroupDef *** Select *** Add
ADMIN-MODULE	17	Everyone [✓ No Effect] Nobody	☐	ChangeCommentAll DeleteFolder DeleteVault DeleteVersion MemberUnlockAll Mkmod Move Rmalias Rmconf Rmedge Rmmod Rollback SetOwner SwitchLocker TagRelease UnlockAll

Category

Name of the command category. This name should correspond to DesignSync naming conventions. For a list of reserved characters that should not be used in the command category name, see the *ENOVIA Synchronicity DesignSync Data Manager User's Guide*: URL Syntax.

Priority

Priority of the command category within the processing order. The higher the number, the greater the priority given to the command category.

Note: Access restrictions need to be positioned with high priority in order to be enforced.

Grant

Permissions granted for a category on a specific object is one of the following types:

- **Everybody** - Grants permission for all defined users to access the specified command.
- **No Effect** - Does not create an access-control for the specified command, but allows the server to dictate the access control.
- **Nobody** - Denies permissions for all defined users to access the specified command. If certain users or groups are permitted to access the command, that access priority must be higher than this one.

Object Independent

Indicates whether the group of actions is server-specific or object-dependent.

- Object dependent - These actions apply to specific objects on the server, for example, a project or module. (Default) To specify object-dependant, do not check the **Object Independent** box.
- Object independent - These actions are server-wide, not applicable to specific objects, for example, user creation. To specify object-independent, check the **Object Independent** box.

Commands

Lists the access controls associated with the category. You may either type the command name or select the command from the drop-down list and press **Add** to add the command to list. To remove a command you added, highlight the command and delete it.

Delete Command Category

This page allows you to delete command categories. These categories are then removed from the system and any object definitions they were associated with.

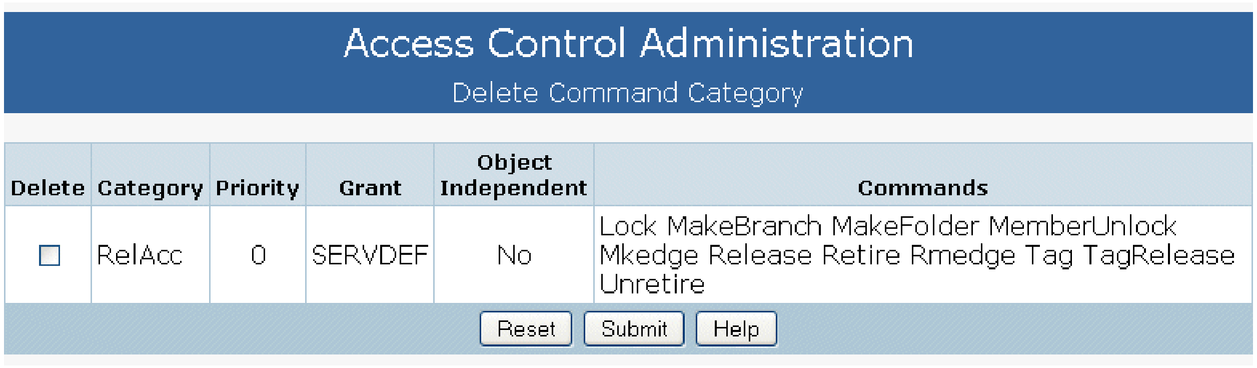
Note: Usually only the DesignSync administrators should have permission to delete command category definitions. The access control that determines whether a user can create or modify category definitions is `AcaProjCatPrmDef`, This is included in the command group ACAactions assigned to the SVR_ADMIN category. For more information on ACActions, see Access Controls for Groups of Commands.

To delete Command Categories:

1. Select **Delete** in the Category Management section of the Access Administrator. You can specific a command category to delete or, if you do not specify a command category, the Delete page displays the complete list of command categories.
2. Click the checkbox next to the command category to delete.
3. Select **Submit** to accept the options. DesignSync allows you to review your selections on the Preview Pending Command Category Update page.
4. To accept the changes, press **Submit**. If the command is successful, you see a confirmation page. If the command was not successful, you see an error page explaining the failure. If you do not accept the changes, you may click **Reset** to cancel the changes.

Click on the fields in the following illustration for information.

Note: The page displays all defined categories. The image below is a small sampling, and not intended to show the complete page.



Access Control Administration

Delete Command Category

Delete	Category	Priority	Grant	Object Independent	Commands
☐	RelAcc	0	SERVDEF	No	Lock MakeBranch MakeFolder MemberUnlock Mkedge Release Retire Rmedge Tag TagRelease Unretire

Delete

Click this checkbox to delete this command category. Unclick the checkbox to leave this command category unmodified.

Category

Name of the command category.

Priority

Priority of the command category within the processing order. The higher the priority, the greater the priority given to the command category.

Grant

Permissions granted for a category on a specific object is one of the following types:

- **Everybody** - Grants permission for all defined users to access the specified command.
- **No Effect** - Does not create an access-control for the specified command, but allows the server to dictate the access control.
- **Nobody** - Denies permissions for all defined users to access the specified command. If certain users or groups are permitted to access the command, that access priority must be higher than this one.

Object Independent

Indicates whether the group of actions is object-independent or object-dependent.

- Object dependent - These actions apply to specific objects on the server, for example, a project or module. (Default) To specify object-dependant, do not check the **Object Independent** box.

- Object independent - These actions are server-wide, not applicable to specific objects, for example, user creation. To specify object-independent, check the **Object Independent** box.

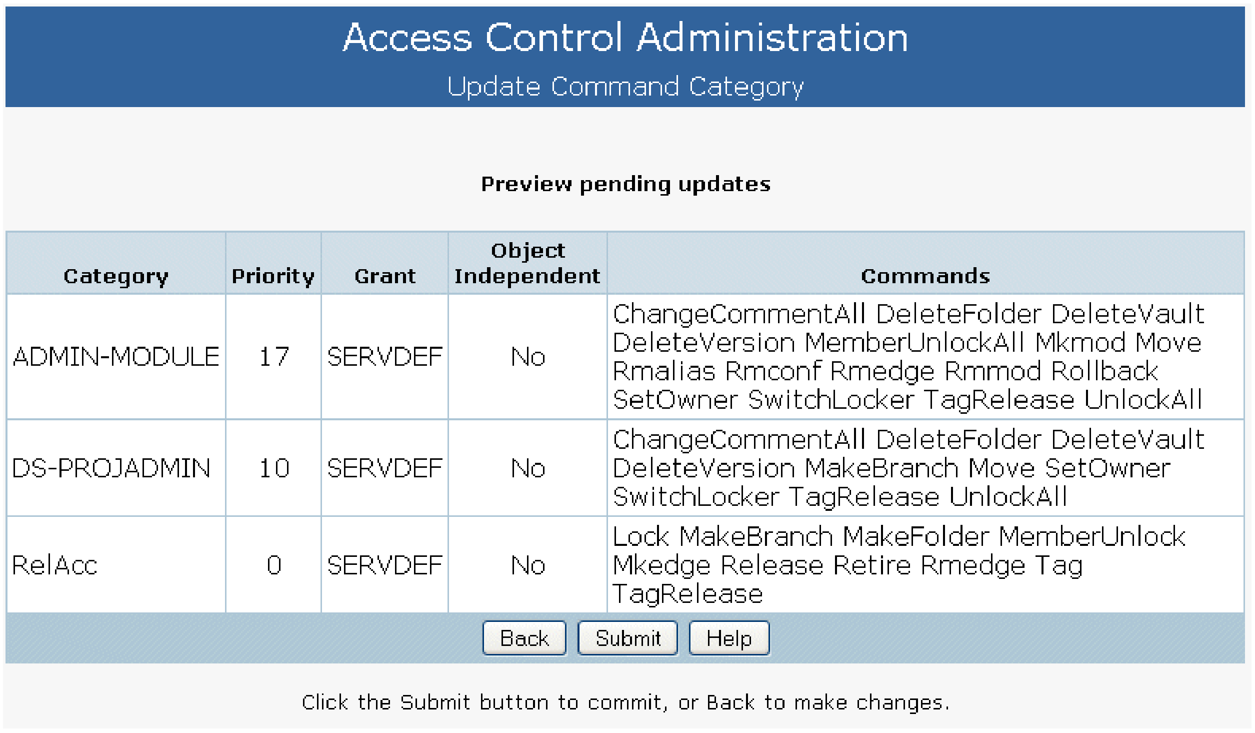
Commands

Lists the access controls associated with the category.

Preview Pending Command Category Updates

This page allows you to review the changes you've made to the command categories either by modifying or deleting a command category and confirm the changes before committing them to the server.

Accept the listed changes by pressing the Submit button, or modify the changes by pressing the back button to return to the Modify Command Category or Delete Command Category page.



Access Control Administration

Update Command Category

Preview pending updates

Category	Priority	Grant	Object Independent	Commands
ADMIN-MODULE	17	SERVDEF	No	ChangeCommentAll DeleteFolder DeleteVault DeleteVersion MemberUnlockAll Mkmod Move Rmalias Rmconf Rmedge Rmmod Rollback SetOwner SwitchLocker TagRelease UnlockAll
DS-PROJADMIN	10	SERVDEF	No	ChangeCommentAll DeleteFolder DeleteVault DeleteVersion MakeBranch Move SetOwner SwitchLocker TagRelease UnlockAll
RelAcc	0	SERVDEF	No	Lock MakeBranch MakeFolder MemberUnlock Mkedge Release Retire Rmedge Tag TagRelease

Click the Submit button to commit, or Back to make changes.

Custom Command and Filter Management

Custom Command Management

Using custom commands, you can create an access control command that includes an access control with all the desired optional modifiers such as:

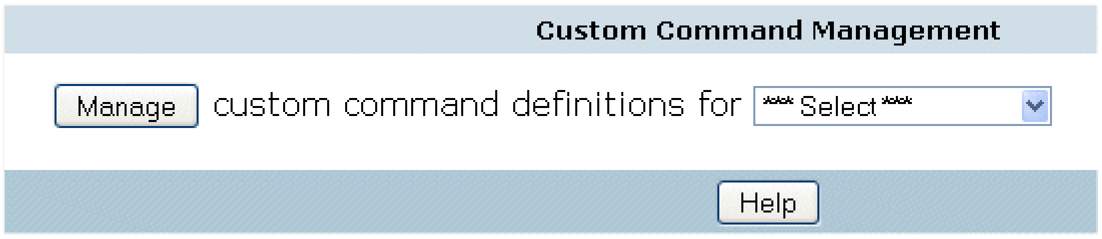
- a single filter file
- unlimited when clauses
- a single because clause

This allows a finer granularity of control over the command definition.

Custom command can be directly associated with access filters that grant or deny permissions.

Like all the access controls, when you create the custom command, you can make it available enterprise wide, site wide, or server wide. There is one definition file for each of these locations.

Select the location and then press **Manage** to edit the command definitions for the location.

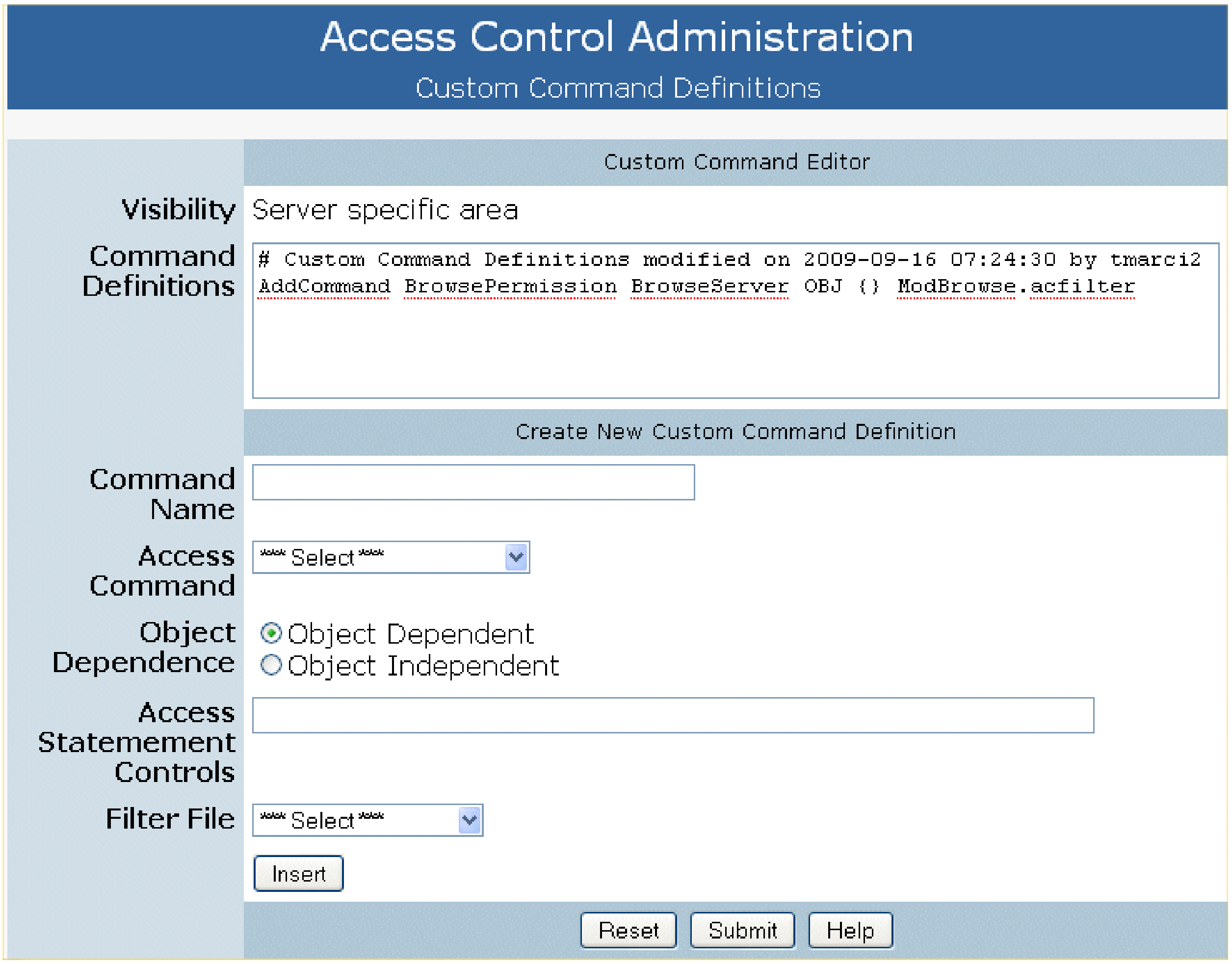


Custom Command Definitions

The custom command interface provides two methods for editing. You may directly type tcl-formated code to call the command(s) in the **Command Definitions** text box, or you may fill in the sections below the **Command Definitions** text box and press the **Insert** button to have the command created for you and inserted at the end of the text box.

When you have completed your modifications, press **Submit** to recreate the custom command definitions file for the location. If you do not wish to save your changes, press **Reset** to refresh the command definitions file from the server.

Click on the fields in the following illustration for information.


Access Control Administration
Custom Command Definitions
Custom Command Editor
Create New Custom Command Definition
Insert
Reset
Submit
Help

Visibility	Server specific area
Command Definitions	# Custom Command Definitions modified on 2009-09-16 07:24:30 by tmarci2 AddCommand BrowsePermission BrowseServer OBJ {} ModBrowse.acfilter
Command Name	
Access Command	*** Select ***
Object Dependence	[✓ Object Dependent] Object Independent
Access Statemement Controls	
Filter File	*** Select ***

Visibility

Shows the location of the custom command definitions. The commands can be enterprise wide, site-wide, or server specific.

Command Definitions

Displays the contents of the custom command definitions for the location. You can manually edit the contents of the Command Definitions text box; adding, removing, or modifying the commands as required.

To add a comment to the command definitions file, prefix the line with the hash (#) symbol.

Command Name

Enter the name of the command to automatically generate.

Access Command

Select the access control associated with the custom command from the drop-down list.

Object Dependence

Indicates whether the group of actions is object-independent or object-dependent.

- Object dependent - These actions apply to specific objects on the server, for example, a project or module. (Default) To specify object-dependant, select the **Object Dependent** radio button.
- Object independent - These actions are server-wide, not applicable to specific objects, for example, user creation. To specify object-independent, select the **Object Independent** radio button.

Access Statement Controls

The parameter qualifiers for the access control statements. These parameters limit the scope of the command (`when` clauses) or provide an explanatory message for users when a DesignSync access fails (`-because` clauses) For more information on parameter qualifiers, see Using Access Command Qualifiers.

Filter File

File containing the Custom Command Filter Definition.

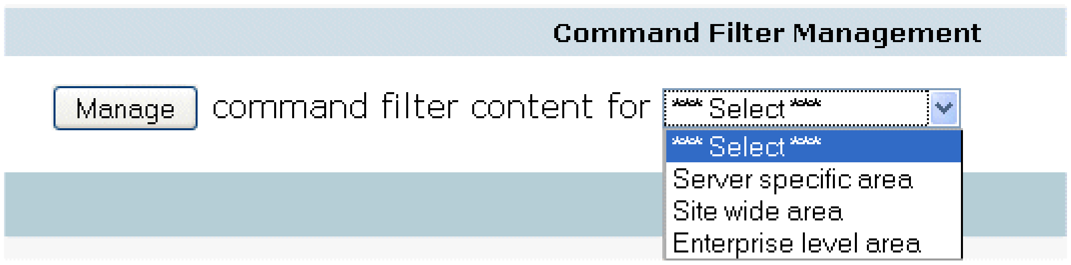
Command Filter Management

Custom command filters allow for the formation of complex decision making algorithms to determine access policies for a given operation. A custom command filter is a tcl procedure associated with a custom command.

Note: A custom command can only be associated with one filter.

Like all the access controls, when you create the custom command, you can make it available enterprise wide, site wide, or server wide. There is one definition file for each of these locations.

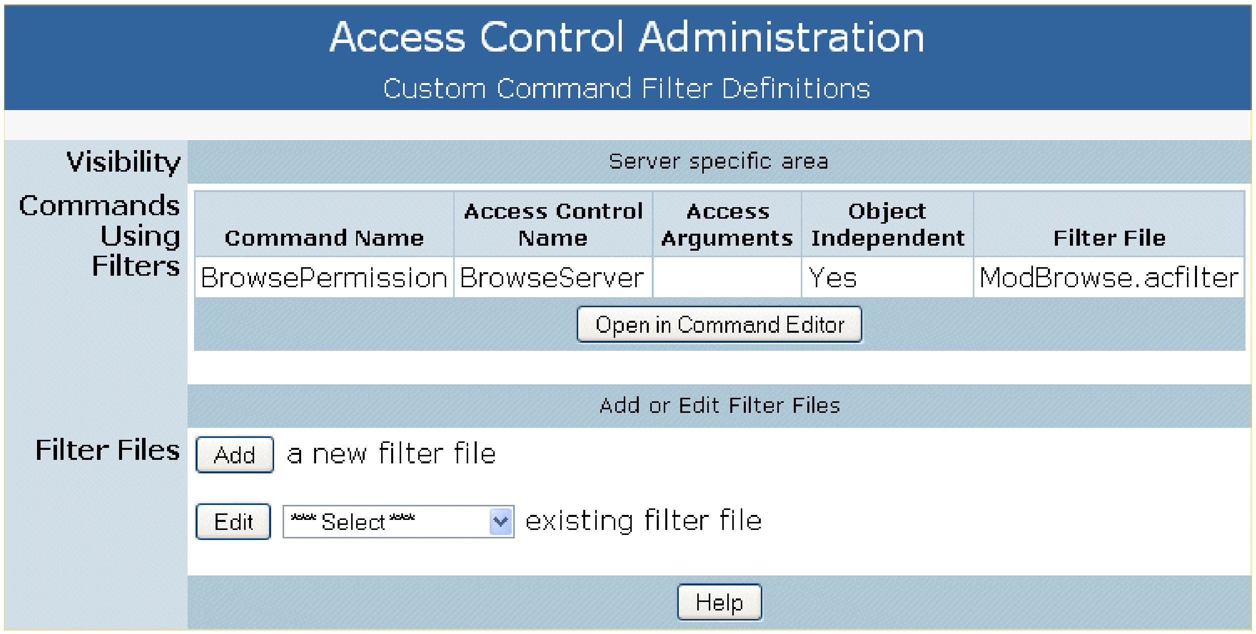
Select the location and then press **Manage** to edit the command definitions for the location.



Custom Command Filter Definitions

This page allows you to specify which filter to modify. You can select an existing filter file or create a new one.

Click on the fields in the following illustration for information.



Access Control Administration

Custom Command Filter Definitions

Visibility: Server specific area

Commands Using Filters

Command Name	Access Control Name	Access Arguments	Object Independent	Filter File
BrowsePermission	BrowseServer		Yes	ModBrowse.acfilter

Visibility

Shows the location of the custom command definitions. The commands can be enterprise wide, site-wide, or server specific.

Commands Using Filters

Lists the custom command definitions using filters and the following properties:

- **Command name** - name of the defined custom command.
- **Access Control Name** - access control command contained in the defined custom command.
- **Access Arguments** - parameter qualifiers for the access control statements, including where and because clauses.
- **Object Independent** - indicates whether the command is object-independent or object-dependent.
    - Yes - command is object-independent.
    - No - command is object-dependent.
- **Filter file** - Name of the filter file associated with the defined custom command.

Filter Files

Select **Add** to create a new filter file, or select the desired file to edit and press **Edit**.

Open in Command Editor

Press this button to open the Custom Command Definition page which allows you to edit the custom commands for that location.

Filter Files: Add

Creates a new filter file.

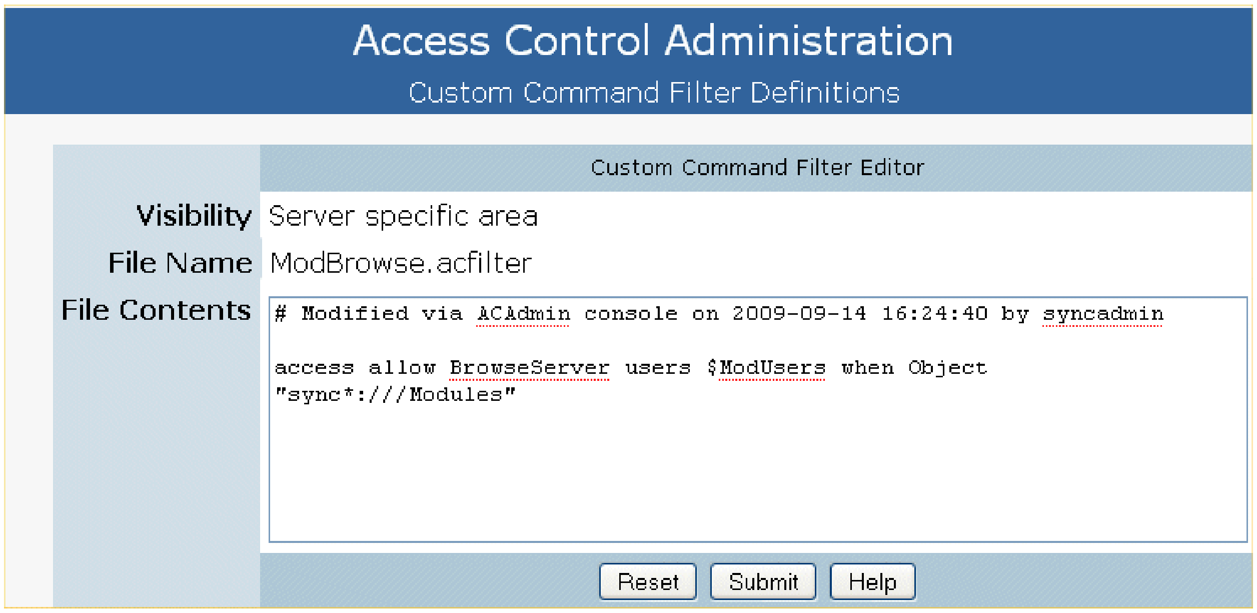
Filter Files: Edit

Select a defined filter file from the drop-down list and press **Edit** to edit an existing filter file.

Edit Custom Command Filter Definitions

This page allows you to create and edit custom command filter definitions for use with custom commands.

Click on the fields in the following illustration for information.


Access Control Administration
Custom Command Filter Definitions
Custom Command Filter Editor
Reset
Submit
Help

Visibility	Server specific area
File Name	ModBrowse.acfilter
File Contents	

```
# Modified via ACAdmin console on 2009-09-14 16:24:40 by syncadmin

access allow BrowseServer users $ModUsers when Object
"sync*:///Modules"
```

Visibility

Shows the location of the custom command definitions. The commands can be enterprise wide, site-wide, or server specific.

File Name

For new files, allows you to type in the file name. For existing files, displays the name of the file being modified.

File Contents

Displays the contents of the custom command filter. You can manually edit the contents of the file contents text box; adding ,removing, or modifying the filters as required.

To add a comment to the filter definition, prefix the line with the hash (#) symbol.

ACAdmin Reset

When you Submit your changes to the Access Administrator, most of them effective immediately. The only changes that require a reset are changes to dynamic user groups, for example if you change the users assigned to a group, or the roles of a user group on the server.

You can reset the server by selecting the **ACAdmin Reset** command from the ProjectSync menu. The command updates the AccessControl file and reloads the access controls so that DesignSync users can use them.

Customizing ACAdmin

Customizing ACAdmin

In ACAdmin you can change certain aspects of the default behavior by providing custom settings and custom implementation for some functions. DesignSync provides custom files for modifying the default behavior.

Note: These files are stored in the custom file areas as described in Setting Up Access Controls. Unless noted otherwise this means that the server custom path is searched first, then the site custom area, and finally we look into the enterprise custom directory. The first found file is sourced and no further attempt is made to continue searching. Other files are, where noted, sourced in the reverse order and accumulated.

The custom files are listed in the following table:

Click on the file name in the following table for the full description of the file and its usage.

File Name	Location	Brief Description
`aca_xusers.def`	`share/config/`	Provides a list of external users.

acaCallbacks.tcl	share/tcl/	Provides custom versions of adaptable func
aca_commands.def	share/config/	Provides custom command definitions. **Note:** Reversed search and accumulate
acaConfigCustom.tcl	share/tcl/	Provides custom definitions for a numbe tunable parameters. **Note:** Reversed search and accumula
AccessControl.acaACFilterExtension	share/AC_Components/	A custom filter file.
AccessControl.acaACFilterExtensionCustom	share/AC_Components/	A custom filter file, like AccessControl.acaACFilterExte that is reserved searched and accumulated. **Note:** Reversed search and accumulated.

Defining External Users using the aca_xusers.def File

This file is used to define a list of external users, people who do not have an account on the ProjectSync server.

In the `aca_xusers.def` file, external users should be added to the list variable called `projgroup`. This can be done using either of these two formats:

```
set projgroup {user1 user2...}
```

or

set projgroup {}

```
lappend projgroup user1 user2...}

lappend projgroup user3 user4 user5 user7
```

...

These usernames are displayed in the ACAdmin graphical web interface along with the defined groups and existing users, so they can be added to groups or individually referenced in permissions.

Note: There is only one instance of any given username. Even if the user is defined both internally and externally, the username will appear only once in the select list and all access accorded to that username will be available to that user.

Defining External Groups Using the aca_common_groups.def File

This file is used to define a list of external groups, groups of users who do not necessarily have an account on the ProjectSync server.

In the `aca_common_groups.def` file, external groups should be added to the list constant called `UserGroups`. The format for adding groups is:

```
set UserGroups(<GroupName>) {<UserName> [...]}
```

Note: User names are entered in a TCL list

These group names are displayed in the ACAdmin graphical web interface along with the defined groups and existing users, so they can be added to groups or individually referenced in permissions.

IMPORTANT: External group names cannot collide with existing group names. To avoiding naming collisions, it is highly recommended that you use a consistent naming convention for both internal and external names. One example of a naming convention would be:

- Internal users - username in all lower case (ie: rsmith)
- Internal groups - group name begins with capital letter, or each significant piece of the group names begins with a capital letter: (Admins, ModuleAdmins)
- External groups - being with a fixed prefix (SITE_Admins, SITE_ProjectLeaders)

This will prevent a collisions as well as being obvious visually to the user whether the user is seeing a group or user, and whether a group is internal to the ACadmin system on the server or available across servers.

Customizing Adaptable Functions using the acaCallbacks.tcl File

This file provides custom implementation for certain adaptable methods. These implementations override the default ACAdmin procedures.

The `acaCallbacks.tcl` file may contain valid TCL proc definitions for any of the following procedures:

- Provide a custom filter list using cbGetGroupFilter
- Provide list of external users using cbGetExtUsers
- Provide list of external users groups using cbGetExtUserGroupscbGetExtUserGroups

Provide a custom filter list using cbGetGroupFilter

The default method returns a list of all Projects and Modules defined on the server. These values are used by ACAdmin to build a pick list for User Group filters.

You can customize the proc to return an arbitrary Tcl formatted list of strings.

Important: Do not use reserved characters in your string list. For a list of reserved characters that should not be used, see the ENOVIA Synchronicity DesignSync Data Manager User's Guide: URL Syntax.

To modify the proc, add your code after the following line in the file:

```
proc cbGetGroupFilter {}{

  # return a TCL list

}
```

Provide list of external users using cbGetExtUsers

This method is used to provide a list of external users, people who do not have accounts on the ProjectSync server. You can either modify this method, or use the custom file aca_xusers.def to provide a fixed list of external users. For more information on the aca_xusers.def file, see Defining External Users using the aca_xusers.def File. To provide a list external user groups, see cbGetExtUserGroups.

Note: If your situation requires something more complex than a fixed list of external users make sure that your custom method returns a valid TCL list. If your situation only requires a fixed list of external users, use the aca_xusers.def File.

You can customize the proc to return an Tcl formatted list of strings.

To modify the proc, add your code after the following line in the file:

```
proc cbGetExtUsers {}{

  # return a TCL list

}
```

Note: There is only one instance of any given username. Even if the user is defined both internally and externally, the username will appear only once in the select list and all access accorded to that username will be available to that user.

Provide list of external users groups using cbGetExtUserGroups

The default method returns an empty list. Use the custom proc to define any external user groups using users who do not have accounts on the ProjectSync server. This should be used in conjunction with defined external users. You can define external users You can either modify this the cbGetExtUsers proc, or by using the custom file aca_xusers.def to provide a fixed list of external users.

Note: You can also create or modify user groups containing defined externals groups using the ACAdmin. For more information, see User Group Management.

IMPORTANT: External group names cannot collide with existing group names. To avoiding naming collisions, it is highly recommended that you use a consistent naming convention for both internal and external names. One example of a naming convention would be:

- Internal users - username in all lower case (ie: rsmith)
- Internal groups - group name begins with capital letter, or each significant piece of the group names begins with a capital letter: (Admins, ModuleAdmins)
- External groups - being with a fixed prefix (SITE_Admins, SITE_ProjectLeaders)

This will prevent a collisions as well as being obvious visually to the user whether the user is seeing a group or user, and whether a group is internal to the ACadmin system on the server or available across servers.

Adding External Groups

To modify the proc, add your code after the following line in the file:

```
proc cbGetExtUserGroups {host port}
```

Example of Adding External Groups with cbGetExtUserGroups

This example shows a simplified modification to cbGetExtUserGroups.

```
proc cbGetExtUserGroups {host port} {

   # could make use of host and port args and

   # generate different groups for different servers

   set extgrps(ExtAdmins) {name1 name2 }

   set extgrps(ExtDesigners) {name3 name4}

   set extgrps(ExtUsers) {name5 name6 name7}

 return [array get extgrps]

}
```

Defining Custom Access Control Commands with the aca_commands.def File

This file is used to provide a list of custom Access Control commands. Using the external file, instead of the ACAdmin web interface to create custom commands, allows you to provide and maintain the same set of custom commands for multiple servers on your site. For information on using the AcAdmin custom commands web interface, see Custom Command Management.

You can define an unlimited number of custom commands in the `aca_commands.def` file. As with the ACAdmin's AddCommand syntax, you define one command per line within the file.

```
AddCommand <Alias> <ACName> OBJ|NOOBJ {<extra-spec>} [filter-file]
```

Where:

- **Alias** is the name of your command. The name should be descriptive and easy to remember for ease of use.

  Important: Do not use reserved characters in your alias. For a list of reserved characters that should not be used, see the ENOVIA Synchronicity DesignSync Data Manager User's Guide: URL Syntax.

- **ACName** is a valid access name, such as AddNote, Checkin, BrowseServer, Checkout, etc.
- **OBJ or NOOBJ** is one of two values that indicates whether the command is object-dependant (OBJ) or object-independant (NOOBJ).
- **extra-spec** contains parameter qualifiers for the access control statements. These parameters limit the scope of the command (`when` clauses) or provide an explanatory message for users when a DesignSync access fails (`-because` clauses) For more information on parameter qualifiers, see Using Access Command Qualifiers. If no parameter qualifiers are needed, this string must be empty {}
- **filter-file** is an optional argument that allows you to specify the file containing the Custom Command Filter Definition.

Tip: To avoid potential command name collisions, DesigSync recommends that you choose a naming convention such as a common prefix, for example SITE for external commands. Thus if you had an command that grouped maintenance permissions (such as backup, restore, freezemod, etc), the alias might be SITE_Maintenance.

Example

This shows a simple example of a custom command.

```
AddCommand SITE_BrowsePermission BrowserServer OBJ {when user
Admin} ModBrowse.acfilter
```

Setting Tunable Parameters in the acaConfigCustom.tcl File

This file can be used to set any of the tunable parameters available in ACAdmin.

Common tunable parameters

The following table lists the most commonly used tunable parameters.

Variable Name	Definition	Default Value
`acaDefaultPermission`	Defines default permission for new Categories. By default, the server sets the default.	`SERVDEF`
`acaDisplayExternalGroups`	Determines whether to show or hide external groups. To hide external groups, set the parameter to 0. To show external groups, set the parameter to 1.	`1`
`acaShowGroups`	Determines whether to include all defined groups in the Users Select-box. To hide defined groups, set the parameter to 0. To show defined groups, set the parameter to 1.	`1`
`acaImportFilter`	Determines whether to incorporate filter code directly into ACAdmin's AccessControl file or use the `source` command in the ACadmin AccessControl file to source the filter code stored in a different file. Embedding filters may provide a better performance, while sourcing a separate file avoids any potential discrepancy if the embedded code and the actual file become out of sync. To use the source command in the ACAdmin AccessControl file to source the filter code, set the parameter to 0. To embed the filters into the ACAdmin AccessControl file, set the parameter to 1.	`0`
`acaDoBackup`	Determines whether to rename and save the current AccessControl file, or other *.def file before writing a new version of it. To disable backup file creation, set the parameter to 0. To enable backup file creation, set the parameter to 1.	`1`
`acaServerUsers`	Alias for dynamic User Group representing all server users.	`All-Server-Users`
`acaModuleOwners`	Alias for dynamic User Group representing all	`All-Module-`

	Module owners.	`Owners`
`acaProjectOwners`	Alias for dynamic User Group representing all Project owners.	`All-Project-Owners`
`acaCreateDynamicGroups`	Whether to create the three dynamic groups: • `acaServerUsers` • `acaModuleOwners` • `acaProjectOwners` Disabling this feature might be an advantage for some LDAP configurations, where, depending on the use model, only a subset of user records may exist on the database at any given time.	`1`

Internal parameters

These parameters are generally used only internally. You can modify them, but their usage is less common.

Variable Name	**Definition**	**Default Value**
`acaDenyOnNONE`	Determines whether to put `access deny` statements for OBJECT on Permission NONE. To put access deny statements an object when the permission is set to NONE, set the parameter to 1.	`1`
`acaBecauseExpandUserList`	Determines whether to expand the User Groups in a -because statement. To pass the UserGroup without expanding it in a -because statement, set the parameter to 0. To expand the user group to the members of the group in a -because statement, set the parameter to 1.	`0`
`acaRemovePermForDeletedGroups`	Determines whether to automatically remove permissions (if any) for deleted groups. To not remove any permissions set for removed groups, set the parameter to 0. To remove any permissions set for removed groups, set the parameter to 1. Note: If the parameter is set to 1, the `aca_perms.def` file is updated when the user group is deleted.	`1`

Command Related Settings

These parameters control command behavior.

Variable Name	Definition	Default Value
`acaSuppressObjWildcards`	List of Commands that do not need the wildcard extension (".../*") for Objects. The default is an empty list (pictured in the Default Value column).	`{}`
`acaEnforceObjWildcards`	List of Commands that do need an access statement with and without the wildcard extension (".../*") for Objects	`{BrowseServer AcaProjCatPrmDef AcaProjUserGroupDef}`

Custom AccessControl Extension Filters

In addition to the filters created via the ACAdmin web interface, DesignSync supports extra filters defined in the following optional custom files. These files can be sources by the AcAdmin:

- **AccessControl.acaACFilterExtension** - This is a single file that is located and sourced DesignSync. Once DesignSync has found the file, it does not continue looking for additional versions.
- **AccessControl.acaACFilterExtensionCustom** - DesignSync searches for and sources all instances of this file.

Related Topics

Custom Command Filter Definitions

Access Controls for DesignSync Commands

DesignSync Action Definitions

DesignSync Action Definitions

You set up access controls on particular DesignSync **actions**, or operations. To set up an access control on an operation, the operation must have an action definition specified with an `access define` command. If an action definition exists for an operation, you can control access to that operation using the stcl `access allow`, `access deny`, and `access filter` commands. You can also control access to operations on module data by using the `access decline` command.

DesignSync provides predefined actions corresponding to most operations you might want to access control. These actions are defined in the default access control file for DesignSync:

```
$SYNC_DIR/share/AC_Components/AccessControl.ds
```

See Introduction to Access Control for details on the individual files used to define access controls.

Important: Do not edit the `$SYNC_DIR/share/AC_Components/AccessControl.ds` file; changes will be lost upon upgrading. Instead, edit your site or server custom `AccessControl` file (see Setting Up Access Controls).

DesignSync actions governed by access controls:

To control access to this DesignSync operation...	**Customize this DesignSync access control...**
`cancel`	`Unlock` See Access Controls for Unlocking for details.
caching disable	`Caching` See Access Controls For Object Caching for details.
caching enable	`Caching` See Access Controls For Object Caching for details.
`ci`	`Checkin`

	See Access Controls for Checking In for details.
A `ci` that creates a new branch (`ci -new` of an unmanaged object, or auto-branching)	`Checkin` and `MakeBranch` See Access Controls for Checking In and Access Controls for Creating Branches for details.
A `co` that creates a new branch (`co -lock` while auto-branching)	`Checkout` and `MakeBranch` See Access Controls for Checking Out and Access Controls for Creating Branches for details.
`co -[share|mirror]` when the requested object is already in the cache or mirror directory, or `co -reference` (without also specifying the `-lock` option)	`BrowseServer` See Access Controls for Browsing the Server for details. (`Checkout` access is not required.)
`compare` **Note**: The `compare` command reads and obeys the access controls set at the directory level. Access controls applies at the object level may not be obeyed.	`BrowseServer` See Access Controls for Browsing the Server for details.
`contents` **Note**: The `contents` command reads and obeys the access controls set at the directory level. Access controls applies at the object level may not be obeyed.	`BrowseServer` See Access Controls for Browsing the Server for details.
`datasheet`	`BrowseServer` See Access Controls for Browsing the Server for details.
`ls`	`BrowseServer` See Access Controls for Browsing the Server for details.
`mkbranch`	`MakeBranch` See Access Controls for Creating

	Branches for details.
`mkfolder`	`MakeFolder` See Access Controls for Creating Folders for details.
`mvfile` without specifying the `–allconfigs` option	`Retire` and `Checkin` See Access Controls for Retiring and Access Controls for Checking In for details.
`mvfile -allconfigs`	`Move` See Access Controls for Moving a File for details.
`mvfolder`	`MakeFolder` and `Delete when Type FOLDER` See Access Controls for Creating Folders and Access Controls for Deleting for details.
`populate`	`Checkout` See Access Controls for Checking Out for details.
A `populate` that creates a new branch (`populate -lock` while auto-branching)	`Checkout` and `MakeBranch` See Access Controls for Checking Out and Access Controls for Creating Branches for details.
`populate -share` when the requested object is already in the cache or `populate -reference` (without also specifying the `-lock` option) Note: `populate -mirror` does not check access controls	`BrowseServer` See Access Controls for Browsing the Server for details. (`Checkout` access is not required.)
`purge`	`Delete when Type VERSION` See Access Controls for Deleting for details.
`retire` (without specifying the `–unretire` option)	`Retire` See Access Controls for Retiring for

	details.
`retire -unretire`	`Unretire` See Access Controls for Unretiring for details.
`rmfolder`	`Delete when Type FOLDER` See Access Controls for Deleting for details.
`rmvault`	`Delete when Type VAULT` See Access Controls for Deleting for details.
`rmversion`	`Delete when Type VERSION` See Access Controls for Deleting for details.
setowner	SetOwner See Access Controls for Setting Owners for details.
`setvault`	`BrowseServer` See Access Controls for Browsing the Server for details.
`showlocks`	`BrowseServer` See Access Controls for Browsing the Server for details.
`switchlocker`	`SwitchLocker` See Access Controls for Changing a Lock Owner for details.
`tag`	`Tag` See Access Controls for Tagging for details.
`unlock`	`Unlock` See Access Controls for Unlocking for details.
`url setprop` of a version's `log` property; which is the comment associated with the version.	`ChangeComment` See Access Controls for Changing a Checkin Comment

`vhistory`	`BrowseServer` See Access Controls for Browsing the Server for details.

You can create access rules for predefined DesignSync actions. You also can create access rules for multiple actions that are grouped into a set of commands. See Access Controls for Groups of Commands for details.

Although most DesignSync actions are predefined in the default `AccessControl.ds` file, you might want to create your own action definitions for custom operations. Do not redefine the existing actions in your site and server `AccessControl` files. You will not be able to access the server if it detects duplicate `access define` statements. You can, however, define new actions within your site or server custom `AccessControl` files.

See Access Control Scripting for an example of a custom action definition.

Related Topics

Module Action Definitions

ProjectSync Action Definitions

Setting Up Access Controls

User Authentication Access Controls

User Authentication Action Definitions

Access Controls for Browsing the Server

The , `BrowseServer` access control access to browsing both the DesignSync and ProjectSync server.

Defining and Using the Access Control

The `BrowseServer` access control is located in the default DesignSync access control file:

```
$SYNC_DIR/share/AC_Components/AccessControl.ds
```

You can use the `BrowseServer` action to control user-initiated browse operations such as using `ls` on a server-side object, requesting a data sheet, viewing a data replication root, module view, or vault using the DesignSync graphical interface. `BrowseServer` also controls other operations, such as checking out files, where the server must browse the vault.

If you use the `BrowseServer` access control to restrict browsing of an object rather than a folder, DesignSync List View and the `ls` command output still list the object. However, users will not be able to view details of the restricted object.

For ProjectSync, `BrowseServer` lets you control access to objects other than notes through the data sheet or browser pop-up window.

The `BrowseServer` action also restricts email notifications. If a user does not have permission to access a project, the user does not receive email when a note associated with that project is created or modified, even when the user is on the note's CC list.

If you restrict browsing, you probably want to use the `$DesignSyncActions` or `$AllActions` variables to restrict access to all operations. (See Access Controls for Groups of Commands for details on these variables.)

See Browsing Modules on a Server to learn how the `BrowseServer` access control applies to module data.

Note: Setting a `BrowseServer` access control at a directory level does not set permissions for browsing for paths **below** the directory. In order to control how an entire directory structure is browsed, you must specify the access control in the form "`...`*`<path>`*`/*`", as shown in the example below.

The access control definition is:

```
access define BrowseServer <Object>
```

Where `<Object>` is the path to the object on the server.

Examples using Access Control Files

The following access control rules ensure that only members of the ASIC team can browse the server containing the ASIC project:

```
access init {
   set ASICteam { chan lynch kapoor vega }
}

access deny BrowseServer everyone when Object
"sync:///Projects/ASIC/*" \
-because "To perform this operation, you must be a member of the
ASIC team."
```

```
access allow BrowseServer only users $ASICteam when Object
"sync:///Projects/ASIC/*"
```

The first rule closes the server to everyone, and the `-because` clause prints an error message when access fails.

The second rule builds on the first and opens the server to members of the ASIC team, `$ASICteam`, as specified by the `access init` definition. (See Using Access Commands for details on using `access init` to create custom variables.)

The following access control rule prevents users from browsing the `layout` subproject under the `ASIC` project:

```
access deny BrowseServer everyone when Object
"sync:///Projects/ASIC/layout/*"
```

To allow only a certain group of users (for example, layout team members) to browse the `layout` subproject, add the line:

```
access allow BrowseServer only users $layout_team_members when
Object "sync:///Projects/ASIC/layout/*"
```

See Setting Up Access Controls for information on creating your own custom `AccessControl` files. See Using Access Commands for a list of the access commands and examples of how to use them.

To allow browsing across the server, for example, to allow the users to view the data replication roots (DRR)s located on a MAS server, enter the server root in the custom access control that governs access to the server on which the MAS runs:

```
access allow BrowseServer admins when Object sync:///
```

Defining Enterprise Access Map

When access control is delegated, you can enable the `BrowseServer` access map. A sample version of the access map is provided, commented out, in the custom DesignSync access control file for the server, `$SYNC_CUSTOM_DIR/servers/<`*`serverName`*`>/<`*`portNumber`*`>/share/AccessControl`.

```
access map BrowseServer {

    lappend checkmasks [list read $Object]
    return [list "masks" $checkmasks]

}
```

Related Topics

Access Control Scripting

Sample Access Controls

Setting Up Access Controls

Using Access Commands

Browsing Modules on a Server

Access Controls for Checking Out

The default DesignSync access control file:

`$SYNC_DIR/share/AC_Components/AccessControl.ds`

defines two actions, `Checkout` and `CheckoutLock` to control check-outs from the DesignSync vault:

The CheckoutLock access control is invoked whenever a checkout operation is performed with a lock. If there are no CheckoutLock access controls granted, DesignSync uses the Checkout access control settings, including the option Lock property to determine if the user is allowed to perform a checkout with a lock.

Using Checkout and CheckoutLock Access Controls

Checkout Access Control

CheckoutLock Access Control

Enterprise Access Design Map

Using Checkout and CheckoutLock Access Controls

Both the Checkout and CheckoutLock access controls are used to control DesignSync vault checkouts. Using the CheckoutLock access control allows the administrator to finely control the ability to modify vaults by locking an object on modification. This model is especially useful when you want to provide read access to DesignSync objects used by different development teams as part of their designs, but not actually intended to be modified by those teams. The administrator can define an access control for the referencing teams that provides Checkout access and a different access control for the owning team, who is responsible for the modifications to the objects, that includes the CheckoutLock ability and other vault modification access, such as tag, Checkin, etc.

Note: If there is a conflict in permissions, for example, if you have the following two access controls defined:

```
access allow Checkout everyone when Lock “yes”

access deny CheckoutLock everyone
```

DesignSync enforces the more restrictive access control and denies all users the ability to perform a Checkout with a lock.

Access controls on the `Checkout/CheckoutLock` actions also affect populate operations because populating a folder involves checking out files. Checking out with or without a lock, and therefore populating with or without a lock, can be restricted based on access control definitions. You cannot define different access controls for checkout and populate operations.

When users are denied the ability to check out files, they still can "check out" a file under certain circumstances:

- When they check out from a mirror. This operation is permitted because it only creates a link to an existing file in the mirror directory.
- When they check out a reference, which does not transfer the contents of the file from the server.

  Note, however, that users cannot check out locked references when they do not have `Checkout` or `CheckoutLock` access. See the ENOVIA Synchronicity Command Reference: co Command and populate Command for more information.

See Fetching a Module to learn how the Checkout access controls apply to module data.

Note: Pattern matching for Checkout and CheckoutLock is slightly different for module members. DesignSync vault objects and modules use a format like this:

`sync:///Projects/<`*`Path to Vault Objects`*`>/*`

`sync:///Modules/<`*`path to module`*`>/<`*`Module`*<<<<<<<="" local="" class="Mono">>;<*`version`*>

Module members use a format like this:

```
sync:///Modules/<path to module>/<Module>;*
```

Access to module members may also be controlled with module specific access controls, such as MemberCheckin; discussed in Creating a New Version of a Module; MemberCheckout, and MemberCheckoutLock, discussed in Fetching a Module.

Checkout Access Control

```
access define Checkout {<Object> <Lock> <Merge> <NewBranchName>
<Branch> <ViewName>}
```

Where:

- `<Object>` is the URL of the object being checked out. For example, `sync:///Projects/ASIC/layout/spec.doc;1.5`.
- `<Lock>` is `yes` for checkout with lock and `no` for checkout without lock (fetch).
- `<Merge>` is `yes` if a merge has been requested and is needed and `no` otherwise.

  `<Merge>` is `yes` only if a merge is needed. For example, suppose the Latest version of a file in the vault is 1.3 and a user has an unmodified local copy of version 1.2. If the user executes `co -merge -nocomment <filename>`, then `<Merge>` is `no` because no merge is attempted. If version 1.2 had been locally modified, then `<Merge>` would be `yes`, indicating a merge would be attempted.

- `<NewBranchName>` is the name of a new branch that is created upon branching checkout.

  Use the `<NewBranchName>` parameter to allow or deny branching checkouts on particular branches - where the branch is created upon checkout. For nonbranching checkouts, the `<NewBranchName>` parameter is the empty string ("").
- `<Branch>` is the URL of the branch the object is being checked out from.

  `<Branch>` is the URL of the branch where the object will be checked out from, when the operation is a non-branching checkout.

  Use the `<Branch>` parameter to allow or deny nonbranching check-outs on particular branches where the object already exists and is not autobranched.

  The `<Branch>` parameter is not sufficient to control checking out from particular branches because new or autobranched objects do not have a `<Branch>` URL until after they are checked in; thus, you need to control the `<NewBranchName>` parameter for new or autobranched objects.

- <*ViewName*> is the name of the single view, or a Tcl list of multiple views being used. Use this parameter to allow or deny view usage for module checkout. For more information on module views, see ENOVIA Synchronicity DesignSync: Understanding Module Views.

(See DesignSync User's Guide: Autobranching: Exploring "What-If" Scenarios for a description of autobranching.)

Example

The following access control rule prevents a particular set of users from checking out with a lock from the ASIC project:

```
access deny Checkout everyone when Lock "yes" \
  when Object "sync:///Projects/ASIC/*"
```

See Setting Up Access Controls for information on creating your own custom `AccessControl` files. See Using Access Commands for a list of the access commands and examples of how to use them.

The following access control rule prevents a particular set of users from checking out with a lock from any version of the CPU module:

```
access deny Checkout everyone when Lock "yes" \
 when Object "sync:///Modules/Components/CPU; \

           *"
```

CheckoutLock Access Control

```
access define CheckoutLock {<Object> <Merge> <NewBranchName>
<Branch> <ViewName>}
```

Where:

- <*Object*> is the URL of the object being checked out. For example, `sync:///Projects/ASIC/layout/spec.doc;1.5`.
- <*Merge*> is `yes` if a merge has been requested and is needed and `no` otherwise.

  <*Merge*> is `yes` only if a merge is needed. For example, suppose the Latest version of a file in the vault is 1.3 and a user has an unmodified local copy of version 1.2. If the user executes `co -merge -nocomment <filename>`, then `<Merge>` is `no` because no merge is attempted. If version 1.2 had been locally modified, then `<Merge>` would be `yes`, indicating a merge would be attempted.

- *<NewBranchName>* is the name of a new branch that is created upon branching checkout.

  Use the `<NewBranchName>` parameter to allow or deny branching checkouts on particular branches - where the branch is created upon checkout. For nonbranching checkouts, the `<NewBranchName>` parameter is the empty string (`""`).
- *<Branch>* is the URL of the branch the object is being checked out from.

  <Branch> is the URL of the branch where the object will be checked out from, when the operation is a non-branching checkout.

  Use the *<Branch>* parameter to allow or deny nonbranching check-outs on particular branches where the object already exists and is not autobranched.

  The *<Branch>* parameter is not sufficient to control checking out from particular branches because new or autobranched objects do not have a `<Branch>` URL until after they are checked in; thus, you need to control the `<NewBranchName>` parameter for new or autobranched objects.
- <*ViewName*> is the name of the single view, or a Tcl list of multiple views being used. Use this parameter to allow or deny view usage for module checkout. For more information on module views, see ENOVIA Synchronicity DesignSync: Understanding Module Views.

(See DesignSync User's Guide: Autobranching: Exploring "What-If" Scenarios for a description of autobranching.)

Example

The following access control rule prevents users from checking out with a lock from the ASIC project:

```
access deny CheckoutLock ASICDevGroup \
  when Object "sync:///Projects/ASIC/*"
```

See Setting Up Access Controls for information on creating your own custom `AccessControl` files. See Using Access Commands for a list of the access commands and examples of how to use them.

Enterprise Design Access Map

When Enterprise Design access map is enabled for the server, you can enable the Checkout and CheckoutLock access maps. Sample versions of the access maps are provided, commented out, in the custom DesignSync access control file for the server,

`$SYNC_CUSTOM_DIR/servers/<serverName>/<portNumber>/share/AccessControl.`

Note: This example shows CheckoutLock being access control verified, locally and on the Enterprise Server.

```
access map Checkout {

    # verify checkout

    lappend checkmasks [list read $Object]

    return [list "masks" $checkmasks]

}

access map CheckoutLock {

    # verify checkout and lock

    lappend checkmasks [list read $Object]

    return [list "masks" $checkmasks "localcheck" "yes"]
```

Related Topics

Access Control Scripting

Sample Access Controls

Setting Up Access Controls

Using Access Commands

Fetching a Module

Access Controls for Changing a Lock Owner

The default DesignSync access control file:

`$SYNC_DIR/share/AC_Components/AccessControl.ds`

defines a single action, `SwitchLocker`, to control changing the owner of a locked object:

```
access define SwitchLocker <Object> <CurrentLockOwner>
<NewLockOwner>
```

Where:

- `<Object>` is the URL of the branch (for example, `sync:///Projects/ASIC/layout/spec.doc;1`).
- `<CurrentLockOwner>` is the user name of the person who currently owns the lock.
- `<NewLockOwner>` is the user name of the person who should get the new lock.

Because changing lock ownership is an unusual and potentially dangerous action, the default access control for `SwitchLocker` is to deny access to everyone.

See Changing a Module's Lock Owner to learn how the `SwitchLocker` access control applies to module data.

Example

The following access control rule permits the Admin to change the locks held by Jackie, who recently left the company and forgot to unlock a number of files:

```
access init {
   set admin { chan kapoor }
}

access deny SwitchLocker everyone

access allow SwitchLocker
  when CurrentLockOwner jackie \
  when NewLockOwner $admin \
  when Object "sync:///Projects/ASIC/*"
```

The first rule prevents everyone from switching lock owners.

The second rule builds on the first and let administrators change lock owners. The `$admin` variable is specified by the `access init` definition. (See Using Access Commands for details on using `access init` to create custom variables.)

See Setting Up Access Controls for information on creating your own custom `AccessControl` files. See Using Access Commands for a list of the access commands and examples of how to use them.

Related Topics

Access Control Scripting

Sample Access Controls

Setting Up Access Controls

Changing a Module's Lock Owner

Access Controls for Checking In

DesignSync Checkin access is controlled by the Checkin access control. Because hierarchical references changes also require a checkin, the checkin access control can be called by the Checkin command or any of commands that create, modify, or remove hierarchical references.

DesignSync Access Controls.

Enterprise Design Access Map.

DesignSync Access Controls

The default DesignSync access control file:

`$SYNC_DIR/share/AC_Components/AccessControl.ds`

defines the action `Checkin`, to control check-ins to a DesignSync vault:

`access define Checkin <`*`Object`*`> <`*`Comment`*`> <`*`CommentLen`*`> <`*`Branch`*`> <`*`NewBranchName`*`> <`*`Action`*`>`

Where:

- `<`*`Object`*`>` is the URL of the object being checked in. For example: `sync:///Projects/ASIC/layout/spec.doc;1.5`.
- `<`*`Comment`*`>` is the check-in comment string.
- `<`*`CommentLen`*`>` is the comment length (for example, "20").
  Note: The value of CommentLen is a per byte character value. Comments entered in a language that uses multibyte characters may be visually shorter than the defined minimum comment length, but still meet the minimum comment length. (for example, this Korean phrase "이것은 코멘트입니다" visually appears as 10 characters, but is understood by the system as 28 characters long.)
- `<`*`Branch`*`>` is the URL of the branch where the object will be checked in.

  Use the `<Branch>` parameter to allow or deny nonbranching check-ins on particular branches where the object already exists and is not autobranched.

The *<Branch>* parameter is not sufficient to control checking in on particular branches because new or autobranched objects do not have a `<Branch>` URL until after they are checked in; thus, you need to control the `<NewBranchName>` parameter for new or autobranched objects.

- *<NewBranchName>* is the name of the new branch.

  Use the `<NewBranchName>` parameter to allow or deny branching check-ins on particular branches where the object is new or the object is to be autobranched upon check-in. For nonbranching check-ins, the `<NewBranchName>` parameter is the empty string (`""`).

- <*Action*> the origin of the revison control command that initiated the need to check an access control. The possible values are:
  - Addhref - Access Controls for addhref
  - Edithref - access for the edithref operations is provided for individual operations within the edithref command. The edithref command adds and removes the hrefs, and are subject to those access controls.
  - Rmhref - Access Controls for rmhref
  - Checkin

(See DesignSync User's Guide: Autobranching: Exploring "What-If" Scenarios for a description of autobranching.)

Notes:

- Moving or renaming a managed object (see the `mvfile` command) requires `Checkin` permission in order to create the vault for the moved or renamed object.
- Tagging a file during the checkin operation (`ci -tag`), requires ADD permission for the tag command.

See Creating a New Version of a Module to learn how the `Checkin` access control applies to module data.

Example using Access Control Files

The following access control rules ensure that only members of the ASIC team can check into the ASIC project:

```
access init {
   set ASICteam { chan lynch kapoor vega }
}

access deny Checkin everyone when Object
"sync:///Projects/ASIC/*"
```

```
access allow Checkin only users $ASICteam when Object
"sync:///Projects/ASIC/*"
```

The first rule prevents everyone from checking into the project. The second rule builds on the first and permits members of the ASIC team, as specified by the `access init` definition, to check into the project. (See Using Access Commands for details on creating variables using `access init`.)

See Access Control Scripting for a filter script that shows how you use the `<Branch>` and `<NewBranchName>` parameters to prevent check-ins on a particular branch. Also, see Sample Access Controls for examples of requiring comments.

See Setting Up Access Controls for information on creating your own custom `AccessControl` files. See Using Access Commands for a list of the access commands and examples of how to use them.

Enterprise Design Access Map

When access control is delegated, the checkin access map is automatically available. The access map is located in the default DesignSync access control file, `$SYNC_DIR/share/AC_Components/AccessControl.hcm.`

Tip: The checkin access map has the ability to process commands coming in from addhref or rmhref (initiated through edithref, addhref, or rmhref), but DesignSync recommends that you leave the addhref and rmhref maps active to process those commands, rather than using the Checkin access map. The code sample below shows that Checkin declines access control when the Action originator is Addhref or Rmhref.

```
access map Checkin {

    # start with checkin, if action has to do with hrefs

    # add appropriate masks. we are limited to the 'from'

    # object, because here we do not have the target.

    lappend checkmasks [list revise $Object]



    if { $Action == "Addhref" } {

        lappend checkmasks [list fromconnect $Object]

    } elseif { $Action == "Rmhref" } {
        lappend checkmasks [list fromdisconnect $Object]
    }
    return [list "masks" $checkmasks]


access filter Checkin {
    if { $Action == "Addhref" || $Action == "Rmhref" } {
        if { $IsPlatformDelegated == "yes" } {
            return DECLINE
        }
    }
    return UNKNOWN
}
```

Related Topics

Access Control Scripting

Sample Access Controls

Setting Up Access Controls

Using Access Commands

Access Controls for Changing a Checkin Comment

Creating a New Version of a Module

Access Controls for Tagging

Access Controls for Changing a Checkin Comment

The `ChangeComment` access control allows access to for checkin comments.

Defining and Using the Access Control

The ChangeComment access control is defined in the default DesignSync access control file:

`$SYNC_DIR/share/AC_Components/AccessControl.ds:`

Because access controls affect only server operations, you cannot access control changing checkin comments of versions of client-side vaults.

By default, `ChangeComment` is included in the group definition DesignSyncActions and DesignSyncWriteActions.

Note: This access control was formerly set by an access control called `SetUrlProperty`. If you have references to `SetUrlProperty`, you must update them `ChangeComment`.

```
access define ChangeComment <Object> <Comment> <IsOwner>
```

```
Where:
```

- `<Object>` is the URL of the object whose property (comment) is being set. For example: `sync:///Projects/ASIC/layout/spec.doc;1.5`.
- `<Comment>` is the text of the comment. If the comment contains spaces, surround the comment with quotation marks (" ")
- `<IsOwner>` is a boolean set to `yes` or `no` depending on who is changing the checkin comment. <`IsOwner`> is `yes` if the user performing the operation is the object owner; otherwise, `no`.

Example

The following access filter ensures that you will be denied changing checkin comment in cases your new comment is less than 20 characters:

```
access filter ChangeComment {

    if {[string length $Comment] > 19} {

        return UNKNOWN

    }

    return "You must enter a comment of at least 20 characters."

}
```

The return of the `UNKNOWN` defers to any other access controls that might be set up.

In the example above, access is denied if the comment string has less than 20 characters, in which case you will see the message `"You must enter a comment of at least 20 characters."`.

Defining the Enterprise Access Map

When access control is delegated, you can enable the `ChangeComment` access map. A sample version of the access map is provided, commented out, in the custom DesignSync access control file for the server, `$SYNC_CUSTOM_DIR/servers/<serverName>/<portNumber>/share/AccessControl`.

```
access map ChangeComment {

    lappend checkmasks [list modify $Object]

    return [list "masks" $checkmasks]

}
```

Related Topics

Access Control Scripting

Sample Access Controls

Setting Up Access Controls

Using Access Commands

Access Controls for Checking In

Access Controls for Creating Branches

Defining and Using the Access Control

The default DesignSync access control file:

`$SYNC_DIR/share/AC_Components/AccessControl.ds`

defines a single action, `MakeBranch`, to control who can create new branches:

```
access define MakeBranch <Object> <BranchName>
```

Where:

- `<Object>` is the URL of the version from which the branch emanates (the branch-point version). For example: `sync:///Projects/ASIC/layout/spec.doc;1.5`.
- `<BranchName>` is the name (tag) of the branch being created.

The `MakeBranch` action is obeyed by any command that creates a branch: `mkbranch`, `ci` (with autobranching), `co` (with autobranching).

See Branching a Module to learn how the `MakeBranch` access control applies to module data.

Example

The following access rules ensure that only project leaders can create new branches:

```
access init {
   set projectLeaders { chan kapoor }
}

access deny MakeBranch everyone

access allow MakeBranch only users $projectLeaders
```

The first access rule prevents everyone from making branches. The second rule builds on the first and allows only project leaders, as defined using `access init`, to make branches. (See Using Access Commands for details on using `access init` to create custom variables.)

See Setting Up Access Controls for information on creating your own custom `AccessControl` files. See Using Access Commands for a list of the access commands and examples of how to use them.

Defining the Enterprise Access Map

When access control is delegated, you can enable the `MakeBranch` access map. A sample version of the access map is provided, commented out, in the custom DesignSync access control file for the server, `$SYNC_CUSTOM_DIR/servers/<serverName>/<portNumber>/share/AccessControl`.

```
access map MakeBranch {
    lappend checkmasks [list majorReviseAccess $Object]

   return [list "masks" $checkmasks]

 }
```

Related Topics

Access Control Scripting

Sample Access Controls

Setting Up Access Controls

Using Access Commands

Branching a Module

Access Controls for Creating Folders

The default DesignSync access control file:

`$SYNC_DIR/share/AC_Components/AccessControl.ds`

defines a single action, `MakeFolder`, to control for creating folders on the server:

```
access define MakeFolder <Object>
```

Where `<Object>` is the URL of the folder being created. For example: `sync:///Projects/ASIC/layout`.

Because access controls only affect server operations, you cannot access control operations in users' local workspaces. For example, `mkfolder /usr1/Projects/ASIC/layout` creates a local folder so this action cannot be access controlled. However, `mkfolder sync:///Projects/ASIC/layout` creates a server-side folder, so this action can be access controlled.

Example

The following access rules prevent everyone but project leaders from creating new folders for the ASIC project:

```
access init {
   set projectLeaders { chan kapoor }
}
```

```
access deny MakeFolder everyone when Object
"sync:///Projects/ASIC/*"

access allow MakeFolder only users $projectLeaders when Object
"sync:///Projects/ASIC/*"
```

The first access rule prevents everyone from making folders in the ASIC project. The second rule builds on the first and allows only project leaders, as defined using `access init`, to make folders. (See Using Access Commands for details on using `access init` to create custom variables.)

See Setting Up Access Controls for information on creating your own custom `AccessControl` files. See Using Access Commands for a list of the access commands and examples of how to use them.

Related Topics

Access Control Scripting

Sample Access Controls

Setting Up Access Controls

Using Access Commands

Access Controls for Deleting

The default DesignSync access control file:

```
$SYNC_DIR/share/AC_Components/AccessControl.ds
```

defines a single action, `Delete`, to control deleting an object:

```
access define Delete <Object> <Type> <Owner> <IsOwner>
```

Where:

- `<Object>` is the URL of the object being deleted. For example: `sync:///Projects/ASIC/layout/spec.doc;1.5`.
- `<Type>` is one of the following object types: `VERSION`, `FOLDER`, `VAULT`.
- `<Owner>` is:
  - The owner of the vault (when `<Type>` is `VAULT`).

    The owner of a vault is defined as the owner of the main branch (branch 1) of the design object. The default owner of a branch is the creator of the branch's initial version but ownership can be changed using the `setowner` command.

Deleting a vault removes all versions of a design object from the SyncServer. Because of the potential danger of this operation, the default `AccessControl.ds` file denies everyone the ability to delete a vault. This default access control can be modified using a site or server `AccessControl` file.

- The author of the version (when `<Type>` is `VERSION`).

  The `AccessControl.ds` file included with DesignSync denies everyone the ability to delete a version except for its owner. This default access control can be modified using a site or server `AccessControl` file.

- The owner of the SyncServer process (when `<Type>` is `FOLDER`).

* `<IsOwner>` is `yes` if the user performing the operation is the owner; otherwise, `no`.

Because access controls affect only server operations, you cannot access control the deletion of local folders, local files, and client vaults. The `FOLDER` and `VAULT` types refer to server-side folders and vaults.

`purge` uses `rmversion` to delete versions. By default, users can only `rmversion` versions that they created. This limits users' ability to remove a branch, because they will only be allowed to remove versions that they created. A user will be able to remove a branch, if the user created all of the versions on the branch. You can prevent the removal of branches, disallowing users from removing branches even when the user created all versions on the branch. This is shown in the Examples section below.

Examples

The following access rules control who can delete objects:

```
access init {
   set admin { lynch vega }
}

access deny Delete everyone

access allow Delete everyone when Type FOLDER \
when IsOwner "yes"

access allow Delete only users $admin when Type VAULT
```

The first rule prevents everyone from deleting any object. The second rule builds on the first and allows users to delete folders when they are the owner. The third rule specifies that only an Admin can delete vaults.

The following access rule prevents the removal of branches, even if the user created all of the versions on the branch. This is accomplished by disallowing removal of the first (".1") version on the branch.

```
access filter Delete {
if { $Type != "VERSION" } {
return UNKNOWN
}
if { [string match "*.1" $Object] } {
return "Removing the .1 version and the branch is not allowed"
}
return UNKNOWN
}
```

To allow some users to remove branches, modify the above access filter, allowing those users to remove the .1 version. Those users should also be allowed to remove versions that they did not create:

```
access allow Delete only users $admin when Type VERSION
```

See Setting Up Access Controls for information on creating your own custom `AccessControl` files. See Using Access Commands for a list of the access commands and examples of how to use them.

Related Topics

Access Control Scripting

Sample Access Controls

Setting Up Access Controls

Using Access Commands

Access Controls for Moving a File

The default DesignSync access control file:

```
$SYNC_DIR/share/AC_Components/AccessControl.ds
```

defines a single action, `Move`, to control moving a vault file:

```
access define Move <Object> <Type> <Owner> <IsOwner>
```

Where:

- `<Object>` is the file being moved. For example, the URL of a managed file being moved would be:
  `sync:///Projects/ASIC/layout/spec.doc;1.5`.
- `<Type>` is always `VAULT`.
- `<Owner>` is:
  - The owner of the vault (when `<Type>` is `VAULT`).

    The owner of a vault is defined as the owner of the main branch (branch 1) of the design object. The default owner of a branch is the creator of the branch's initial version but ownership can be changed using the `setowner` command.

    By default, it gives access to everyone, and can be modified using a site or server `AccessControl` file.

- `<IsOwner>` is `yes` if the user performing the operation is the owner; otherwise, `no`.

Because access controls affect only server operations, you cannot access control the moving of files in local workspaces. The type `VAULT` refer to server-side vaults.

Example

The following access rules control who can move a file:

```
access init {
set admin { kapoor }
}
access deny Move everyone

access allow Move everyone when Type VAULT when IsOwner "yes"

access allow Move only users $admin when Type VAULT
```

The first rule prevents everyone from moving a file. The second rule builds on the first and allows users to move files only when the type is vault and when they are the owner of the vault. The third rule specifies that only an Admin can move files when the type is vault.

See Setting Up Access Controls for information on creating your own custom `AccessControl` files. See Using Access Commands for a list of the access commands and examples of how to use them.

Related Topics

Access Control Scripting

Sample Access Controls

Setting Up Access Controls

Using Access Commands

Access Controls For Object Caching

The default access control file, `$SYNC_DIR/share/AC_Components/AccessControl.ds` defines whether users can turn on or off caching for individual objects. This is an administrative feature that should be reserved for the mirror administrator. By disabling caching for specific objects, you are able to finely control access to that IP.

defines a single action, `Caching`, to control changing checkin comments:

```
access define Caching <Object> Enable|Disable
```

```
Where:
```

- `<Object>` is the URL of the object whose property (comment) is being set. For example: `sync:///Modules/ChipDesign/Chip`.
- `Enable|Disable` string representing the action which is either enable or disable.

Because these access controls affect only server operations, you cannot access control of caching on client-side vaults.

By default, access to the caching enable/disable commands is denied to everyone:

```
access deny Caching everyone -because "Default access controls
disallow changing the cacheable state of a vault.\nContact your
Synchronicity tool administrator to enable this capability."
```

Note: Access to the `caching status` command is allowed for all users with BrowseServer access.

Example

The following access controls caching enable/disable access for your DesigSync administrators.:

```
access init {
   set SyncAdmin { admin syncowner syncmgr }
}

access allow Caching only users $SyncAdmin when Object
"sync:///Projects/ASIC/*"
```

Related Topics

DesignSync Action Definitions

Access Controls for Retiring

The default DesignSync access control file:

```
$SYNC_DIR/share/AC_Components/AccessControl.ds
```

defines a single action, `Retire`, to control retiring an object:

```
access define Retire <Object>
```

Where `<Object>` is the URL of the branch being retired. For example, `sync:///Projects/ASIC/layout/spec.doc;1` might be the main branch for `spec.doc`.

Moving or renaming a managed object (see the `mvfile` command) requires `Retire` permission in order to retire the current branch of the existing vault.

Example

The following access rules control who can retire objects in the ASIC project:

```
access init {
   set projectLeaders { chan kapoor }
}

access deny Retire everyone when Object
"sync:///Projects/ASIC/*"

access allow Retire only users $projectLeaders when Object
"sync:///Projects/ASIC/*"
```

The first rule ensures that no one can retire any object in the ASIC project. The second rule builds on the first and allows the project leaders, as defined using `access init`, to retire objects. (See Using Access Commands for details on using `access init` to create custom variables.)

See Setting Up Access Controls for information on creating your own custom `AccessControl` files. See Using Access Commands for a list of the access commands and examples of how to use them.

Related Topics

Access Control Scripting

Sample Access Controls

Setting Up Access Controls

Using Access Commands

Access Controls for Setting Owners

The action, `SetOwner`, controls setting the owner of an object.

Defining and Using the SetOwner Access Control

The `setOwner` definition is defined in the default DesignSync access control file:

```
$SYNC_DIR/share/AC_Components/AccessControl.ds

access define SetOwner <Object> <NewOwner> <IsOwner>
```

Where:

- `<Object>` is the URL of the object for which the owner is being set. For example: `sync:///Projects/ASIC/*`
- `<NewOwner>` is the user name of the desired owner.
- `<IsOwner>` is `yes` if the user performing the operation is the same as the current owner; otherwise `no`.

Example

The following access rules ensure that only the owner of an object or an administrator can set a new owner for an object:

```
access init {
   set admin { lynch vega }
}

access deny SetOwner everyone when IsLockOwner "no"
access allow SetOwner users $admin
```

The first rule specifies that only the owner of an object can set a new owner. The second rule builds on the first to also allow administrators, as defined using `access init`, to set a new owner. (See Using Access Commands for details on using `access init` to create custom variables.)

See Setting Up Access Controls for information on creating your own custom `AccessControl` files. See Using Access Commands for a list of the access commands and examples of how to use them.

Defining the Access Map

When access control is delegated, you can enable the `setOwner` access map. A sample version of the access map is provided, commented out, in the custom DesignSync access control file for the server, `$SYNC_CUSTOM_DIR/servers/<`*`serverName`*`>/<`*`portNumber`*`>/share/AccessControl`.

```
access map SetOwner {

    # verify changeOwner

    lappend checkmasks [list changeOwner $Object]

    return [list "masks" $checkmasks]

}
```

Related Topics

Access Control Scripting

Sample Access Controls

Setting Up Access Controls

Using Access Commands

Access Controls for Tagging

The `Tag` access control defines a single action to allow adding, deleting, or replacing a tag on an object:

Defining and Using the Access Control

The Tag access control is defined in the default DesignSync access control file:

`$SYNC_DIR/share/AC_Components/AccessControl.ds`

Note: The `ci` command allows you to tag objects on checkin. In order to use this feature, users must be allowed to ADD a Tag.

See Tagging a Module to learn how the `Tag` access control applies to module data.

```
access define Tag <Object> <NewTag> <Action> <Type> <Mutability>
```

Where:

- `<Object>` is the URL of the object being tagged. For example: `sync:///Projects/ASIC/layout/spec.doc;1.5`.
- `<NewTag>` is the name of the tag.
- `<Action>` the action you want to perform:
  - `ADD` - `<NewTag>` is added to `<Object>`.
  - `DELETE` - `<NewTag>` is deleted from `<Object>`. A tag can exist on only one version or branch of an object, so specifying a version or branch number for `<Object>` is ignored.
  - `REPLACE` - `<NewTag>` is moved to `<Object>`. The access-control action is `REPLACE` only when an actual replace is required (`<NewTag>` already exists). If a user tries to replace a nonexistent tag from the command line or graphical interface, then the access-control action is `ADD` and the type of object whose tag is being deleted is `<Type>`.

- `<Type>` - the type of object you want to operate on:

  - `VERSION` - A version tag is being added, moved, or deleted.
  - `BRANCH` - A branch tag is being added, moved, or deleted.
- `<Mutability>` - this parameter pertains only to module data
  - `MUTABLE` - the tag is mutable
  - `IMMUTABLE` - the tag is immutable

Notes:

- If a tag is being replaced, and is being changed from mutable to immutable at the same time, then the `<Mutability>` value will be `IMMUTABLE`.
- For non-module data, the `<Mutability>` value is `MUTABLE`.

Example

The following access rule ensures that only project leaders can delete or replace tags:

```
access init {
   set projectLeaders { chan kapoor }
}

access allow Tag only users $projectLeaders \
  when Action DELETE

access allow Tag only users $projectLeaders \
when Action REPLACE
```

The two actions cannot be specified in a single `access allow` rule because the actions are ANDed. You cannot have an operation in which a tag is simultaneously deleted and replaced.

The `$projectLeaders` variable is defined using an `access init` command. See Access Commands for details on using `access init` to create custom variables.

See Setting Up Access Controls for information on creating your own custom `AccessControl` files. See Using Access Commands for a list of the access commands and examples of how to use them.

Defining the Enterprise Access Map

When access control is delegated, you can enable the `Tag` access map. A sample version of the access map is provided, commented out, in the custom DesignSync access control file for the server, `$SYNC_CUSTOM_DIR/servers/<serverName>/<portNumber>/share/AccessControl`.

```
access map Tag {

    lappend checkmasks [list modify $Object]

    return [list "masks" $checkmasks]

}
```

Related Topics

Access Control Scripting

Sample Access Controls

Setting Up Access Controls

Using Access Commands

Tagging a Module

Access Controls for Checking In

Access Controls for Unlocking

The single action, `Unlock,` controls unlocking an object or canceling the lock on an object:default

Defining and Using the Unlock Access Control

The Unlock defintion is located in the DesignSync access control file:

```
$SYNC_DIR/share/AC_Components/AccessControl.ds
```

```
access define Unlock <Object> <IsLockOwner>
```

Where:

- `<Object>` is the URL of the object being unlocked. For example: `sync:///Projects/ASIC/layout/spec.doc;1.5`.
- `<IsLockOwner>` is `yes` if the user performing the unlock operation is the owner, otherwise `no`.

The `Unlock` action refers to a server-side operation, in this case unlocking a branch. You can initiate the unlocking of a branch with both the `cancel` and `unlock` commands, so both commands are access controlled through the `Unlock` action.

See Unlocking Module Data to learn how the `Unlock` access control applies to module data.

Example

The following access rules prevent everyone except lock owners and administrators from canceling the lock on an object in the ASIC project:

```
access init {
   set admin { lynch vega }
}

access deny Unlock everyone \
  when Object "sync:///Projects/ASIC/*" \
  -because "To unlock, you must be the lock owner or an admin."

access allow Unlock everyone when IsLockOwner "yes"
access allow Unlock users $admin
```

The firs access rule prevents everyone from canceling a lock on an object in the project. The second rule builds on the first and lets lock owners cancel their own locks. The third rule builds on the first two rules and lets administrators, as defined using `access init`, cancel locks. (See Using Access Commands for details on using `access init` to create custom variables.)

See Setting Up Access Controls for information on creating your own custom `AccessControl` files. See Using Access Commands for a list of the access commands and examples of how to use them.

Defining the Access Map

When access control is delegated, you can enable the `Unlock` access map. A sample version of the access map is provided, commented out, in the custom DesignSync access control file for the server, `$SYNC_CUSTOM_DIR/servers/<serverName>/<portNumber>/share/AccessControl`.

```
access map Unlock {

    # verify unlock

    lappend checkmasks [list unlock $Object]

    return [list "masks" $checkmasks]

}
```

Related Topics

Access Control Scripting

Sample Access Controls

Setting Up Access Controls

Using Access Commands

Unlocking Module Data

Access Controls for Unretiring

The default DesignSync access control file:

`$SYNC_DIR/share/AC_Components/AccessControl.ds`

defines a single action, `Unretire`, to control unretiring an object:

```
access define Unretire <Object>
```

Where `<Object>` is the URL of the branch being unretired. For example, `sync:///Projects/ASIC/layout/spec.doc;1` is the main branch for `spec.doc`.

Example

The following access rules ensure that users can retire objects in the ASIC project, but not in any other project:

```
access deny Unretire everyone

access allow Unretire everyone when Object
"sync:///Projects/ASIC/*"
```

See Setting Up Access Controls for information on creating your own custom `AccessControl` files. See Using Access Commands for a list of the access commands and examples of how to use them.

Related Topics

Access Control Scripting

Sample Access Controls

Setting Up Access Controls

Using Access Commands

Access Controls for Upload

The default access control file, `$SYNC_DIR/share/AC_Components/AccessControl.ds` defines a set of actions to control upload actions.

The action `TransferFile`, controls whether a user can upload a compressed archive file (tar or zip) into DesignSync and uses the abilities of DesignSync to update only the differences between the files in the archive from version to version. (using the `upload` command). The access control definition is:

```
access define TransferFile {Destination}
```

where

- *Destination* takes a value of `Tmp` and indicates the location of the temporary storage of the uncompressed file while the comparison operations are in process if the upload command is specified with the -vault option. The comparison is done in the server tmp area SYNC_TMP_DIR.

Note: When the upload command is run with -workspace, the comparison operations are performed in the workspace. You still require access to the TransferFile command.

By default, access to `TransferFile` is denied to everyone.

```
access deny TransferFile everyone -because "Default access
controls disallow file transfer to the server.\nContact your
Synchronicity tool administrator to enable this capability."
```

Additional Access Controls Required

The user may also need the following additional permissions and the appropriate permissions for the vault type (Module or Files based)

- Checkin - for all upload operations.
- Tag - for all upload operations.
- Checkout - for upload operations after the initial upload.
- Mkmod -for the initial upload only, if you are checking the IP into a module
- MemberRemove - for upload operations on a module after the initial upload. An upload operation requires this permission in case an object was removed from newest version of the archive.
- MakeFolder - for the initial upload only, if the vault folder doesn't exist.
- Retire - for file-based operations after the initial upload. An upload operation requires this permission in case an object was removed from newest version of the archive.
- MakeBranch - for file-based operations that upload to a new branch.

Access Controls for Workspace Duplication

The default access control file, `$SYNC_DIR/share/AC_Components/AccessControl.ds` defines actions to control duplicating a workspace actions.

The action `DuplicateWorkspace` controls whether a user can duplicate a workspace that is designated for duplication. The access control definition is:

```
access define DuplicateWorkspace {Object Path}
```

where

- Object is the string representation of the URL of the module including the selector,
  for example: syncs:///Modules/ChipDesign/ChipNZ7;Trunk .
- Path is the String representing the path of the Module in workspace.

By default, access to `DuplicateWorkspace` is allowed for all users .

```
access allow DuplicateWorkspace everyone
```

Additional Access Controls Required

The user also needs the following additional permissions:

- Browse access required for the module being duplicated.

Defining Enterprise Access Map

When access control is delegated, you can enable the `BrowseServer` access map. A sample version of the access map is provided, commented out, in the custom DesignSync access control file for the server, `$SYNC_CUSTOM_DIR/servers/<serverName>/<portNumber>/share/AccessControl`.

```
# access map DuplicateWorkspace {
#     # verify checkout
#     lappend checkmasks [list read $Object]
#     return [list "masks" $checkmasks]
# }
```

Access Controls for Groups of Commands

You may want to grant or deny complete access to DesignSync actions to some set of users. Variables defined in the default DesignSync access control file (`$SYNC_DIR/share/AC_Components/AccessControl.ds`) make it more convenient to set the same access rights on DesignSync actions.

You also can create your own variables for sets of commands in your custom `AccessControl` file.

Predefined Variables

The default `AccessControl.ds` file defines some default variables which can be used by the user:

Variable	Description	DesignSync Commands Included

`$DesignSyncActions`	Controls all the commands defined in the AccessControl.ds file By default access to this .	`BrowseServer Unlock Checkin Checkout Tag Delete Retire Unretire MakeFolder Move SwitchLocker MakeBranch SetOwner ChangeComment Lock`
`$DesignSyncWriteActions`	Controls all DesignSync commands that perform modifications.	`Unlock Lock Checkin MakeBranch Tag Delete Retire Unretire MakeFolder SetOwner ChangeComment Move SwitchLocker`
`$DesignSyncReadActions`	Controls all DesignSync commands that allow you to read DesignSync data.	`BrowseServer Checkout DuplicateWorkspace`
`$DesignSyncDevelopmentActions`	Controls all the DesignSync commands that work on Enterprise Developments.	`DesignSyncDevelopments AddDevelopmentInstance ModifyDevelopmentInstance DeleteDevelopmentInstance`

Note: While access is granted to all commands defined in the AccessControl.ds file by default, immediately afterwards, access is denied for the following operations:

- Deleting a vault
- Deleting a version you do not own
- Switching owner
- Writing directly to the access control vault

Another variable, `$AllActions`, controls all ProjectSync, DesignSync and Module actions. The `$AllActions` variable is defined in the `AccessControl.ps` file. See Access Controls for Groups of Commands in the ProjectSync section for details.

For example, you could use the `$DesignSyncActions` variable to prevent users from performing any DesignSync operations while you are upgrading your server. In your custom `AccessControl` file, you could specify:

```
access deny $DesignSyncActions everyone
```

Custom-Defined Variables

In your custom `AccessControl` file, you can use the `set` command to create variables that represent your own groupings of access control actions.

For example, to create a variable that controls the ability to retire, unretire, and delete objects, you could specify:

```
set RetireActions {Retire Unretire Delete}
```

You can then use the variable `$RetireActions` in your access control rules.

You also can create variables that include other variables. However, when declaring a list that requires variable substitution, you must enclose the list in quotation marks, not curly braces. (Curly braces prevent variable substitution.) For example:

```
set AdminActions "$RetireActions Tag"
```

Related Topics

Access Control Scripting

Sample Access Controls

Setting Up Access Controls

Using Access Commands

Example DesignSync Access Controls

Sample Access Controls

Sample access controls are provided in:

```
$SYNC_DIR/share/examples/ExampleAccessControl
```

You can use this sample `ExampleAccessControl` file as a template for creating access control files.

This topic provides examples of revision-control access controls that you might put in your custom `AccessControl` file:

Denying or Allowing Access by Users

Using `when` Clauses

Using `access filter` with an `access init` Block

Requiring Check-in Comments

Using `access filter` to Check an Action

Providing an Error message When Permission Is Denied

See Access Control Scripting for more advanced examples.

Note: Do not edit any of the access control files in the `$SYNC_DIR/share` area; you edit the site or server `AccessControl` file. See Setting Up Access Controls for the locations of these `AccessControl` files.

See Setting Up Access Controls for information on creating your own custom `AccessControl` files. See Using Access Commands for a list of the access commands and examples of how to use them.

Denying or Allowing Access by Users

```
# Everyone but bob can check files in
access deny Checkin users bob

# jack and jill can't check files in or out
access deny {Checkin Checkout} users {jack jill}

# only frida can tag anything
access allow Tag only users frida
```

Using when Clauses

```
# Only 'bmeister' can check in Makefiles.
# Note that the initial wildcard (*) matches any path so
# that any Makefile in your hierarchy is a match. The second
# wildcard ensures a match against any version of Makefile.

access allow Checkin only users bmeister \
  when Object "*/Makefile;*"

# No one but owner can unlock files
access deny Unlock everyone when IsLockOwner "no"

# Deny Bob all access to the Projects/ASIC project. Uses the
# $DesignSyncAccess variable to simplify the specification,
# otherwise a deny statement would be needed for each
# DesignSync action (Checkin, Tag, and so on).
```

```
access deny $DesignSyncActions users bob \
  when Object "sync:///Projects/ASIC/*"

# Multiple when clauses are ANDed; this access control
# verifies that the user is rick if the RELEASE tag is being
# applied to files in 'Code' directories.

access allow Tag only users rick when NewTag "RELEASE" \
  when Object "sync:///*/Code/*" when Action ADD
```

Using access filter with an access init Block

```
# set up a variable that defines who the project leader is
access init {
  set projectLeader jane
}

# Only the project leader can checkin
access filter Checkin {
  if {$user == $projectLeader} {
    return ALLOW
  }
  return DENY
}
```

For access filters, you may get better performance using `access global` instead of `access init` to define variables and procs.

When `access init` is used in filters, the `access init` statement is sourced each time the filter is run. This behavior can introduce performance penalties for operations such as viewing a note because the statement is sourced for each note. The `access global` is used inside filter scripts and is sourced only once, when the access control system is initialized. See the `access init` and `access global` command descriptions for more information.

Requiring Check-in Comments

A common project requirement is that every check-in must include a comment of a given length. You can define access control statements to enforce this requirement. The **Require the specification of a Check In comment** option from the **Tools=>Options=>General** dialog box helps users comply with the comment-length requirement that you define.

Here are two examples of access controls for requiring check-in comments:

```
# Deny check-in when there is no comment (length is 0).
# Return a message string if access denied.
access deny Checkin everyone when CommentLen "0" \
  -because "You must enter a comment."

# Deny check-in when the comment is less than 20 characters.
# The return value of UNKNOWN defers to any other access
# controls that might be set up. Returns a message string
# if access is denied.
access filter Checkin {
  if {[expr $CommentLen > 19]} {
    return UNKNOWN
  }
  return "You must enter a comment of at least 20 characters."
}
```

Using access filter to Check an Action

This example checks the action and, depending on the action, performs a lock check or a comment check.

```
access init {
  set admin "elaine"
}

access filter { Checkin Checkout } {
  if { $action == "Checkout" } {
    puts "action = $action"
    # allow only admin users to Checkout with lock
    if { $Lock == "yes" } {
      if { [lsearch -exact $admin $user] >= 0 } {
        return ALLOW
      }
      return "You do not have rights to Checkout with a lock"
    }
  }
  if { $action == "Checkin" } {
    if {[expr $CommentLen > 19 ]} {
      return ALLOW
    }
    return "You must enter a comment of at least 20 \
            characters."
  }
}
```

Providing an Error message When Permission Is Denied

You can provide an error message when permission is denied by using a `-because` switch for an `access deny` statement or returning a message in place of the `DENY` value in an `access filter` statement:

```
# Prevent user george from checking in and provide a message
access deny Checkin users {george} \
  -because "george is not allowed to check in."
```

Related Topics

Access Control Scripting

Setting Up Access Controls

Using Access Commands

Revision Control Action Definitions

Sample Access Controls (ProjectSync)

Sample Server Access Controls

User Authentication Action Definitions

Using Access Command Qualifiers

Access Control Scripting

For many security requirements, `access allow` and `access deny` rules are sufficient. These rules let you allow or deny access to DesignSync actions, such as `Checkin`. If you have more complex security requirements, you might need to develop `access filter` scripts. This section provides examples of access controls implemented using `access filter` scripts, as well as examples of some general access control solutions:

Preventing Users From Checking into a Particular Branch

Preventing Check-Ins of Unlocked Files (Forced Locking Model)

Setting Up Access Controls for Projects

Restricting Access to Parts of a Project

Allowing Project Owners to Delete Project Vaults

Note: Do not edit any of the access control files in the `$SYNC_DIR/share` area; you edit the site or server `AccessControl` file. See Setting Up Access Controls for the locations of these `AccessControl` files.

Preventing Users From Checking into a Particular Branch

Problem

You want to prevent users from checking into a branch, while allowing them to check into other branches. Closing off branches is a useful operation when you are merging branches. In cases of merging branches, you might want to prevent all users from checking objects into a branch. In other cases, you might want to allow a specified group of users, such as administrators, to check objects into a branch.

Solution

The following access filter prevents all users from checking into the "`Trunk`" branch:

```
access filter Checkin {
  #
  # Check if this command creates a new branch.
  if {$NewBranchName != ""} {
    # This is a new branch, so check for Trunk
    if {$NewBranchName == "Trunk"} {
      return "You are not allowed to create new \
              files on Trunk."
    }
    # New branch is not tagged with "Trunk", so don't deny.
    return UNKNOWN
  }
  #
  # Not a new branch so extract all the
  # branch tags associated with this branch.
  set tags [url tags -btags $Branch]
  if {[lsearch [split $tags " ,"] "Trunk"] < 0} {
    # Trunk tag is not found.
    return UNKNOWN
  }
  # Trunk tag is found.
  return "You are not allowed to check in to Trunk."
}
```

Discussion

The access filter above applies to the `Checkin` action. Access filters restrict the specified action based on the script provided within the access filter. The return value of `UNKNOWN` defers to any other `Checkin` access controls that have been set up; thus, the user has satisfied the filter script if the value returned is `UNKNOWN`. Any other return value besides `ALLOW` and `UNKNOWN`, such as `DENY` or a text string, denies access to the action.

Parameters are passed into the script based on the type of action specified. For example, the predefined `Checkin` action has the following definition:

```
# Checkin:
# Arguments:
#    Object == string representation of the URL of the object
#              being checked in. eg:
#              sync:///Projects/ASIC/layout/spec.doc;1.5
#    Comment == description comment that was entered for this
#               check-in.
#    CommentLen == length (number of characters) in Comment
#    Branch == Url of the branch where the file will be
#              checked into, in case of a
#              non-branching check-in
#    NewBranchName == Name of the new branch to create, in
#              case of a branching check-in. If not a branching
#              check-in, then NewBranchName will be empty
#              string ("")

access define Checkin {Object Comment CommentLen Branch
NewBranchName}
```

For an access filter restricting the `Checkin` action, the access control system passes the script the <`Object`>, <`Comment`>, <`CommentLen`>, `<Branch>`, and <`NewBranchName`> parameters.

The script above accesses both the `<Branch>` parameter and the <`NewBranchName`> parameter. A check-in can be:

- A nonbranching check-in, where you are checking into an existing branch. For this case, the script checks that the `<Branch>` parameter does not contain "`Trunk`".
- A branching check-in, where you are checking in a new object or autobranching (creating new branches automatically during check-ins). For these cases, the script checks that the <`NewBranchName`> parameter does not include the "`Trunk`" branch.

Note that the `<Branch>` parameter is not sufficient to prevent checking in on particular branches because new or autobranched objects do not have a `<Branch>` URL until after they are checked in; thus, you need to examine the `<NewBranchName>` parameter for new or autobranched objects. (See DesignSync User's Guide: Autobranching: Exploring "What If" Scenarios for details of autobranching.)

As an example, assume a user tries to check in an object, "`sync://holzt:2647/Projects/Asic/x.v;1.4`", thus creating a new object, "`sync://holzt:2647/Projects/Asic/x.v;1.5`". The script determines the object's branch, "`sync://holzt:2647/Projects/Asic/x.v;1`", then gathers the tags associated with that branch. In this case, the branch is "`Trunk`", so the check-in is prevented.

In addition to restricting check-ins to an existing branch, you can also restrict the creation of new branches, by setting up an access control for the `MakeBranch` action.

You can enhance this script by allowing a select group of users to check into the closed branch or create new objects on the closed branch. The following `access init` statement defines the members of the release engineering team:

```
access init {
  set ReleaseTeam {user1 user2}
}
```

To allow the release engineering team to check into the closed branch, add the following lines to the beginning of the `Checkin` access filter script. This code checks whether the `<user>` variable, a variable passed implicitly to all access filter scripts, is included in the `<ReleaseTeam>` namespace variable:

```
# If the user is a member of the Release Team, return
# without restricting access. The "lsearch" command below
# returns -1 if $user is not found in $ReleaseTeam.
if {[lsearch $ReleaseTeam $user] >= 0 } {
  # Must be a member of the Release Team - No Controls
  return UNKNOWN
}
```

Preventing Check-Ins of Unlocked Files (Forced Locking Model)

Problem

You want to enforce a locking model where a user must have a lock on an object to be able to check it in.

Solution

The following script enforces a locking development model, where users can only check in files when they have them checked out with locks.

```
# This filter should be added to the AccessControl file
# in the custom area.
#
# Notes:
# - This will affect performance, as checks will be
#   conducted before the server allows the check-in.
#
# Filter function to deny check-in unless the branch
# is locked:

access filter Checkin {
  # If new branch, allow its creation
  if {$NewBranchName != ""} {
    return UNKNOWN
  }

  # Branch exists, so now allow check-in
  # only if locked, else deny
  url properties $Branch props
  if { $props(locked) == 0 } {
    return "File must be locked for Checkin."
  } else {
    return UNKNOWN
  }
}
```

Discussion

A lock is applied to a branch of an object (so that different users can lock different branches of an object).

Setting Up Access Controls for Projects

Problem

You would like to customize the access controls for each project.

Solution

The following access commands prevent any users not on the Asic team from checking objects into the `Asic` project.

```
# Set up a variable that defines the Asic team
access init {
  set AsicTeam {mgeorge mmartin lsmith fpaul}
}

# Only the Asic team members can check into Project Asic
access allow Checkin only users $AsicTeam \
  when Object "sync:///Projects/Asic/*" \
  -because "You are not a member of the Asic project team."
```

Discussion

Each DesignSync or ProjectSync action takes an `<Object>` parameter that can be used with a wildcard to match entire projects. For example, assume the `Asic` project has been checked into `sync://{host}:{port}/Projects/Asic`. The `access init` statement above creates a variable, `AsicTeam`, containing the members of the Asic team. Then, the `access allow` rule limits access to the Asic project based on the `AsicTeam` variable.

Notice that the host and port are not included in the project URL, `sync:///Projects/Asic/*`. Because access control scripts are server-side scripts, you leave out the host and port in the project URL.

Restricting Access to Parts of a Project

Problem

You would like to have fine-grained control over which members of your project team can access specific projects, modules within projects, or Cadence view types.

Solution

The following access commands deny `BrowseServer` access to all projects, then selectively allow access for specific users:

```
access deny BrowseServer everyone \
  when Object sync:///Projects/*

access allow BrowseServer users {kdalton} \
  when Object sync:///Projects/verif*

access allow BrowseServer users {cdent} \
  when Object sync:///Projects/*
```

The vault directory structure corresponding to the access control rules follows:

```
$SERVER_ROOT/
    Projects/
      desACS/
        top/alu/alu.gv
        top/alu/alu.v
        top/alu/mult8.gv
        top/alu/mult8.v
      verification/
        jtag.v
        test.v
```

Because of the first access control rule, user `kdalton` can browse at the top level of the server, but cannot browse into `Projects` and cannot set vault to `Projects` (using the `setvault` command) :

```
stcl> url contents sync://acae220:2647
sync://acae220:2647/Projects sync://acae220:2647/Partitions
sync://acae220:2647/sync_server_trace.log

stcl> url contents sync://acae220:2647/Projects
som-E-199: Permission denied by the AccessControl system.

stcl> pwd
/home/kdalton/Projects

stcl> setvault sync://acae220:2647/Projects .
An error occurred while setting the Vault URL.
The error was: som: Error 199: Permission denied by
the AccessControl system.

Please verify that:
  The vault URL is correct.
  The SyncServer is running.
  You have write permission for the parent directory
  of the directory for which you are setting the vault.
```

The second access control rule selectively overrides the `BrowseServer access deny` rule for user `kdalton` who can browse into the `Projects/verification` vault directory, but no other vault directories:

```
stcl> url contents sync://acae220:2647/Projects
som-E-199: Permission denied by the AccessControl system.

stcl> url projects sync://acae220:2647/Projects
som-E-199: Permission denied by the AccessControl system.
```

```
stcl> url contents sync://acae220:2647/Projects/verification
{sync://acae220:2647/Projects/verification/test.v;}
{sync://acae220:2647/Projects/verification/jtag.v;}
sync://acae220:2647/Projects/verification/sync_project.txt

stcl> pwd
/home/kdalton/Projects/verification

stcl> setvault sync://gilmour:30048/Projects/verification .

Beginning Setvault operation...

Setting Vault URL sync://acae220:2647/Projects/verification
on folder file:///home/kdalton/Projects/verification

Finished Setvault operation.

stcl> populate -get -recur
Beginning Populate operation...

test.v        : Success - Fetched version: 1.1
jtag.v        : Success - Fetched version: 1.1

Populate operation finished:
/home/kdalton/Projects/verification

{Objects succeeded (2)} {}
```

The third access control rule provides user `cdent` with complete `BrowseServer` access to the `Projects` vault directory structure:

```
stcl> pwd
/home/cdent/Projects

stcl> setvault sync://acae220:2647/Projects .
Beginning Setvault operation...
Setting Vault URL sync://acae220:2647/Projects on
folder file:///home/cdent/Projects

Finished Setvault operation.

stcl> populate -get -recur
Beginning Populate operation...
desACS/top/alu/alu.v     : Success - Fetched version: 1.1
desACS/top/alu/mult8.v   : Success - Fetched version: 1.1
desACS/top/alu/mult8.gv : Success - Fetched version: 1.1
desACS/top/alu/alu.gv    : Success - Fetched version: 1.1
```

```
verification/test.v      : Success - Fetched version: 1.1
verification/jtag.v      : Success - Fetched version: 1.1

Populate operation finished: /home/cdent/Projects

{Objects succeeded (6)} {}
```

Discussion

This example shows how you can restrict access to all users and then use wildcards to allow access to particular projects for particular users. You also can use `access init` statements to specify the groups of users that are allowed access to particular projects, subprojects, or view types. For example, the following access controls allow only layout engineers to edit layout views:

```
access init {
  set layoutusers "jdonne mleni snardeau"
}

access allow Checkout only users $layoutusers \
  when Lock "yes" when Object "*/layout.sync.cds;*"
```

Note: This method of restricting editing based on view type depends on a consistent naming convention for cell view objects. (Cadence does not require views to be named according to their type.) For example, if layout views are always named `layout` then `"*/layout.sync.cds;*"` pertains to layout views. The first "*" wildcard matches the vault path leading to the object. The second "*" wildcard matches any version of the object.

You can also specify that all DesignSync actions be controlled for a particular project and group of users, as in the following example:

```
access init {
  set ASIC1team {psmith mabraham gtarbox}
  set ASIC2team {jboswell cdent}
}

access allow $DesignSyncActions only users $ASIC1team \
  when Object "sync:///Projects/ASIC1/*"
access allow $DesignSyncActions only users $ASIC2team \
  when Object "sync:///Projects/ASIC2/*"
```

For each project in this example, corresponding access control rules allow only the project's team to perform any DesignSync actions on that project's data. The `DesignSyncActions` variable is defined in the access control file,

<SYNC_DIR>/share/AC_Components/AccessControl.ds included with DesignSync.

Note: This method differs from the BrowseServer access controls above in that the BrowseServer access controls deny access to the top level of the SyncServer for all users, whereas the method used in the DesignSyncActions access controls allows all users to retain access to the top level of the SyncServer. This method of keeping the SyncServer accessible to all users has two advantages:

- DesignSync DFII users can continue to use the Server Browser.

  Access to the top level of the SyncServer is required for DesignSync DFII users; the DesignSync DSII's Server Browser must recognize that the server is active.

- All users can obtain a list of the projects on the SyncServer.

  This method of allowing all users access to the top level of the SyncServer enables them to use the url projects command to list out all of the projects on the SyncServer. The access controls prevent them only from accessing those projects they do not have permission to access.

Allowing Project Owners to Delete Project Vaults

Problem

The access control file included with DesignSync denies access to the Delete action for all users. The Delete action controls the deletion of vaults, versions, and vault folders on SyncServers; client-side objects are not access-controlled by the Delete action.

Many sites modify their custom AccessControl files to allow administrators to remove vaults, versions, and vault folders, as in the following access control rules:

```
access init {
  set admin { syncmgr }
}

access allow Delete only users $admin
```

These access controls allow only authorized users, defined in the admin list, to perform the rmvault, rmversion, and rmfolder commands on server-side objects.

These access controls prevent project owners from deleting server-side vaults, versions, and folders from their own projects. You might want to allow project owners to be able to delete objects within their own projects.

Note: You set up projects by associating a project name with an owner, a DesignSync vault, and an optional configuration using ProjectSync. See ProjectSync Help to learn how to set up and manage projects.

Solution

The following access controls allow project owners to delete vaults within their projects:

```
access init {
  set admin { mkaley gmining pquinn mskelley }
}

# Procedures and variables for Delete filter.
access init Delete {

################################################################
##
###
###     getSyncProjOwner
###
### This procedure returns the project owner for the object
### URL passed. It returns an empty string if this is not
### an official project (as defined in ProjectSync).
###
################################################################
##

proc getSyncProjOwner {objectURL} {
  url properties $objectURL props
  set type $props(type)
  if { $type == "Project" } {
    return [url owner $objectURL]
  } else {
  # recurse on parent
    set parent [url container $objectURL]
    if { $parent != $objectURL } {
      return [getSyncProjOwner $parent]
    }
  }

  # Here if no project found (as defined in ProjectSync),
  # and thus no owner.
  return ""
  }
}
```

```
access filter Delete when Type VAULT {

# This filter allows the vault delete function if the
# invoking user is a $admin, or the owner of the project
# containing the vault.
#
# By default, the delete function is not allowed for vaults.
# Thus we only need to return ALLOW or UNKNOWN.

  if { [lsearch -exact $admin $user] >= 0 } { return ALLOW }
  set proj_owner [getSyncProjOwner $Object]
  if { "$proj_owner" == "$user" } { return ALLOW }
  return UNKNOWN
}
```

Discussion

The Delete access filter in this example calls the `getSyncProjOwner` procedure to determine the project owner of the vault being removed. If the user attempting to delete the vault is either the project owner (as defined using ProjectSync) or is included in the `admin` list, he or she can delete the vault.

Related Topics

Access Control Scripting

Using Access Commands

Revision Control Action Definitions

Sample Access Controls (DesignSync)

Sample Access Controls (ProjectSync)

Sample Server Access Controls

Setting Up Access Controls

Access Controls for Modules

Modules Action Definitions

Modules Action Definitions

You set up access controls on particular module **actions**, or operations. To set up an access control on an operation, the operation must have an action definition specified with an `access define` command. If an action definition exists for an operation, you can control access to that operation using the stcl `access allow`, `access deny`, and `access filter` commands. You can control access to operations on module data by using the `access decline` command.

DesignSync provides predefined actions corresponding to most module operations you might want to access control. These actions are defined in the default access control file for modules:

```
$SYNC_DIR/share/AC_Components/AccessControl.hcm
```

See Introduction to Access Control for details on the individual files used to define access controls.

Important: Do not edit the `$SYNC_DIR/share/AC_Components/AccessControl.hcm` file; changes will be lost upon upgrading. Instead, edit your site or server custom `AccessControl` file (see Setting Up Access Controls).

The `AccessControl.hcm` file contains module-specific access control actions for most module commands. However, a few module commands are governed by DesignSync or ProjectSync access controls:

Module Actions Governed by access controls

To control access to this module operation...	Customize this access control...
`addhref`	`Checkin` (a DesignSync access control) and `Addhref` See Creating a New Version of a Module and Access Controls for addhref for details.
`hcm addlogin`	`Addlogin` See Access Controls for addlogin for

	details.
`exportmod`	`Exportmodule` See Access Controls for Export/Import Operations.
`freezemod`	`Freezemodule` See Access Controls for Export/Import Operations.
`get` (for use with legacy modules)	`Checkout` (a DesignSync access control) See Access Controls for Checking Out for details.
`importmod`	`Importmodule` See Access Controls for Export/Import Operations.
`lock`	`Lock` See Locking Module Data for details.
`migratetag`	`HcmUpgrade` and `Mkmod` See Access Controls for upgrade and Access Controls for mkmod for details.
`mkedge`	`Checkin` (a DesignSync access control) and `Mkedge` See Creating a New Version of a Module and Access Controls for mkedge for details.
`mkmod`	`Mkmod` See Access Controls for mkmod for details.
`mvmember`	`Checkin` (a DesignSync access control) and `MemberRename` See Creating a New Version of a Module and Access Controls for mvmember for details.
mvmod	`MoveMod` on both servers and `MkMod` on the destination server. See Access Controls for Export/Import

	Operations for details.
`hcm put` (for use with legacy modules)	`Put` See Access Controls for put for details.
`reconnectmod`	`Reconnectmodule` See Access Controls for reconnectmod.
`hcm release` (for use with legacy modules)	`Release` See Access Controls for release for details.
`remove`	`Checkin` (a DesignSync access control) and `MemberRemove` See Creating a New Version of a Module and Access Controls for remove for details.
`hcm rmalias` (for use with legacy modules)	`Rmalias` See Access Controls for rmalias for details.
`hcm rmconf` (for use with legacy modules)	`Rmconf` See Access Controls for rmconf for details.
`rmedge`	`Checkin` (a DesignSync access control) and `Rmedge` See Creating a New Version of a Module and Access Controls for rmedge.
`rmhref`	`Checkin` (a DesignSync access control) and `Rmhref` See Creating a New Version of a Module and Access Controls for rmhref for details.
`hcm rmlogin`	`Rmlogin` See Access Controls for rmlogin for details.
`rmmod`	`Delete` (a DesignSync access control) and `Rmmod` See Access Controls for Deleting and Access Controls for rmmod for details.

`rollback`	`Checkin` (a DesignSync access control) and `Rollback` See Creating a New Version of a Module and Access Controls for rollback for details.
`showconfs` (for use with legacy modules)	`BrowseServer` (a DesignSync access control). See Access Controls for Browsing the Server for details.
`showhrefs`	`BrowseServer` (a DesignSync access control). See Access Controls for Browsing the Server for details.
`hcm showlogins`	`Showlogins` See Access Controls for hcm showlogins for details.
`showmods`	`BrowseServer` (a DesignSync access control). See Access Controls for Browsing the Server for details.
`showstatus`	`BrowseServer` (a DesignSync access control). See Access Controls for Browsing the Server for details.
`unfreezemod`	`Freezemodule` See Access Controls for Export/Import Operations.
`upgrade`	`HcmUpgrade` See Access Controls for upgrade for details.
view Commands	`BrowseServer` (a DesignSync access control). `Checkout` (a DesignSync access control), and `ModuleView`. See Access Controls for Browsing the Server, Access Controls for Checking Out, and Access Controls for Module

	Views.

Related Topics

ProjectSync Action Definitions

DesignSync Action Definitions

Setting Up Access Controls

User Authentication Action Definitions

Access Controls for Module Views

The default access control file, `$SYNC_DIR/share/AC_Components/AccessControl.hcm`, defines the action `ModuleView`, to control the maintenance (creating, deleting, viewing, and changing) of module view definitions.

```
access define ModuleView (<Object> <ViewName> <Action>)
```

where:

<Object> is the path to the object on the server. .

<`ViewName`> the name of the view.

<`Action`> is the permitted actions. If this value is omitted in the access control, the permissions granted or denied apply to all possible actions. the Possible values are

- read- grants/denies permission to read the view definition.
- add - grants/denies permission to create a view.
- replace- grants/denies permission to modify the view definition..
- delete - grants/denies permission to delete a view.

Note: You may also need to make sure the user has BrowseServer access for the module. You can also use the Checkout access control to grant/deny populate access for specific module views.

Related Topics

Setting Up Access Controls

Using Access Commands

Access Control Scripting

Access Controls for Export/Import Operations

The default access control file, `$SYNC_DIR/share/AC_Components/AccessControl.hcm`, defines a set of actions to control module export, import, move, and freeze/unfreeze actions.

Export/Import Actions Group

In order to grant or deny access to the entire set of Export/Import actions in a single access control, DesignSync provides a predefined access group, `HcmExportImportActions`. This is part of a set of module administration actions, `HcmAdministrationActions`. For more information on predefined access groups for modules, see Access Controls for Groups of Commands.

ExportMod

The action `ExportMod`, controls whether a user can export a module to a compressed file to move to a different server (using the `exportmod` command). The access control definition is:

```
access define ExportModule {Object}
```

where

- `<Object>` is the module URL

By default, access to `ExportMod` is denied to everyone.

```
access deny ExportModule everyone -because "Default access
controls disallow module export.\nContact your Synchronicity
tool administrator to enable this capability."
```

Additional Access Controls Required

The user will require BrowseServer access for the module.

ImportMod

The action `ImportMod`, controls whether a user can export a module to a compressed file to move to a different server (using the `importmod` command). The access control definition is:

```
access define ImportModule {Object}
```

where

- `<Object>` is the module URL

By default, access to `ImportModule` is denied to everyone.

```
access deny ImportModule everyone -because "Default access
controls disallow module import.\nContact your Synchronicity
tool administrator to enable this capability."
```

Additional Access Controls Required

The user will require BrowseServer access for the module.

MoveMod

The action `MoveModule`, controls whether a user can move a module to a different virtual location (using the `mvmod` command). The access control definition is:

```
access define MoveModule {Object}
```

where

- `<Object>` is the module URL

By default, access to `MoveModule` is denied to everyone.

```
access deny MoveModule everyone -because "Default access
controls disallow module moves.\nContact your Synchronicity tool
administrator to enable this capability."
```

Additional Access Controls Required

The user will require MkMod access for the new module location.

Enterprise Access Map

When access control is delegated, you can enable the `MoveModule` access map. A sample version of the access map is provided, commented out, in the custom DesignSync access control file for the server, `$SYNC_CUSTOM_DIR/servers/<`*`serverName`*`>/<`*`portNumber`*`>/share/AccessControl`.

```
access map MoveModule {
    # verify create/modify

    lappend checkmasks [list create $Object]

    lappend checkmasks [list modify $Object]

    return [list "masks" $checkmasks]

}
```

FreezeMod

The action `FreezeModule` controls whether a user can move a module to a different virtual location (using the `mvmod` command). The access control definition is:

```
access define FreezeModule {Object Action}
```

where

- `<Object>` is the module URL
- <`Action`> is the permitted actions. If this value is omitted in the access control, the permissions granted or denied apply to all possible actions. the Possible values are
    - FREEZE- grants/denies freeze a module, preventing any modifications from being made to the module.
    - UNFREEZE - grants/denies permission to unfreeze a module, allowing it to be used normally.

By default, access to `FreezeModule` is denied to everyone.

```
access deny FreezeModule everyone -because "Default access
controls disallow freezing and unfreezing of modules.\nContact
your Synchronicity tool administrator to enable this
capability."
```

Additional Access Controls Required

The user will require BrowseServer access for the module.

Enterprise Access Map

When access control is delegated, you can enable the `FreezeModule` access map. A sample version of the access map is provided, commented out, in the custom DesignSync access control file for the server, `$SYNC_CUSTOM_DIR/servers/<`*`serverName`*`>/<`*`portNumber`*`>/share/AccessControl.`

```
access map FreezeModule {

    if { $Action == "FREEZE" } {

        lappend checkmasks [list freeze $Object]

    } else {

        lappend checkmasks [list thaw $Object]

    }

    return [list "masks" $checkmasks]

}
```

Access Controls for Adding Hierarchical References

DesignSync addhref access is controlled by the Addhref access control. A successful `addhref` operation results in a new version of the module. Consequently, `Checkin` access for the module is the first access control checked, to determine whether subsequent access checks are necessary for the href. See Creating a New Version of a Module for details.

If `Checkin` access for the module is allowed, then it is not necessary to check `Addhref` access. Similarly, if `Checkin` access for the module is denied, then it is not necessary to check `Addhref` access.

If `Checkin` access for the module is declined, then `Addhref` access is checked. If `Addhref` access is allowed, that will result in a new version of the module.

DesignSync Access Controls.

Enterprise Design Access Map.

DesignSync Access Controls

The default access control file, `$SYNC_DIR/share/AC_Components/AccessControl.hcm`, defines the action `Addhref`, to control whether a user can add a hierarchical reference to a module version (using the `addhref` command):

```
access define Addhref <Object> <ToTarget> <Relpath>
```

where:

- `<Object>` is the string representation of the URL of the from-target to which the href is being added.
- `<ToTarget>` is the string representation of the URL of the to-target of the href
- `<Relpath>` is the relative path of the href being added

A successful `addhref` operation results in a new version of the module. Consequently, `Checkin` access for the module is the first access control checked, to determine whether subsequent access checks are necessary for the href. See Creating a New Version of a Module for details.

If `Checkin` access for the module is allowed, then it is not necessary to check `Addhref` access. Similarly, if `Checkin` access for the module is denied, then it is not necessary to check `Addhref` access.

If `Checkin` access for the module is declined, then `Addhref` access is checked. If `Addhref` access is allowed, that will result in a new version of the module.

Examples

```
access init {
   set ProjectLeads {ProjLead1 ProjLead2}
   set TeamMembers {ProjLead1 ProjLead2 john jane}
   set Team1 {ProjLead1 john}
   set Team2 {ProjLead2 jane}
   set admin { syncmgr }
}

# # Restrict adding and removing hrefs to only project leaders
#
access allow {Addhref Rmhref} only users $ProjectLeads \
      -because "only Project leaders can add or remove hrefs"

# # In this scenario we wish to allow individual users to
# # be able to add hrefs to modules with selectors that indicate they
# # are experimental branches, but not other branches.
# # Using a naming convention, all experimental
# # branches are identified by the selectors starting with 'Expt'
# # Project leaders can change (addhref, rmhref) any configuration.
#
access filter Addhref {
# get the selector extention
  set selector ""
  set clist [split $Object ";@"]
  # get the first part of the selector, which is the
  # part we will check.
  if { [llength $clist] > 1 } { set selector [lindex $clist 1] }
    if { [string match Expt* $selector] || \
       [lsearch $ProjectLeads $user] >= 0 } {
         # ok allowed
         return UNKNOWN
    }

    # not Expt* and not project lead
    return "Only Project Lead allowed to add hrefs on this branch"
}
```

Enterprise Design Access Map

When Enterprise Design access map is enabled for the server, the addhref access map is automatically available. The access map is located in the default DesignSync access control file, `$SYNC_DIR/share/AC_Components/AccessControl.ds.`

Tip: The checkin access map has the ability to process commands coming in from addhref or rmhref (initiated through edithref, addhref, or rmhref), but DesignSync recommends that you leave the addhref and rmhref maps active to process those commands, rather than using the Checkin access map.

```
access map Addhref {

    # verify checkin, fromconnect and toconnect access

    lappend checkmasks [list revise $Object]

    lappend checkmasks [list fromconnect $Object]

    lappend checkmasks [list toconnect $ToTarget]

    return [list "masks" $checkmasks]
```

Related Topics

Setting Up Access Controls

Using Access Commands

Access Control Scripting

Access Controls for rmhref

Enterprise Design Access Maps

Access Controls for Adding Logins

The default access control file, `$SYNC_DIR/share/AC_Components/AccessControl.hcm`, defines the action `Addlogin`, to control storing a server login (using the `addlogin` command):

```
access define Addlogin <ToTarget> <FromUser> <ToUser> IsSelf>
```

Where:

- `<ToTarget>` is the URL of the referenced server.

  Specify `ALLTARGETS` to indicate that the `-toalltargets` option was specified and that the login being stored will apply to all referenced servers.

- `<FromUser>` is the username for which this login is being stored.

  Specify `ALLUSERS` to indicate that the `-fromallusers` option was specified and that the login being stored will apply to all users on the server.

- `<ToUser>` is the username of the login on the referenced server.
- `<IsSelf>` is `yes` if the user performing the `addlogin` operation is the same as `<FromUser>` and `no` otherwise.

Examples

```
access init {
   set ProjectLeads {ProjLead1 ProjLead2}
   set TeamMembers {ProjLead1 ProjLead2 john jane}
   set Team1 {ProjLead1 john}
   set Team2 {ProjLead2 jane}
   set admin { syncmgr }
}

# # We would like users to be able to add/remove/view login
# # information only for themselves, and allow admins to be
# # able to add/remove/view login information for all users.
#
# # This can be done by denying the action for 'everyone'
# # and then selectively allowing for 'admin', and for 'self'.
#
```

```
access deny Addlogin everyone
access allow Addlogin only users $admin
access allow Addlogin everyone when IsSelf yes
access deny Rmlogin everyone
access allow Rmlogin only users $admin
access allow Rmlogin everyone when IsSelf yes
access deny Showlogins everyone
access allow Showlogins only users $admin
access allow Showlogins everyone when IsSelf yes
```

Related Topics

Setting Up Access Controls

Using Access Commands

Access Control Scripting

Access Controls for Locking a Module Branch

The `Lock`, access control governs locking a module branch, using the `lock` command.

Defining and Using the Lock Access Control

The default DesignSync access control file, `$SYNC_DIR/share/AC_Components/AccessControl.ds`, defines the action:

```
access define Lock <Object>
```

where `<Object>` is the URL of the module branch.

Examples

```
access init {

   set ProjectLeads {ProjLead1 ProjLead2}
   set TeamMembers {ProjLead1 ProjLead2 john jane}
   set Team1 {ProjLead1 john}
   set Team2 {ProjLead2 jane}
   set admin { syncmgr }
}

# # Lock of module members, using the 'populate -lock' operation,
# # is controlled by the MemberCheckout access control. Locking of
# # a whole branch is controlled by the Lock access control.
#
```

```
# # Allow users to lock their private modules, but only ProjectLeads
# # to lock other module branches.
access deny Lock everyone
access allow Lock only users $ProjectLeads
access filter Lock {
   if {[string match sync:///Modules/users/$user/* $Object]} {
      return ALLOW
   }
   return UNKNOWN
}
```

Defining the Access Map

When access control is delegated, you can enable the `Lock` access map. A sample version of the access map is provided, commented out, in the custom DesignSync access control file for the server, `$SYNC_CUSTOM_DIR/servers/<serverName>/<portNumber>/share/AccessControl`.

```
access map Lock {

    # verify lock

    lappend checkmasks [list lock $Object]

    return [list "masks" $checkmasks]

}
```

Related Topics

Setting Up Access Controls

Using Access Commands

Access Control Scripting

Access Controls for Creating Merge Edges

The `Mkedge` access control allows the creation of merge edges (using the `mkedge` command) when merging modules across branches:

Defining and Using the Access Control

TheMkedge access control is located in the default access control file, `$SYNC_DIR/share/AC_Components/AccessControl.hcm:`

```
access define Mkedge {<Object> <VersionFrom>}
```

where `<Object>` is the full module URL to which the merge edge is being added.

<versionFrom> is the numeric version from which the merge edge is being added.

Examples

```
access init {
   set ProjectLeads {ProjLead1 ProjLead2}
   set TeamMembers {ProjLead1 ProjLead2 john jane}
   set Team1 {ProjLead1 john}
   set Team2 {ProjLead2 jane}
   set admin { syncmgr }
}
 # Mkedge:
# Arguments:
#    Object == string representation of the full module version URL
#            to which the edge is being added
#    VersionFrom == string representation of the numeric version
#                from which the edge is being added
access define Mkedge {Object VersionFrom}
access allow Mkedge only users $ProjectLeads \
      -because "only Project leaders can create merge edges"
```

Defining the Enterprise Access Map

When access control is delegated, you can enable the `Mkedge` access map. A sample version of the access map is provided, commented out, in the custom DesignSync access control file for the server, `$SYNC_CUSTOM_DIR/servers/<serverName>/<portNumber>/share/AccessControl`.

```
access map Mkedge {

    lappend checkmasks [list modify $Object]

    return [list "masks" $checkmasks]

}
```

Related Topics

Setting Up Access Controls

Using Access Commands

Access Control Scripting

Access Controls for Creating Modules

The default access control file, `$SYNC_DIR/share/AC_Components/AccessControl.hcm`, defines the action `Mkmod`, to control the creation of a new module (using the `mkmod` command):

```
access define Mkmod <Object>
```

where `<Object>` is the URL of the module to be created.

Examples

```
access init {
   set ProjectLeads {ProjLead1 ProjLead2}
   set TeamMembers {ProjLead1 ProjLead2 john jane}
   set Team1 {ProjLead1 john}
   set Team2 {ProjLead2 jane}
   set admin { syncmgr }
}

# # Prevent all uses except the project leads from creating new modules
# # outside of the sync:///Modules/users/$user category.
#
access filter Mkmod {
  set parent [url container $Object]
  if { ![string match sync:///Modules/users/$user/* $parent] && \
       ("$parent" != "sync:///Modules/users/$user") } {
    if {[lsearch -exact $ProjectLeads $user] == -1} {
      return "Not allowed to create a module outside of the users/$user category"
    }
  }
  return UNKNOWN
}
```

Related Topics

Setting Up Access Controls

Using Access Commands

Access Control Scripting

Access Controls for Moving Module Members

The default access control file, `$SYNC_DIR/share/AC_Components/AccessControl.hcm`, defines the action `MemberRename`, to control changing the " natural path" of a module member (using the `mvmember` command):

```
access define MemberRename <Object> <NaturalPath>
<NewNaturalPath> <Type>
```

Where:

- `<Object>` is the module branch URL
- `<NaturalPath>` is the original value of the module member's natural path
- `<NewNaturalPath>` is the new value of the module member's natural path
- `<Type>` is one of the following object types: FILE, FOLDER

DesignSync only checks the access controls if the command accesses the server. Thus, if the only module members being renamed are workspace items that have never been checked into the module, the `mvmember` command does not access the server, and the access controls are not checked.

If at least one of the module members being renamed is already in a version of the module, the `mvmember` operation will result in a new version of the module. Consequently, `Checkin` access for the module is the first access control checked, to determine whether subsequent access checks are necessary at the member level. See Creating a New Version of a Module for details.

If `Checkin` access for the module is allowed, it is not necessary to check individual `MemberRename` access. Similarly, if `Checkin` access for the module is denied, it is not necessary to check individual `MemberRename` access.

If `Checkin` access for the module is declined, `MemberRename` access is checked for each member being renamed (by the `mvmember` command). If `MemberRename` access is allowed for at least one module member, a new version of the module is created.

Examples

```
access init {
   set ProjectLeads {ProjLead1 ProjLead2}
   set TeamMembers {ProjLead1 ProjLead2 john jane}
   set Team1 {ProjLead1 john}
   set Team2 {ProjLead2 jane}
   set admin { syncmgr }
}

# # For the commands that creates a new module version:
```

```
# #     ci, addhref/rmhref and remove/mvmember,
# # a check is first made for Checkin access, and the
# # check for the sub-command only performed if the
# # Checkin access returns a value of "decline"
#
# # So, first "decline" the Checkin access for everyone,
# # so that the the member check is called for all users
#
access decline Checkin everyone

# # Now can restrict the Team2 users to only checking in
# # objects with Natural Paths under /src area
#
access deny MemberCheckin only users $Team2
access allow MemberCheckin only users $Team2 \
        when NaturalPath /src/*

# # Similarly, we can restrict Team2 to not removing anything
# # outside their area
access deny MemberRemove only users $Team2
access allow MemberRemove only users $Team2 \
        when NaturalPath /src/*

# # And can restrict Team2 to moving things where BOTH the
# # paths are in their area.
access deny MemberRename only users $Team2
access allow MemberRename only users $Team2 \
        when NaturalPath /src/* \
        when NewNaturalPath /src/*

# # And can restrict adding hrefs to only project leaders
#
access allow {Addhref Rmhref} only users $ProjectLeads \
       -because "only Project leaders can add or remove hrefs"
```

Related Topics

Setting Up Access Controls

Using Access Commands

Access Control Scripting

Access Controls for Removing Module Members

The default access control file, `$SYNC_DIR/share/AC_Components/AccessControl.hcm`, defines the action `MemberRemove`, to control removing a module member from a module version (using the `remove` command):

```
access define MemberRemove <Object> <NaturalPath>
```

where:

- `<Object>` is the module branch URL
- `<NaturalPath>` is the natural path of the module member

If the only module members being removed are workspace items that have never been checked into the module, (they were only added locally with the `add` command) then access controls are not checked, because the `remove` command does not access the server in this case.

If at least one of the module members being removed is already in a version of the module, the `remove` operation will result in a new version of the module. Consequently, `Checkin` access for the module is the first access control checked, to determine whether subsequent access checks are necessary at the member level. See Creating a New Version of a Module for details.

If `Checkin` access for the module is allowed, then it is not necessary to check individual `MemberRemove` access. Similarly, if `Checkin` access for the module is denied, then it is not necessary to check individual `MemberRemove` access.

If `Checkin` access for the module is declined, then `MemberRemove` access is checked for each member that is being removed (by the `remove` command). If `MemberRemove` access is allowed for at least one module member, that will result in a new version of the module.

Examples

```
access init {
   set ProjectLeads {ProjLead1 ProjLead2}
   set TeamMembers {ProjLead1 ProjLead2 john jane}
   set Team1 {ProjLead1 john}
   set Team2 {ProjLead2 jane}
   set admin { syncmgr }
}

# # For the commands that creates a new module version:
# #     ci, addhref/rmhref and remove/mvmember,
# # a check is first made for Checkin access, and the
# # check for the sub-command only performed if the
# # Checkin access returns a value of "decline"
#
# # So, first "decline" the Checkin access for everyone,
# # so that the the member check is called for all users
#
access decline Checkin everyone

# # Now can restrict the Team2 users to only checking in
# # objects with Natural Paths under /src area
#
access deny MemberCheckin only users $Team2
access allow MemberCheckin only users $Team2 \
        when NaturalPath /src/*

# # Similarly, we can restrict Team2 to not removing anything
# # outside their area
access deny MemberRemove only users $Team2
access allow MemberRemove only users $Team2 \
        when NaturalPath /src/*

# # And can restrict Team2 to moving things where BOTH the
# # paths are in their area.
access deny MemberRename only users $Team2
access allow MemberRename only users $Team2 \
        when NaturalPath /src/* \
        when NewNaturalPath /src/*

# # And can restrict adding hrefs to only project leaders
#
access allow {Addhref Rmhref} only users $ProjectLeads \
       -because "only Project leaders can add or remove hrefs"
```

Related Topics

Setting Up Access Controls

Using Access Commands

Access Control Scripting

Access Controls for Removing Merge Edges

The `Rmedge` access control allows the removal of merge edges (using the `rmedge` command) when merging modules across branches:

Defining and Using the Access Control

The `RmEdge` access control is located in the default access control file, `$SYNC_DIR/share/AC_Components/AccessControl.hcm`:

```
access define Rmedge {<Object> <VersionFrom>}
```

where `<Object>` is the full module URL to which the merge edge is being removed.

<versionFrom> is the numeric version from which the merge edge is being removed.

Examples

```
access init {
   set ProjectLeads {ProjLead1 ProjLead2}
   set TeamMembers {ProjLead1 ProjLead2 john jane}
   set Team1 {ProjLead1 john}
   set Team2 {ProjLead2 jane}
   set admin { syncmgr }
}
 # Rmedge:
# Arguments:
#    Object == string representation of the full module version URL
#            to which the edge was added
#    VersionFrom == string representation of the numeric version
#                from which the edge was added
access define Rmedge {Object VersionFrom}
access allow Rmedge only users $ProjectLeads \
       -because "only Project leaders can remove merge edges"
```

Defining the Enterprise Access Map

When access control is delegated, you can enable the `Rmedge` access map. A sample version of the access map is provided, commented out, in the custom DesignSync access control file for the server, `$SYNC_CUSTOM_DIR/servers/<`*`serverName`*`>/<`*`portNumber`*`>/share/AccessControl`.

```
access map Rmedge {

     lappend checkmasks [list modify $Object]

     return [list "masks" $checkmasks]

}
```

Related Topics

Setting Up Access Controls

Using Access Commands

Access Control Scripting

Access Controls for Removing Hierarchical References

DesignSync rmhref access is controlled by the Rmhref access control. A successful `rmhref` operation results in a new version of the module. Consequently, `Checkin` access for the module is the first access control checked, to determine whether subsequent access checks are necessary for the href. See Creating a New Version of a Module for details.

If `Checkin` access for the module is allowed, then it is not necessary to check `Rmhref` access. Similarly, if `Checkin` access for the module is denied, then it is not necessary to check `Rmref` access.

If `Checkin` access for the module is declined, then `Rmhref` access is checked. If `Rmhref` access is allowed, that will result in a new version of the module.

DesignSync Access Controls.

Enterprise Design Access Map.

DesignSync Access Controls

The default access control file, `$SYNC_DIR/share/AC_Components/AccessControl.hcm`, defines the action `Rmhref` to control removing a hierarchical reference between modules (using the `rmhref` command):

```
access define Rmhref <Object> <ToTarget> <Relpath>
```

where:

- `<Object>` is the string representation of the URL of the from-target from which the href is to be removed.
- `<ToTarget>` is the string representation of the URL pattern of the to-target of the href(s) to be removed.
- `<Relpath>` is the relative path (pattern) of the href(s) to be removed.

Examples

```
access init {
   set ProjectLeads {ProjLead1 ProjLead2}
   set TeamMembers {ProjLead1 ProjLead2 john jane}
   set Team1 {ProjLead1 john}
   set Team2 {ProjLead2 jane}
   set admin { syncmgr }
}

# # Restrict adding and removing hrefs to only project leaders
#
access allow {Addhref Rmhref} only users $ProjectLeads \
      -because "only Project leaders can add or remove hrefs"
```

Enterprise Design Access Map

When Enterprise Design access map is enabled for the server, the rmhref access map is automatically available. The access map is located in the default DesignSync access control file, `$SYNC_DIR/share/AC_Components/AccessControl.ds.`

Tip: The checkin access map has the ability to process commands coming in from addhref or rmhref (initiated through edithref, addhref, or rmhref), but DesignSync recommends that you leave the addhref and rmhref maps active to process those commands, rather than using the Checkin access map.

```
access map Rmhref {

    # verify checkin, fromdisconnect and todisconnect access


    lappend checkmasks [list revise $Object]

    lappend checkmasks [list fromdisconnect $Object]

    lappend checkmasks [list todisconnect $ToTarget]

    return [list "masks" $checkmasks]
```

Related Topics

Setting Up Access Controls

Using Access Commands

Access Control Scripting

Access Controls for Removing Logins

The default access control file, `$SYNC_DIR/share/AC_Components/AccessControl.hcm`, defines the action `Rmlogin` to control removing a login stored on the server (using the `hcm rmlogin` command):

```
access define Rmlogin <ToTarget> <FromUser> <IsSelf>
```

`where:`

- `<ToTarget>` is the URL of the referenced server.

  Specify the `ALLTARGETS` value to indicate that the `-toalltargets` option was specified and the stored login being removed applies to all referenced servers.

- `<FromUser>` is the username for which this stored login is being removed.

  Specify the `ALLUSERS` value to indicate that the `-fromallusers` option was specified and the login being removed applies to all users on this server.

- `<IsSelf>` is `yes` if the user performing the `hcm rmlogin` operation is the same as `<FromUser>` and `no` otherwise.

Examples

```
access init {
   set ProjectLeads {ProjLead1 ProjLead2}
   set TeamMembers {ProjLead1 ProjLead2 john jane}
   set Team1 {ProjLead1 john}
   set Team2 {ProjLead2 jane}
   set admin { syncmgr }
}

# # We would like users to be able to add/remove/view login
# # information only for themselves, and allow admins to be
# # able to add/remove/view login information for all users.
#
# # This can be done by denying the action for 'everyone'
# # and then selectively allowing for 'admin', and for 'self'.
#
access deny Addlogin everyone
access allow Addlogin only users $admin
access allow Addlogin everyone when IsSelf yes
access deny Rmlogin everyone
access allow Rmlogin only users $admin
```

access allow Rmlogin everyone when IsSelf yes
access deny Showlogins everyone
access allow Showlogins only users $admin
access allow Showlogins everyone when IsSelf yes

Related Topics

Setting Up Access Controls

Using Access Commands

Access Control Scripting

Access Controls for Removing a Module

The access control `Rmmod` controls removing a module and its configurations from a server (using the `rmmod` command):

Defining and Using the Access Control

The `Rmmod` action is defined in the default access control file, `$SYNC_DIR/share/AC_Components/AccessControl.hcm`,

```
access define Rmmod <Object>
```

where `<Object>` is the URL of the module that is being removed.

This command does not obey ProjectSync's `EditNote` and `DeleteNote` access controls (see Access Controls for Notes and Note Types). You should take this behavior into consideration when granting access to this command.

Examples

```
access init {
    set ProjectLeads {ProjLead1 ProjLead2}
    set TeamMembers {ProjLead1 ProjLead2 john jane}
    set Team1 {ProjLead1 john}
    set Team2 {ProjLead2 jane}
    set admin { syncmgr }
}

# # Restrict module removal to project leaders.
#
access allow Rmmod only users $ProjectLeads \
        -because "only Project leaders can delete modules"
```

Defining the Access Map

When access control is delegated, you can enable the Rmod access map. A sample version of the access map is provided, commented out, in the custom DesignSync access control file for the server, `$SYNC_CUSTOM_DIR/servers/<serverName>/<portNumber>/share/AccessControl`.

```
access map Rmmod {

    # verify delete

    lappend checkmasks [list delete $Object]

    return [list "masks" $checkmasks]
```

Related Topics

Setting Up Access Controls

Using Access Commands

Access Control Scripting

Access Controls for Reconnecting a Module

The reconnectmod command does not require a unique access control. In order to run reconnectmod, which is used to reconnect hierarchical references to a module that has changed locations, you need the following access:

- BrowseServer access for the module.
- Checkin access to the module.

Related Topics

Access Controls for Export/Import Operations

Access Controls for Rolling Back a Module

The default access control file, `$SYNC_DIR/share/AC_Components/AccessControl.hcm`, defines the action `Rollback` to control reverting a module (or rolling the module back) to a previous module version (using the `rollback` command):

```
access define Rollback <Object>
```

where `<Object>` is the URL of the module branch.

A successful `rollback` operation results in a previously existing module being set to the Trunk:Latest version. The first access control checked is `Checkin`, to determine whether subsequent access checks are necessary. See Creating a New Version of a Module for details.

If `Checkin` access for the module is denied, then it is not necessary to check `Rollback` access, the command fails immediately.

If `Checkin` access for the module is allowed, then `Rollback` access is checked. If `Rollback` access is allowed, the module rollback succeeds.

Examples

```
access init {
   set ProjectLeads {ProjLead1 ProjLead2}
   set TeamMembers {ProjLead1 ProjLead2 john jane}
   set Team1 {ProjLead1 john}
   set Team2 {ProjLead2 jane}
   set admin { syncmgr }
}
```

```
# # Example for 'rollback' command.
# #
# # Restrict rollback to Project Leads only.
# # Remember that rollback is denied to all users by default.
access allow Rollback only users $ProjectLeads \
-because "Only ProjectLeaders can rollback modules "
```

Defining Enterprise Access Map

When access control is delgated, you can enable the `Rollback` access map. A sample version of the access map is provided, commented out, in the custom DesignSync access control file for the server, `$SYNC_CUSTOM_DIR/servers/<serverName>/<portNumber>/share/AccessControl`.

Related Topics

Rolling Back a Module

Access Controls for Showing Logins

The default access control file, `$SYNC_DIR/share/AC_Components/AccessControl.hcm`, defines the action `Showlogins` to control the display of logins stored on a server (using the `hcm showlogins` command):

```
access define Showlogins <FromUser> <IsSelf>
```

where:

- `<FromUser>` is used to restrict the set of stored logins displayed. This parameter can take the following values to determine what the `showlogins` operation displays:

  `" "` : Displays all stored logins.

  `ALLUSERS` : Displays only logins stored using the `-fromallusers` option.

  `<username>` : Displays only logins stored using the `-fromuser <username>` option.

- `<IsSelf>` is `yes` if the user performing the `showlogins` operation is the same as `<FromUser>` and `no` otherwise.

Examples

```
access init {
   set ProjectLeads {ProjLead1 ProjLead2}
   set TeamMembers {ProjLead1 ProjLead2 john jane}
   set Team1 {ProjLead1 john}
   set Team2 {ProjLead2 jane}
   set admin { syncmgr }
}

# # We would like users to be able to add/remove/view login
# # information only for themselves, and allow admins to be
# # able to add/remove/view login information for all users.
#
# # This can be done by denying the action for 'everyone'
# # and then selectively allowing for 'admin', and for 'self'.
#
access deny Addlogin everyone
access allow Addlogin only users $admin
access allow Addlogin everyone when IsSelf yes
access deny Rmlogin everyone
access allow Rmlogin only users $admin
access allow Rmlogin everyone when IsSelf yes
access deny Showlogins everyone
access allow Showlogins only users $admin
access allow Showlogins everyone when IsSelf yes
```

Related Topics

Setting Up Access Controls

Using Access Commands

Access Control Scripting

Access Controls for Upgrading to Modules

The default access control file, `$SYNC_DIR/share/AC_Components/AccessControl.hcm`, defines the action `HcmUpgrade` to control upgrading a DesignSync vault hierarchy or a legacy module (pre-DesignSync 5.x) to the current module format (using the `upgrade` command):

```
access define Upgrade <Object> <ModuleURL>
```

where:

- `<Object>` is the URL of the object to be upgraded. The object to be upgraded is a DesignSync vault folder URL. That URL can be the location of a legacy module.
- `<ModuleURL>` is the URL of the module to be created.

Example

```
access init {
set admin { syncmgr }
}

# Allow only admins to upgrade pre-5.0 data to 5.0 modules
access allow Upgrade only users $admin
```

Related Topics

Setting Up Access Controls

Using Access Commands

Access Control Scripting

HCM Action Definitions for legacy modules

Access Controls for hcm put

The `hcm put` command, and its corresponding access control, only applies to legacy modules that meet all three of these conditions:

- The legacy module was created on the server prior to the server's upgrade use the new modules format.
- The module has not been upgraded to the new modules format. (The `upgrade` command was not run on the module.)
- The module is being operated on by a DesignSync 4.X client.

The default access control file, `$SYNC_DIR/share/AC_Components/AccessControl.hcm`, defines the action `Put`, to control checking in a configuration (using the `put` command):

```
access define Put <Object>
```

where `<Object>` is the URL of the configuration to put.

Non-legacy modules use DesignSync's `ci` command to checkin data. See Creating a New Version of a Module for access controls pertaining to the checkin of non-legacy module data.

Example

```
# # Deny put access to everyone for the CPU and all of its
configurations.
#
# access filter Put {
#     if {[string match "*/Projects/CPU" $Object] || [string
match "*/Projects/CPU@*" $Object]} {
#         return "CPU can not be modified."
#     }
#     return UNKNOWN
# }
```

Related Topics

Setting Up Access Controls

Using Access Commands

Access Control Scripting

Access Controls for hcm release

The `hcm release` command, and its corresponding access control, only applies to legacy modules that meet all three of these conditions:

- The legacy module was created on the server prior to the server's upgrade to use the new modules format.
- The legacy module has not been upgraded. (The `upgrade` command was not run on the module.)
- The module is being operated on by a DesignSync 4.X client.

The default access control file, `$SYNC_DIR/share/AC_Components/AccessControl.hcm`, defines the action `Release`, to control creating a release configuration (using the `release` command):

```
access define Release <Object> <Name>
```

where:

- `<Object>` is the URL of the module being released.
- `<Name>` is the name of the release being created.

Non-legacy modules use DesignSync's `tag` command for configuration management. See Tagging a Module for access controls pertaining to the tagging of non-legacy module data.

Example

```
# access init {
#     set ProjectLeads {ProjLead1 ProjLead2}
# }
# # We would like to allow uses to create releases for their own
# # and for internal sharing purposes. But we wish to enforce a
policy
# # that reserved release names (all prefixed with 'Rel') can be
created
# # only by project leaders.
#
# access filter Release {
#
#   if { [string match Rel* $Name] && \
#        [lsearch $ProjectLeads $user] == -1 } {
#     # Is a reserved release name, and user is not
#     # a project leader, so deny
#     return "Only project lead is allowed to create 'Rel'
releases"
#   }
#   return UNKNOWN
# }
```

Related Topics

Setting Up Access Controls

Using Access Commands

Access Control Scripting

Access Controls for hcm rmalias

The `hcm rmalias` command, and its corresponding access control, only applies to legacy modules that meet all three of these conditions:

- The legacy module was created on the server prior to the server's upgrade to DesignSync 5.x.
- The legacy module has not been upgraded. (The `upgrade` command was not run on the module.)
- The legacy module is being operated on by a DesignSync 4.X client.

The default access control file, `$SYNC_DIR/share/AC_Components/AccessControl.hcm`, defines the action `Rmalias`, to control removing an alias for a release (using the `rmalias` command):

```
access define Rmalias <Object>
```

where `<Object>` is the URL of the alias that is being removed.

This command does not obey ProjectSync's `EditNote` and `DeleteNote` access controls (see Access Controls for Notes and Note Types). Take this behavior into consideration when granting access to this command.

Non-legacy modules use DesignSync's `tag` command for configuration management. See Tagging a Module for access controls pertaining to the tagging of non-egacy module data.

Example

```
# access init {
#     set ProjectLeads {ProjLead1 ProjLead2}
# }
#
# #
# # Restrict object removal to project leaders.
#
# access allow {Rmconf Rmalias} only users $ProjectLeads \
#         -because "only Project leaders can delete
configurations and aliases"
```

Related Topics

Setting Up Access Controls

Using Access Commands

Access Control Scripting

Access Controls for hcm rmconf

The `hcm rmconf` command, and its corresponding access control, only applies to legacy modules that meet all three of these conditions:

- The legacy module was created on the server prior to the server's upgrade to use the new module format introduced in DesignSync 5.x.
- The legacy module has not been upgraded. (The `upgrade` command was not run on the module.)
- The legacy module is being operated on by a DesignSync 4.X client.

The default access control file, `$SYNC_DIR/share/AC_Components/AccessControl.hcm`, defines the action `Rmconf`, to control removing a configuration from a server (using the `rmconf` command):

```
access define Rmconf <Object>
```

where `<Object>` is the URL of the module configuration that is being removed.

This command does not obey ProjectSync's `EditNote` and `DeleteNote` access controls (see Access Controls for Notes and Note Types). Take this behavior into consideration when granting access to this command.

Non-legacy modules use DesignSync's `tag` command for configuration management. See Tagging a Module for access controls pertaining to the tagging of Non-legacy module data.

Example

```
# access init {
#     set ProjectLeads {ProjLead1 ProjLead2}
# }
#
# #
# # Restrict object removal to project leaders.
#
# access allow {Rmconf Rmalias} only users $ProjectLeads \
#         -because "only Project leaders can delete
configurations and aliases"
```

Related Topics

Setting Up Access Controls

Using Access Commands

Access Control Scripting

Access Controls for Groups of Commands

You might want to grant or deny complete access to module actions for some set of users. Variables defined in the default module access control file, `$SYNC_DIR/share/AC_Components/AccessControl.hcm`, make it more convenient to set the same access rights on module actions.

You also can create your own variables for sets of commands in your custom `AccessControl` file.

Predefined Variables

The default `AccessControl.hcm` file defines these variables to control module actions:

```
set HcmWriteActions "Mkmod Addhref Rmhref Release Rmmod Rmconf
Rmalias Put MemberCheckin MemberRemove MemberRename MemberUnlock
MemberLock MemberSwitchLocker MemberTag Mkedge Rmedge"

set HcmReadActions "MemberCheckout"

set HCMLoginActions "Addlogin Rmlogin Showlogins"

set HCMCmdActions "Rmmod Mkmod HcmUpgrade"
set HCMActions "$HCMCmdActions $HCMLoginActions"

set HcmExportImportActions "ExportModule ImportModule
FreezeModule MoveModule"

set HcmAdministrationActions "$HcmExportImportActions"
```

Note: The module groupings do not include general DesignSync actions that operate on module data. See Access Controls for Groups of Commands in the DesignSync section for the DesignSync command group variables.

For example, you might use the `LoginActions` variable to restrict the manipulation of ProjectSync accounts used by modules. To do so, in your custom `AccessControl` file, specify:

```
access allow $LoginActions only users syncmgr
```

Another variable, `$AllActions`, controls all ProjectSync actions, as well as all DesignSync and module actions that have an Object parameter (`$CmdActions)`. The `$AllActions` variable is defined in the `AccessControl.ps` file. See Access Controls for Groups of Commands in the ProjectSync section for details.

Custom-Defined Variables

In your custom `AccessControl` file, you can use the `set` command to create variables that represent your own groupings of access control actions.

For example, to create a variable that controls the ability to move and remove module members, you could specify:

```
set ModuleMoveActions {MemberRename MemberRemove}
```

You can then use the variable `$ModuleMoveActions` in your access control rules.

You also can create variables that include other variables. However, when declaring a list that requires variable substitution, you must enclose the list in quotation marks, not curly braces. (Curly braces prevent variable substitution.) For example:

```
set ModuleDesignerActions "$CmdActions $ModuleMoveActions Tag"
```

Related Topics

Access Control Scripting

Sample Access Controls

Setting Up Access Controls

Using Access Commands

Using DesignSync Access Controls with Modules

Browsing Modules on a Server

The `BrowseServer` action, defined in Access Controls for Browsing the Server, applies to all operations that access modules or module data on the server. Those operations include:

- `addbackref` command
- `contents` command
- `compare` command
- `module view` commands
- `mkedge` command
- `rollback` command
- `rmedge` command
- `showhrefs` command
- `showmods` command
- `showstatus` command
- `showlocks` command
- `vhistory` command
- `whereused` command

`BrowseServer` access can be used in two different ways.

- It can be set at on the entire Modules category, or on an individual category level to control access to actions involving all of the modules, or all the modules within a category. These actions allow activities that operate on the module as a unit, such as `showstatus`, which shows the status of the module. When an access control is set on a module or module category, you may need two access control to manage access; one to control access to the top level, and one to control access to the sublevels. See Accessing Modules for more information.
- It can be set on an individual module level to control access to actions involving the module objects, such as `contents` which examines the module contents.

Note: Because you can have different access levels for the Modules folder, the category folders, and individual modules, you may see situations where you can examine the module as a whole, but cannot see the contents of a module.

Accessing Modules

DesignSync provides browsing access at multiple levels, each able to be controlled with its own access control. Like directories, categories, including the Modules category itself, can be restricted either at the directory level, with lower levels controlled by different access controls, or recursively (also with the possibility of different access controls at lower levels.)

For example specifying an access controls that controls the access level at :

sync:///Modules/Category1

would enable or disable the ability to view what modules or categories were contained in Category1, but would have no effect on whether the modules or categories within Category1 were viewable.

sync:///Modules/Category1/*

would enable or disable the ability to view all the contents of Category1, recursively, but would not control whether you could see what modules or categories were contained within Category1.

Thus to deny access to both browse the modules or categories contained within Category1 as well as the contents of those modules and sub-categories, you would use two access controls:

```
access deny BrowseServer everyone when Object
sync:///Modules/Category1
access deny BrowseServer everyone when Object
sync:///Modules/Category1/*
```

You could then add back access, selectively, to the users and groups who need them.

```
access allow BrowseServer ProjGroup when Object
sync:///Modules/Category1/Module1
```

Examples

```
access init {
   set ProjectLeads {ProjLead1 ProjLead2}
   set TeamMembers {ProjLead1 ProjLead2 john jane}
   set Team1 {ProjLead1 john}
   set Team2 {ProjLead2 jane}
   set admin { syncmgr }
}

# # Restrict access to Team Members, to be able to view what
# # modules are available on this server.
```

```
#
access deny BrowseServer everyone when Object sync:///Modules/
access deny BrowseServer everyone when Object sync:///Modules/*
access allow BrowseServer only users $TeamMembers when Object sync:///Modules

# # Team1 works on Modules in category 'Proj1'. Restrict viewing of
# # 'Proj1' modules and retrieval of 'Proj1' hrefs to only members of Team1.
# # This will also limit the ability of non Team1 members to perform any
# # DesignSync actions on 'Proj1'.
#
access filter BrowseServer {
  if { $Object == "sync:///Modules/Proj1" || \
        1 == [string match sync:///Modules/Proj1/* $Object] } {
    if { [lsearch $Team1 $user] == -1 } {
      return "Only Proj1 team members are allowed access to Proj1 modules"
    }
  }
  return UNKNOWN
}
```

Related Topics

Setting Up Access Controls

Using Access Commands

Access Control Scripting

Fetching a Module

The populate command can be used to fetch the contents of an individual module, or the contents of an entire module hierarchy. The contents of a module (the individual module members) can be locked by using the -lock option to `populate`. The `lock` command is used to lock a branch of a module. For access controls pertaining to the `lock` command, see the topic Locking Module Data.

When a request to `populate` module data is made, Checkout access for the module version is the first access control checked, to determine whether subsequent access checks are necessary. The `Checkout`/`CheckoutLock` actions are defined in Access Controls for Checkout Out. The module version URL is used as the `Object` parameter for the Checkout action.

If Checkout/CheckoutLock access for the module version is allowed, then no additional access checks are necessary for that module version; DesignSync fetches the module version's member objects. Similarly, if Checkout access for the module version is denied, then no additional access checks are necessary for that module version; the attempt to fetch the module version fails.

Note: If CheckoutLock or CheckoutLockMember are denied, the non-locking fetch may still work.

However, if Checkout access for the module version is declined, then additional access checks are necessary; `MemberCheckoutLock/MemberCheckout` access is checked for each member object to be fetched.

The default access control file, `$SYNC_DIR/share/AC_Components/AccessControl.hcm`, defines the actions `MemberCheckout` and `MemberCheckoutLock.`

Using the Member Checkout Access Controls

Both the MemberCheckout and MemberCheckoutLock access controls are used to control DesignSync module member checkouts. Using the MemberCheckoutLock access control allows the administrator to finely control the ability to modify module members by locking an object on modification. This model is especially useful when you want to provide read access to DesignSync module members that are used by different development teams as part of their designs, but are not actually intended to be modified by those teams. The administrator can define an access control for the referencing teams that provides MemberCheckout access and an access for the owning team, who is responsible for the modifications to the objects, that includes the MemberCheckoutLock ability and other vault modification access, such as tag, Checkin, etc.

Note: If there is a conflict in permissions, for example, if you have the following two access controls defined:

```
access allow MemberCheckout everyone when Lock “yes”

access deny MemberCheckoutLock everyone
```

DesignSync enforces the more restrictive access control and denies all users the ability to perform a MemberCheckout with a lock.

MemberCheckout Access Control

```
access define MemberCheckout <Object NaturalPath Lock>
```

where:

- `<Object>` is the version URL of the module.
- `<NaturalPath>` is the natural path of the module member.

- `<Lock>` is `yes` if this is a "locking" populate, `no` if it is not a "locking" populate. See the `populate` command for the definition of a "locking" populate as it pertains to module data.

If `MemberCheckout` access is allowed for a member object, DesignSync fetches the member object.

Examples

```
access init {
   set ProjectLeads {ProjLead1 ProjLead2}
   set TeamMembers {ProjLead1 ProjLead2 john jane}
   set Team1 {ProjLead1 john}
   set Team2 {ProjLead2 jane}
   set admin { syncmgr }
}

# # To restrict the contents (members) of a module that
# # can be fetched, we must first make the Checkout
# # access control return a "decline" value, so that the
# # MemberCheckout control is then called for each member
#
# # Let Team2 only fetch the members in the src area
# # First, decline the overall checkout of the module
access decline Checkout only users $Team2

# # And then allow MemberCheckout only for the items we
# # want them to have access to
access deny MemberCheckout only users $Team2
access allow MemberCheckout only users $Team2 \
        when NaturalPath /src/*
```

MemberCheckoutLock Access Control

```
access define MemberCheckoutLock <Object NaturalPath>
```

where:

- `<Object>` is the version URL of the module.
- `<NaturalPath>` is the natural path of the module member

If `MemberCheckoutLock` access is allowed for a member object, DesignSync fetches and locks the member object as long as it is not denied by a `MemberCheckout` access control..

Examples

```
access init {
   set ProjectLeads {ProjLead1 ProjLead2}
   set TeamMembers {ProjLead1 ProjLead2 john jane}
   set Team1 {ProjLead1 john}
   set Team2 {ProjLead2 jane}
   set admin { syncmgr }
}

# # To restrict the contents (members) of a module that
# # can be fetched, we must first make the Checkout
# # access control return a "decline" value, so that the
# # MemberCheckoutLock control is then called for each member
#
# # First, decline the overall checkout of the module
access decline Checkout only users {$Team1 $Team2}

# # And then allow MemberCheckoutLock for the owning team, Team1
# # and allow MemberCheckout (no lock) for the team that needs only
# # read access $Team2
access allow MemberCheckoutLock only users $Team1
access allow MemberCheckout only users $Team2 when Lock "no"
```

Related Topics

Setting Up Access Controls

Using Access Commands

Access Control Scripting

Locking Module Data

The `lock` command is used to lock a branch of a module. For its corresponding access control, see the topic Access Controls for lock.

The contents of a module (the individual module members) can be locked by using the `-lock` option to `populate`. For its corresponding access control, see the topic Fetching a Module.

Examples

```
access init {
   set ProjectLeads {ProjLead1 ProjLead2}
   set TeamMembers {ProjLead1 ProjLead2 john jane}
   set Team1 {ProjLead1 john}
   set Team2 {ProjLead2 jane}
   set admin { syncmgr }
}

# # Lock of module members, using the 'populate -lock' operation,
# # is controlled by the MemberCheckout access control, but
# # control of locking of a whole branch is controlled by the Lock
# # control
#
# # Allow users to lock their private modules, but only ProjectLeads
# # to lock other module branches.
access deny Lock everyone
access allow Lock only users $ProjectLeads
access filter Lock {
   if {[string match sync:///Modules/users/$user/* $Object]} {
      return ALLOW
   }
   return UNKNOWN
}
```

Changing a Module's Lock Owner

The lock owner of a module member branch and its module branch must be the same user. Therefore, an attempt to change the lock owner of a module branch, if any of the module members on that branch are locked, will fail immediately. Similarly, an attempt to change the lock owner of a module member, if the module's branch is locked, will fail immediately.

If the `switchlocker` attempt does not fail for the above reasons, then the access checks described below take place.

Changing a module branch's lock owner

If the `switchlocker` attempt is made for a module branch, `SwitchLocker` access for the module branch is checked. The `SwitchLocker` action is defined in Access Controls for Changing a Lock Owner. The module branch URL is used as the `Object` parameter of the `SwitchLocker` action. If `SwitchLocker` access is allowed, the lock owner of the module branch is changed. If `SwitchLocker` access is denied, the `switchlocker` attempt fails.

Changing a module member's lock owner

If the `switchlocker` attempt is made for module members, `SwitchLocker` access for the module branch is the first access control checked to determine whether subsequent access checks are necessary. The `SwitchLocker` action is defined in

Access Controls for Changing a Lock Owner. The module branch URL is used as the `Object` parameter of the `SwitchLocker` action.

If `SwitchLocker` access for the module branch is allowed, then no additional access checks are necessary. The module members' lock owners are changed. Similarly, if `SwitchLocker` access for the module branch is denied, then no additional access checks are necessary. The `switchlocker` attempt fails.

However, if `SwitchLocker` access for the module branch is declined, then additional access checks are necessary. `MemberSwitchLocker` access is checked for each member whose lock owner is to be changed.

The default access control file, `$SYNC_DIR/share/AC_Components/AccessControl.hcm`, defines the action `MemberSwitchLocker`:

```
access define MemberSwitchLocker <Object NaturalPath
CurrentLockOwner NewLockOwner>
```

where:

- `<Object>` is the module branch URL
- `<NaturalPath>` is the natural path of the module member
- `<CurrentLockOwner>` is the user name of the person who currently owns the lock
- `<NewLockOwner>` is the user name of the person applying for the new lock.

If `MemberSwitchLocker` access is allowed for a module member, DesignSync changes the lock owner of the module member.

Examples

```
access init {
   set ProjectLeads {ProjLead1 ProjLead2}
   set TeamMembers {ProjLead1 ProjLead2 john jane}
   set Team1 {ProjLead1 john}
   set Team2 {ProjLead2 jane}
   set admin { syncmgr }
}

# # Restrict the ability unlock a module member, and to change
# # the lock owner of a module member, to project leaders.
#
access deny MemberUnlock everyone
access allow MemberUnlock only users $ProjectLeads
access deny MemberSwitchLocker everyone
access allow MembeSwitchLocker only users $ProjectLeads

# # But allow unlock if the user is the lock owner.
access allow MemberUnlock everyone when IsLockOwner "yes"
```

Defining the Enterprise Access Map

When access control is delegated, you can enable the `SwitchLocker` access map. A sample version of the access map is provided, commented out, in the custom DesignSync access control file for the server, `$SYNC_CUSTOM_DIR/servers/<serverName>/<portNumber>/share/AccessControl`.

```
access map SwitchLocker {

    lappend checkmasks [list changeOwner $Object]

    return [list "masks" $checkmasks]

}
```

Related Topics

Setting Up Access Controls

Using Access Commands

Access Control Scripting

Unlocking a Module

The `lock` command is used to lock a branch of a module. The lock on a module branch is released by using the `unlock` command. See Unlocking a Module Branch. The contents of a module (the individual module members) can be locked by using the `-lock` option to `populate`. The lock on a module member is released by using the `cancel` command. See Unlocking Module Contents.

Unlocking a Module Branch

When attempting to `unlock` a module branch, `Unlock` access for the module branch is checked. The `Unlock` action is defined in Access Controls for Unlocking. The module branch URL is used as the `Object` parameter of the `Unlock` action. If `Unlock` access is allowed, the module branch is unlocked. If `Unlock` access is denied, the `unlock` attempt fails.

Unlocking Module Contents

When attempting to `cancel` a lock on a module member, DesignSync first checks the `Unlock` access for the module branch to determine whether subsequent access checks are necessary. The `Unlock` action is defined in Access Controls for Unlocking. The module branch URL is used as the `Object` parameter of the `Unlock` action. (If there are multiple module members whose locks are being cancelled, those module members will all be on the same branch of the module.)

If `Unlock` access for the module branch is allowed, then no additional access checks are necessary. Locks on the module members are cancelled. Similarly, if `Unlock` access for the module branch is denied, then no additional access checks are necessary. The `cancel` attempt fails.

However, if `Unlock` access for the module branch is declined, additional access checks are necessary for the module members whose locks are to be cancelled; `MemberUnlock` access is checked for each of the module's members whose locks are to be cancelled.

The default access control file, `$SYNC_DIR/share/AC_Components/AccessControl.hcm`, defines the action `MemberUnlock`:

```
access define MemberUnlock <Object NaturalPath IsLockOwner>
```

where:

- `<Object>` is the module branch URL
- `<NaturalPath>` is the natural path of the module member
- `<IsLockOwner>` is `yes` if the user performing the unlock operation is the owner, otherwise `no`.

If `MemberUnlock` access is allowed for a module member, DesignSync cancels the module member's lock.

Examples

```
access init {
   set ProjectLeads {ProjLead1 ProjLead2}
   set TeamMembers {ProjLead1 ProjLead2 john jane}
   set Team1 {ProjLead1 john}
   set Team2 {ProjLead2 jane}
   set admin { syncmgr }
}

# # Restrict the ability unlock a module member, and to change
# # the lock owner of a module member, to project leaders.
```

```
#
access deny MemberUnlock everyone
access allow MemberUnlock only users $ProjectLeads
access deny MemberSwitchLocker everyone
access allow MemberSwitchLocker only users $ProjectLeads

# # But allow unlock if the user is the lock owner.
access allow MemberUnlock everyone when IsLockOwner "yes"
```

Related Topics

Setting Up Access Controls

Using Access Commands

Access Control Scripting

Creating a New Version of a Module

A new version of a module is created when any of these operations occur:

- Data is checked into the module (using the `ci` command, or the `-checkin` option to the `mkmod` command)
- An href is added to a module version (using the `addhref` command)
- An href is removed from a module version (using the `rmhef` command)
- A member object is removed from a module (using the `remove` command)
- A member object is renamed (using the `mvmember` command)

When any of the operations above take place, DesignSync first checks the `Checkin` access for the module, to determine whether subsequent access checks are necessary. The `Checkin` action is defined in Access Controls for Checking In. Note that when the `Object` is a module URL, the `Branch` and `NewBranchName` parameters are not applicable.

If `Checkin` access for the module is allowed, no additional access checks are necessary. The operation succeeds, creating a new version of the module. Similarly, if `Checkin` access for the module is denied, no additional access checks are necessary. The operation fails.

However, if `Checkin` access for the module is declined, additional access checks are necessary:

- If the operation is `addhref`, DesignSync checks `Addhref` access. See Access Controls for addhref for details, including the outcome of the access check.

- If the operation is `rmhref`, DesignSync checks `Rmhref` access. See Access Controls for rmhref for details, including the outcome of the access check.
- If the operation is `remove`, DesignSync checks `MemberRemove` access. See Access Controls for remove for details, including the outcome of the access check.
- If the operation is `mvmember`, DesignSync checks `MemberRename` access. See Access Controls for mvmember for details, including the outcome of the access check.
- If the operation is `ci`, DesignSync checks `MemberCheckin` access for each member being checked in.

If the operation is `mkmod` with the `-checkin` option, then the initial step where the module is created on the server will be verified for `Mkmod` access. If access is allowed to make the module, the module is created on the server. The next step is to verify module level `Checkin` access. If that access is granted, then the `-checkin` operation proceeds. Otherwise, no checkin will occur with the module's creation.

MemberCheckin Access

Conceptually, a `ci` operation has two steps:

1. Pre-checkin checks, which include access control checks
2. Transfer of data to the server

If `Checkin` access for the module is declined, DesignSync checks `MemberCheckin` access for each member object participating in the checkin. DesignSync checks the member object because it might have changed, or it might have been newly added or re-added.

The first `MemberCheckin` access that is denied terminates the `ci` operation. DesignSync does not check access controls for the remaining items. If `MemberCheckin` access is allowed for all member objects participating in the checkin, the `ci` operation proceeds.

The default access control file, `$SYNC_DIR/share/AC_Components/AccessControl.hcm`, defines the `MemberCheckin` action:

```
access define MemberCheckin <Object NaturalPath Skip>
```

where:

- `<Object>` is the module branch URL
- `<NaturalPath>` is the natural path of the module member

- `<Skip>` is `yes` if this is a "skipping" checkin, `no` if it is not a "skipping" checkin. See the `ci` command documentation for the definition of a "skipping" checkin as it pertains to module data.

Examples

```
access init {
   set ProjectLeads {ProjLead1 ProjLead2}
   set TeamMembers {ProjLead1 ProjLead2 john jane}
   set Team1 {ProjLead1 john}
   set Team2 {ProjLead2 jane}
   set admin { syncmgr }
}

# # For the commands that creates a new module version:
# #     ci, addhref/rmhref and remove/mvmember,
# # a check is first made for Checkin access, and the
# # check for the sub-command only performed if the
# # Checkin access returns a value of "decline"
#
# # So, first "decline" the Checkin access for everyone,
# # so that the the member check is called for all users
#
access decline Checkin everyone

# # Now can restrict the Team2 users to only checking in
# # objects with Natural Paths under /src area
#
access deny MemberCheckin only users $Team2
access allow MemberCheckin only users $Team2 \
        when NaturalPath /src/*

# # Similarly, we can restrict Team2 to not removing anything
# # outside their area
access deny MemberRemove only users $Team2
access allow MemberRemove only users $Team2 \
        when NaturalPath /src/*

# # And can restrict Team2 to moving things where BOTH the
# # paths are in their area.
access deny MemberRename only users $Team2
access allow MemberRename only users $Team2 \
        when NaturalPath /src/* \
        when NewNaturalPath /src/*

# # And can restrict adding hrefs to only project leaders
```

```
#
access allow {Addhref Rmhref} only users $ProjectLeads \
        -because "only Project leaders can add or remove hrefs"
```

Related Topics

Setting Up Access Controls

Using Access Commands

Access Control Scripting

Branching a Module

DesignSync checks access to the `MakeBranch` action (defined in Access Controls for Creating Branches) when creating a branch of a module object. Note that `MakeBranch` access is never checked for module members, which are never explicitly branched.

Related Topics

mkbranch command

Tagging a Module

DesignSync checks access to the `Tag` action (defined in Access Controls for Tagging) when tagging module data. Note that when a module is tagged, DesignSync checks `Tag` access only once, with the module as the `Object`; `Tag` access is not checked for each of the module's member objects.

Examples

```
access init {
    set ProjectLeads {ProjLead1 ProjLead2}
    set TeamMembers {ProjLead1 ProjLead2 john jane}
    set Team1 {ProjLead1 john}
    set Team2 {ProjLead2 jane}
    set admin { syncmgr }
}

# # Module tags are mutable (may be removed/moved) or immutable (may not)
# # Anyone can add immutable tags, but only our project leads are allowed
# # to remove/move them.
access filter Tag {
  if {("$Mutability" == "IMMUTABLE") && ("$Action" != "ADD")} {
    if {[lsearch -exact $ProjectLeads $user] == -1} {
      return "Only project leads may $Action immutable tags."
    }
  }
  return UNKNOWN
}
```

Related Topics

tag command

Tagging a Module Snapshot

DesignSync checks access to the `Tag` action (defined in Access Controls for Tagging) and the `MemberTag` action when tagging module data in a module snapshot. Unlike Module tag, each member participating in the tagged snapshot is checked to determine that the access is valid.

The default access control file, `$SYNC_DIR/share/AC_Components/AccessControl.hcm`, defines the action `MemberTag` to control access to individual module members for snapshot views.

```
# MemberTag:
# Arguments:
#    Object: string representation of the URL of the Module,
which member is being tagged.
#         eg:  sync:///Modules/ALU
#    NaturalPath:  string representation of the natural path of
the module member being tagged.
#    Version: string representation of the member's numeric
version being tagged.
#    NewTag: value of the tag being added, replaced, or deleted.
#    Action: type of action being performed. Possible values
are:
#              ADD =>     'NewTag' is being added to 'Object'.
#              DELETE =>  'NewTag' is being deleted from
'Object' vault.
#              REPLACE => 'NewTag' present on some other
version/branch is being moved to 'Object'.

access define MemberTag {Object NaturalPath Version NewTag
Action}
```

Related Topics

tag command

Rolling Back a Module

DesignSync checks access to the `Rollback` action when attempting to roll-back a module to a previous module version.

Example

```
access init {
   set ProjectLeads {ProjLead1 ProjLead2}
   set TeamMembers {ProjLead1 ProjLead2 john jane}
   set Team1 {ProjLead1 john}
   set Team2 {ProjLead2 jane}
   set admin { syncmgr }
}


# # Example for 'rollback' command.
# #
# # Allow rollback of modules in the ProjA area to the
# # team members
# # Remember that rollback is denied to all users by default.
access allow Rollback only users $TeamMembers when Object sync:///Modules/ProjA/*
```

Related Topics

Access Controls for Rollback

Making a Module Edge

DesignSync checks access to the `Mkedge` action when attempting to create a merge edge on a module .

Example

```
access init {
   set ProjectLeads {ProjLead1 ProjLead2}
   set TeamMembers {ProjLead1 ProjLead2 john jane}
   set Team1 {ProjLead1 john}
   set Team2 {ProjLead2 jane}
   set admin { syncmgr }
}

# # Example for 'Mkedge' command.
# #
# # Allow Mkedge on modules in the ProjA area to the
# # Project leads
# # Remember that Mkedge is denied to all users by default.
access allow Mkedge only users $ProjectLeads when Object sync:///Modules/ProjA/*
```

Related Topics

Access Controls for Creating Merge Edges

Access Controls for the Enterprise Design System

Access Controls for Enterprise Design Push to DesignSync

If the data structure for the Enterprise Design System is maintained on the Enterprise Design system, not in DesignSync, you should adjust the access controls to deny structure editing from a DesignSync client, but allow for changes to push from ENOVIA clients. To facilitate this working model, DesignSync stores a property on the module or the branch to indicate that the data is managed in ENOVIA.

To prevent modification from a DesignSync client, you must disable Checkin access to the module, but allow checkin of module members.

Examples

```
access decline Checkin everyone when Object sync:///Modules/*
...
access allow MemberCheckin everyone
...
#
# This filter for Addhref/Rmhref prevents href modification
# when the module branch in question is recorded as being managed
# by the Enterprise platform.
# NOTE: to use this, you must also "decline" Checkin access, as
# the filter for Addhref/Rmhref is only called if the Checkin is
# declined.
#

access filter {Addhref Rmhref} {

        #
        # First, check whether the operation is
        # being performed from the platform, which means that
        # an "export" is being run. In that case, we want to
        # allow the operation.
        #

        if {$agent == "Platform"} {

                return UNKNOWN

        }
        #
        # Otherwise, see whether the branch being operated on is
        # recorded as managed.
        #

        # For that, we need the branch numeric, which we can get from the
        # version passed in. The Object always has the version at the
        # end by this point.
        #

        foreach {objectUrl version} [split $Object ";"] {}
        set branchNum [string range $version 0 [expr [string last "." $version] - 1]]
        set branchUrl "$objectUrl\;$branchNum"
        if {[entobj isplatformmanaged $branchUrl]} {

                return "Cannot modify hrefs for $branchUrl : The structure for this item is
                being managed by the Enterprise Platform"

        }
        return UNKNOWN

}
```

Access Controls for Enterprise Design Synchronization

The default access control file, `$SYNC_DIR/share/AC_Components/AccessControl.hcm`, defines the action EnterpriseSynchonrize to control whether a user can perform Enterprise Design actions (commands that use the `entobj` command prefix):

```
access define EnterpriseSynchronize
```

This command takes no arguments. By default, this is enabled for all users.

Examples

```
access init {
   set ProjectLeads {ProjLead1 ProjLead2}
   set TeamMembers {john jane}
   set Team1 {ProjLead1 john}
   set Team2 {ProjLead2 jane}
   set admin { syncmgr }
}
```

```
# # Restrict Enterprise Synchronization commands to  only project leaders
#
access deny {EnterpriseSynchronize} only users $TeamMembers \
        -because "only Project leaders can perform synchronize actions"
```

Related Topics

Access Controls for Enterprise Design Push to DesignSync

Access Controls for Enterprise Design Administration Reference Workspace Creation

There is no explicit control for creating a reference workspace for Enterprise Design Administration, however, in order to create a reference workspace for an Enterprise Development, you need the following access:

- Browse access required for the module being duplicated.
- AddMirrors access.

Access Controls for ProjectSync

ProjectSync Action Definitions

ProjectSync Action Definitions

You set up access controls on particular ProjectSync **actions**, or operations. To set up an access control on an operation, the operation must have an **action definition** specified with an `access define` command. If an action definition exists for an operation, you can control access to that operation using the stcl `access allow`, `access deny`, and `access filter` commands.

If a custom access control rule affects whether menu items are displayed or hidden from users, users will need to refresh their browsers, for their ProjectSync menu to reflect the new access control rules. If the user no longer has permission to access a currently displayed menu item, clicking that menu item will display an Operation Failed panel, with a permission error.

ProjectSync provides a number of predefined actions corresponding to many of the commands you might want to access control. These actions are defined in the default access control files for ProjectSync:

- Project and configuration action definitions:

  ```
  $SYNC_DIR/share/AC_Components/AccessControl.ps
  ```

- Other action definitions (for example, for notes and note types, server administration, and user administration):

  ```
  $SYNC_DIR/share/AC_Components/AccessControl.psipg
  ```

See Introduction to Access Control for details on the individual files used to define access controls.

Important: Do not edit the `AccessControl.ps` or `AccessControl.psipg` file; changes will be lost upon upgrading. Instead, edit your site or server custom `AccessControl` file (see Setting Up Access Controls).

You can set access controls for the following types of predefined ProjectSync actions:

Email administration	Secure communications
Groups of commands	Server administration
Notes and note types	Triggers
Projects and configurations	User profiles

Although most ProjectSync actions are predefined in the default `AccessControl.ps` and `AccessControl.psipg` files, you might want to create your own action definitions for custom operations. Do not redefine the existing actions using `access define` statements in your site and server `AccessControl` files. You will not be able to access the server if it detects duplicate `access define` statements. You can, however, define new actions within your site or server `AccessControl` files. See Access Control Scripting for an example of a custom action definition.

Related Topics

Module Action Definitions

Revision Control Action Definitions

Setting Up Access Controls

User Authentication Action Definitions

Access Controls for Email Administration

The default ProjectSync access control file:

```
$SYNC_DIR/share/AC_Components/AccessControl.psipg
```

defines two actions, `EmailSubscribe` and `EmailMgrAdmin`, that control email subscriptions:

```
access define EmailSubscribe <isSelf>
```

```
access define EmailMgrAdmin
```

The `EmailSubscribe` action controls whether the user can create email subscriptions. The optional argument `isSelf` is a Boolean that takes the value `1` for the current user or `0` for any other user.

The `EmailMgrAdmin` action controls whether the user can access the Email Administrator.

Example

The following access control rule allows users to create email subscriptions only for themselves:

```
access allow EmailSubscribe everyone when isSelf 1
```

See Setting Up Access Controls for information on creating your own custom `AccessControl` files. See Using Access Commands for a list of the access commands and examples of how to use them.

Related Topics

Access Control Scripting

Sample Access Controls

Setting Up Access Controls

Using Access Commands

Access Controls for Notes and Note Types

The default ProjectSync access control file:

```
$SYNC_DIR/share/AC_Components/AccessControl.psipg
```

defines the following actions to control access to notes:

```
access define ViewNote <system> <type> <id>

access define EditNote <system> <type> <id>

access define EditNoteAttachments <system> <type> <id>
<attachType>

access define AddNote <system> <type>

access define DeleteNote <system> <type> <id> <isAuthor>

access define SetNoteProperty <system> <type> <field>

access define ModifyNoteProperty <system> <type> <id> <field>
<oldval> <newval>

access define ReviseNoteHistory <system> <type> <id> <isAuthor>

access define AdministrateNoteTypes <system>
```

Where:

- `<system>` is always `SyncNotes`. This parameter is ignored, but is included for future extensibility of ProjectSync.
- `<type>` is the name of the note type to which you are trying to control access, for instance `"BugReport"`.
- `<id>` is the note ID number of the note that you are trying to modify, for example 125.
- <`attachType`> is the specific "type" of attachment. The type can be "ProjectName", "ProjectConfiguration" or "Other".
- `<isAuthor>` is a Boolean that takes the value `1` for the author of the note or `0` for any other user.
- `<field>` is the name of the property used to generate a field on the GUI.
- `<oldval>` is the old value of a property that you want to modify.
- `<newval>` is the new value of a property that you want to modify.

These parameters are used in `when` clauses in `access allow` and `access deny` statements. (See Using Access Commands for details.)

The `$NoteActions` variable can be used to control access to all of these actions except for `AdministrateNoteTypes`. See Access Controls for Groups of Commands for details.

See Setting Up Access Controls for information on creating your own custom `AccessControl` files. See Using Access Commands for a list of the access commands and examples of how to use them.

The ViewNote Action

The `ViewNote` action controls whether users can view notes. If users do not have permission to view notes, they cannot display the View or Edit panel for any note. In addition, they cannot run queries on notes or view reports stored on the server. Stored reports are saved in a StoredReport note type that is governed by this access control. Disabling `ViewNote` permission removes all access to notes and reports.

The `ViewNote` action also restricts email notifications when notes are created or modified. For example, if a user does not have permission to view SyncDefects, he or she will not receive email when a SyncDefect is created or modified. This is the case even when the individual's username appears in a SyncUserList field, such as **Responsible**.

The EditNote Action

The `EditNote` action controls whether users can edit notes and reports stored on the server. Stored reports are saved in a StoredReport note type that is governed by this access control.

The `ViewNote` and `EditNote` actions together control access to notes and reports on a per-user basis. It is possible to use the `access db_filter` command to limit access to notes or reports that have particular characteristics. For example, suppose a note has a Boolean field called `private`. If this field is set to `True`, only some users can view the note.

You also can use the `access filter` command for limiting access to notes. However, using this command for this purpose can cause noticeable performance degradation in note queries. For every note that matches the query, a Tcl shell is started to run the filter that decides whether to list the note.

The EditNoteAttachments Action

The `EditNoteAttachments` action controls whether users can select Project, Configuration or Other Attachments, when adding or editing a note.

The AddNote Action

The `AddNote` action controls whether users can add notes.

The DeleteNote Action

The `DeleteNote` action controls whether a user can delete notes. If a user does not have permission to delete notes, the **Delete Note** button does not display on the Edit Note panel of any note.

The following sample access control rule allows users to delete notes they created, but not other users' notes:

```
access deny DeleteNote everyone when isAuthor 0
```

The SetNoteProperty Action

The `SetNoteProperty` action controls whether users can edit specified properties of a note when adding a note for the first time. This action normally is used in conjunction with the `ModifyNoteProperty` action.

For example, the following access control rule prevents everyone from editing the Audit Trail field on the Add Note panel:

```
access deny SetNoteProperty everyone when field AuditTrail
```

If this command is added to your custom `AccessControl` file, users adding a note can see the AuditTrail field, but cannot edit it. With this example, you also would use a corresponding `ModifyNoteProperty` access control.

In the GUI, fields that are made non-editable in this way are displayed as non-editable. Currently, the only way remove a field from the Add Note panel is to use custom templates for the note type. (See ProjectSync User's Guide: Generating HTML Templates for a Note Type for details.)

The ModifyNoteProperty Action

The `ModifyNoteProperty` action controls whether users can modify a field of a note.

If a field cannot be modified because of an access restriction, it displays as non-editable text in the Edit Note panel. If a Choice List field can be modified, but some of the values in the choice type are not allowed, then those values will not appear in the choice pulldown menu in the Edit Note panel. The same applies to State Machine fields.

This access control does not apply to the attachments of a note; the `EditNoteAttachments` action controls the attachment of objects to a note.

The following examples show different ways you can apply this action.

Example 1

Some note types are set up with an AuditTrail field that is automatically updated by the system and should not be edited manually. You could enforce this policy as follows:

```
access deny ModifyNoteProperty everyone \
  when type "SyncDefect" \
  when field "AuditTrail"
```

Example 2

Suppose you want to enforce the policy that only a BugReport's author can close the bug. In this case, you can define an access filter similar to the following:

```
access filter ModifyNoteProperty \
  when type "BugReport" \
  when field "State" \
  when newval "closed" \
{
  set note_url "sync:///Note/$system/$type/$id"
  if {[url getprop $note_url Author] == "$user"} {
  return ALLOW
  } else {
  return DENY
  }
}
```

This example uses `url getprop`, but another less-efficient option is `url properties`. The `when` clauses can be moved down into the body of the filter definition as a Tcl `if` statement. Either method is acceptable, but the method used in the example avoids the overhead of firing up a Tcl interpreter in most cases. Because the `ModifyNoteProperty` access control is accessed often, you should make the code as efficient as possible.

Example 3

Suppose you have installed the packaged SyncDefect note type. By default, open to closed is not a valid transition for the State state machine. An interim state, such as verified, is required. However, you want to allow a few users to move SyncDefects directly from open to closed.

First use the Property Type Manager to edit the SD-State State Machine. (See ProjectSync User's Guide: Editing Property Types for details.) Select closed as a valid state transition from open and submit your change.

At this point, all users can move SyncDefects from open to closed. To limit this capability to selected users, add the following access filter to your custom `AccessControl` file:

```
access init {
  set QA "tester1 tester2 tester3"
}
access filter ModifyNoteProperty \
  when type SyncDefect \
  when field State \
  when oldval open \
  when newval closed \
{
# Allow test engineers to transition SyncDefects from "open" to
"closed", as they will perform the necessary verification.
if {[lsearch $QA $user] != -1} {
  return ALLOW
}
# Get the note's properties
set author [url getprop "sync:///Note/$system/$type/$id" Author]
# Allow anyone to close a SyncDefect that they submitted
if {$user == $author} {
  return ALLOW
}
# Otherwise...
  return DENY
}
```

You can create similar access controls for fields of type Choice List using the `newval` parameter.

The ReviseNoteHistory Action

The `ReviseNoteHistory` action controls whether users can make a change the history of a note.

For example, to give a note's author, but not other users, the ability to rewrite history, add the rule:

```
access deny ReviseNoteHistory everyone when isAuthor 0
```

To limit access to a list of trusted users, add the rule:

```
access allow ReviseNoteHistory only users $trusted_users
```

To prevent access for all users, add the rule:

```
access deny ReviseNoteHistory everyone
```

The AdministrateNoteTypes Action

The `AdministrateNoteTypes` action controls whether users can perform the tasks handled by the Note Type Manager in ProjectSync:

- Install a new note type in the system
- Modify an existing note type
- Create a custom note type
- Import a note type from a DEF file
- Delete a note type
- Rename a note type
- Generate an HTML template from a note type
- Add or modify property types used in note types

Note: The `AdministrateNoteTypes` action replaces the `AddNoteType` action. Access control rules that use `AddNoteType` in existing scripts do not cause errors, although the use is ignored.

Related Topics

Access Control Scripting

Sample Access Controls

Setting Up Access Controls

Using Access Commands

Access Controls for Projects and Configurations

The default ProjectSync access control file:

```
$SYNC_DIR/share/AC_Components/AccessControl.ps
```

defines the following actions to control access to projects and configurations.

Project Access Controls

```
access define AddProject

access define ModifyProject <project>

access define DeleteProject <project>

access define ImportProject <project>

access define ExportProject <project>
```

These access controls are applied as follows:

- The `AddProject` action controls who can create new projects.
- The `ModifyProject` action controls who can edit projects.
- The `DeleteProject` action controls whether you can remove projects from ProjectSync. This access control also determines whether you can clean up data exported from a server. See ProjectSync User's Guide: Exporting Projects for details. By default, this access control is turned off for all users.
- The `ImportProject` action controls who can import a project onto a server.
- The `ExportProject` action controls who can export a project for transfer to another server

Configuration Access Controls

```
access define CreateConfig <project>

access define ModifyConfig <project config>

access define DeleteConfig <project config>
```

These access controls are applied as follows:

- The `CreateConfig` action controls who can create configurations.
- The `ModifyConfig` action controls who can edit configurations.
- The `DeleteConfig` action controls who can remove configurations. By default, this access control is turned off for all users.

The `<project>` parameter, when present, specifies a project name such as `ASIC` or `CPU`. The `config` parameter in the `ModifyConfig` action specifies a configuration name such as `rel12` or `stable`.

The `$ProjectActions` variable can be used to control access to all project-related actions; the `$ConfigActions` variable can be used to control access to all configuration-related actions. The `$ProjAndConfigActions` variable controls access to all the project and configuration actions. See Access Controls for Groups of Commands for details.

Example

The following access control rule lets only project leaders create, edit, or delete configurations:

```
access init {
   set projectLeaders { chan kapoor }
}

access allow {CreateConfig ModifyConfig DeleteConfig} only users
$projectLeaders
```

ProjectSync sometimes passes in the "`*`" qualifier for `project` and `config` to `access verify`. See Using Access Command Qualifiers for more information.

See Setting Up Access Controls for information on creating your own custom `AccessControl` files. See Using Access Commands for a list of the access commands and examples of how to use them.

Related Topics

Access Control Scripting

Sample Access Controls

Setting Up Access Controls

Using Access Commands

Access Controls for Server Administration

The actions available from the **Server** section of the ProjectSync menu are controlled by two different actions:

- The default ProjectSync access control file:

  ```
  $SYNC_DIR/share/AC_Components/AccessControl.psipg
  ```

  defines a single action, `AdministrateServer`, to control whether a user can access the **Administer Server** and **Reset Server** options on the ProjectSync menu:

  ```
  access define AdministrateServer
  ```

- The default ProjectSync access control file:

  ```
  $SYNC_DIR/share/AC_Components/AccessControl.ps
  ```

  defines a single action, `ResetAccessControls`, to control whether a user can access the **Access Reset** options on the ProjectSync menu:

  ```
  access define ResetAccessControls
  ```

Example

To allow only administrators to access the Administer Server panels and **Reset Server** menu option, add the following rule to your custom `AccessControl` file:

```
access allow AdministrateServer only users $admin
```

See Setting Up Access Controls for information on creating your own custom `AccessControl` files. See Using Access Commands for a list of the access commands and examples of how to use them.

Related Topics

Access Control Scripting

Sample Access Controls

Setting Up Access Controls

Using Access Commands

Access Controls for Triggers

The default ProjectSync access control file:

```
$SYNC_DIR/share/AC_Components/AccessControl.psipg
```

defines three actions for controlling access to triggers:

```
access define AddTrigger

access define EditTrigger

access define DeleteTrigger
```

These actions let you control who can add, edit, or delete triggers. The commands do not need arguments.

If a user does not have `AddTrigger` or `EditTrigger` access, the corresponding options do not appear on the ProjectSync menu. If a user does not have `DeleteTrigger` access, the **Delete** button does not appear on the Edit Trigger panel.

The `$TriggerActions` variable can be used to control access to all of these actions as well as access to the Email Administrator. See Access Controls for Groups of Commands for details.

Example

For example, to allow only administrators to add or edit triggers, add the following rule to your custom `AccessControl` file:

```
access init {
   set admin { lynch vega }
}

access allow {AddTrigger EditTrigger} only users $admin
```

See Setting Up Access Controls for information on creating your own custom `AccessControl` files. See Using Access Commands for a list of the access commands and examples of how to use them.

Related Topics

Access Control Scripting

Sample Access Controls

Setting Up Access Controls

Using Access Commands

Access Controls for User Profiles

The default ProjectSync access control file:

```
$SYNC_DIR/share/AC_Components/AccessControl.psipg
```

defines three actions to control access to user profiles:

```
access define AddUser

access define EditUser <username <isSelf>

access define DeleteUser <username <isSelf>
```

Where:

- `<username>` is the login name (not the full name) of the user that you are trying to edit or delete.
- `<isSelf>` is a Boolean that takes the value `1` for the current user or `0` for any other user.

The `$UserActions` variable can be used to control access to all of these actions as well as access to the email subscriptions. See Access Controls for Groups of Commands for details.

Example

The following access control rule lets users to modify their own passwords but not other users' passwords:

```
access deny EditUser everyone when isSelf 0
```

See Setting Up Access Controls for information on creating your own custom `AccessControl` files. See Using Access Commands for a list of the access commands and examples of how to use them.

Related Topics

Access Control Scripting

Sample Access Controls

Setting Up Access Controls

Using Access Commands

Access Controls for Stored Reports

The default ProjectSync access control file:

```
$SYNC_DIR/share/AC_Components/AccessControl.psipg
```

defines two actions, `AddStoredReport` and `DeleteStoredReport`, that control reports stored on the server:

```
access define AddStoredReport <noteType <visibility>

access define DeleteStoredReport <id> <visibility> <isAuthor>
```

Stored reports are saved in a StoredReport note type that is governed by these access controls.

The `AddStoredReport` action controls whether the user can create stored reports on the server, where:

- `<noteType>` is the name of the note type for which reports can be stored.
- `<visibility>` is the accessibility of the report, either `private` or `public`.

The `DeleteStoredReport` action controls whether the user can remove reports stored on the server, where:

- `<id>` is the note ID number of the report.
- `<visibility>` is the accessibility of the report, either `private` or `public`.
- `<isAuthor>` is a Boolean that takes the value `1` for the author of the note or `0` for any other user.

By default, all users can view, edit, or delete public reports stored on the server, but only the author can view, edit, or delete private reports. See ProjectSync Help: Saving a Standard Query for details on storing reports on the server.

A user's ability to view or edit a stored report is controlled by the `ViewNote` and `EditNote` access control rules. See Access Controls for Notes and Note Types for details on these actions.

Examples

The following access control rule prevents all users from creating stored reports on the SecretResearch note type:

```
access allow AddStoredReport everyone
access deny AddStoredReport everyone when noteType
SecretResearch
```

The following access control rule prevents all users except the author from deleting public stored reports:

```
access allow DeleteStoredReport everyone
access deny DeleteStoredReport everyone when visibility public
when isAuthor 0
```

See Setting Up Access Controls for information on creating your own custom `AccessControl` files. See Using Access Commands for a list of the access commands and examples of how to use them.

Related Topics

Access Control Scripting

Sample Access Controls

Setting Up Access Controls

Using Access Commands

Access Controls for Groups of Commands

A number of variables defined in the default ProjectSync access control files make it more convenient to set the same access rights on groups of related commands. These variables are defined in the default ProjectSync access control file:

```
$SYNC_DIR/share/AC_Components/AccessControl.ps
```

You also can create your own sets of commands in your custom `AccessControl` file.

Predefined Variables

The variables defined in the default `AccessControl.ps` file are:

```
set NoteActions {ViewNote EditNote EditNoteAttachments AddNote
DeleteNote SetNoteProperty ModifyNoteProperty ReviseNoteHistory}
;#excludes AdministrateNoteTypes

set ProjectActions {AddProject ModifyProject DeleteProject
ImportProject ExportProject}
```

```
set ConfigActions {CreateConfig ModifyConfig DeleteConfig}

set ProjAndConfigActions "$ProjectActions $ConfigActions"

set UserActions {AddUser EditUser DeleteUser EmailSubscribe}

set TriggerActions {AddTrigger EditTrigger DeleteTrigger
EmailMgrAdmin}

set ProjectSyncReadActions {ViewNote BrowseServer}

set ProjectSyncWriteActions {AddNote EditNote
EditNoteAttachments DeleteNote SetNoteProperty
ModifyNoteProperty ReviseNoteHistory AddProject ModifyProject
DeleteProject ImportProject ExportProject CreateConfig
ModifyConfig DeleteConfig AddUser EditUser DeleteUser
EmailSubscribe EmailMgrAdmin AddTrigger EditTrigger
DeleteTrigger AdministrateNoteTypes AdministrateServer
ResetAccessControls}

set ProjectSyncActions "AdministrateNoteTypes AdministrateServer
BrowseServer $NoteActions $ProjAndConfigActions $UserActions
$TriggerActions"

set AllActions "$DesignSyncActions $ProjectSyncActions
$HCMActions $MirrorActions"

set ProjectAndConfigWriteActions {ModifyProject DeleteProject
ImportProject CreateConfig ModifyConfig DeleteConfig}
```

Examples

The following access control rule specifies that only the user `psadmin` can perform the actions defined by the `$UserActions` variable -- adding, editing, or deleting user profiles and creating subscriptions for users:

```
access allow $UserActions only users psadmin
```

The following access control rule specifies that only project leaders (as defined by the variable `$projectLeaders`) can create or edit projects and configurations:

```
access allow $ProjAndConfigActions only users $projectLeaders
```

See Setting Up Access Controls for information on creating your own custom `AccessControl` files. See Using Access Commands for a list of the access commands and examples of how to use them.

See Access Controls for Groups of Commands in the DesignSync section for information on the predefined variables defined for DesignSync.

Custom-Defined Variables

You can use the `set` command to create variables that represent your own groupings of access control actions.

For example, to create a variable that controls the ability to delete objects, you could specify:

```
set DeleteActions {DeleteNote DeleteProject DeleteConfig
DeleteTrigger}
```

You can then use the variable `$DeleteActions` in your access control statements.

You also can create variables that include other variables. However, when declaring a list that requires variable substitution, you must enclose the list in quotation marks, not curly braces. (Curly braces prevent variable substitution.) For example:

```
set AdminActions "$ProjAndConfigActions $TriggerActions
BrowseServer"
```

Related Topics

Access Control Scripting

Sample Access Controls

Setting Up Access Controls

Using Access Commands

Example ProjectSync Access Controls

Sample Access Controls

The following examples demonstrate ways you can apply ProjectSync access controls in your custom `AccessControl` files. You can find other examples in the file:

```
$SYNC_DIR/share/examples/ExampleAccessControl
```

This topic provides examples of access controls implemented using `access filter` and `access db_filter` scripts, as well as examples of some general access control solutions:

Controlling who can perform specified operations

Controlling who can edit defects

Controlling who can close defects

Defining custom actions to conditionalize Tcl code

Note: Do not edit any of the access control files in the `$SYNC_DIR/share` area; you edit the site or server `AccessControl` file. See Setting Up Access Controls for the locations of these `AccessControl` files.

See Setting Up Access Controls for information on creating your own custom `AccessControl` files. See Using Access Commands for a list of the access commands and examples of how to use them.

Controlling Who Can Perform Operations

In this example, the variables `$admin`, `$projectLeaders`, and `$engineers` are defined inside an `access init` statement. These variables are then used to control who can perform certain operations.

```
access init {
  set admin {syncmgr}
  set projectLeaders {bob}
  set engineers {tom dick harry sally dave}
}

# Only administrators can add or manipulate NoteTypes
access allow AdministrateNoteTypes only users $admin

# Only those on the project can edit AdminNote Notes
access allow EditNote only users "$projectLeaders $engineers"
  when type "AdminNote"

# Only project leaders can manipulate projects
access allow {AddProject ModifyProject DeleteProject}
  only users $projectLeaders

# Only project leaders can create or delete configurations
access allow {CreateConfig ModifyConfig DeleteConfig}
  only users $projectLeaders

# Only administrators can edit triggers
access allow {AddTrigger EditTrigger} only users $admin
```

```
# Only administrators can add/delete/modify any user's
# profile or email subscription
access allow $UserActions only users $admin

# But individuals can edit their own profiles
# and email subscriptions
access allow {EditUser EmailSubscribe} everyone when isSelf 1

# Only administrators can administrate email
access allow EmailMgrAdmin only users $admin
```

Controlling Who Can Edit Defects

In this example, a Tcl script is used with an `access db_filter` statement to control who can edit a HW-Defect-1 note. The `access db_filter` command is used in preference to `access filter` because `access db_filter` performs better for operations that involve verifying more than one note.

The script first checks whether the user (`$user`) is a project leader. If so, the user can edit the note. If `$user` is not a project leader, the script sets up a query, `AC_squery`, and calls the `FILTERED_IDS` function to check whether `$user` is the author of the note or the person responsible for the note. If `$user` is the author or person responsible for the note, the user is allowed to edit it. If `$user` does not fit into these categories, the user cannot edit the note.

The `AC_squery` query that gets passed to the `FILTERED_IDS` function is defined using SQL syntax. The naming conventions require that property names in the query have a '`f_`' prefix, for example, you specify the '`Author`' property in queries as '`f_Author`'.

```
access init {
  set projectLeaders { smith jones karen }
}

#
# Allow editing notes only to project leaders,
# the note's author, and its responsible.
#

access db_filter EditNote when type "HW-Defect-1" {

  CHECK_STAR

  # CHECK_STAR checks if first note Id == '*' and
  # allows action if so; this is used so that the
  # note type shows up in the Quick View panel.


  if {[lsearch $projectLeaders $user] != -1} {
    ALLOW_ALL
  }

  # Set up an SQL filter query; only the Resp or
  # Author of the note is allowed.

  set AC_squery "f_Author = [sq $user] \
                OR f_Resp = [sq $user]"

  # The FILTERED_IDS function runs a subquery, which
  # is the original query with the AC_squery tacked
  # onto it.

  set AC_ids [FILTERED_IDS $type $sqlquery $dbquery \
    $attached $AC_squery]

  # Notes that we got back above passed both the user's
  # query criteria and the filter criteria, so let's
  # ALLOW them.

  foreach noteId $AC_ids {
    ALLOW $noteId
  }

  # Now that we've ALLOWed some notes, we call the
  # FOREACH_UNKNOWN function to DENY those notes
  # that did not match our query criteria and are now
  # in an indeterminate state (not ALLOWed or DENYed).

  FOREACH_UNKNOWN noteId {
    DENY $noteId
  }
}
```

Controlling Who Can Close Defects

In this example, a Tcl script is used with an `access filter` statement to control who can close a HW-Defect-1 note. The script first checks whether the user (`$user`) is a project leader. If so, the user can close the note.

If `$user` is not a project leader, the script uses the `url properties` command to get the properties for the note. If the properties show that `$user` is the author of the note or the person responsible for the note, the user is allowed to close it. If `$user` does not fit into these categories, the user cannot close the note.

```
access filter ModifyNoteProperty \
  when type "HW-Defect-1" \
  when field "Status" \
  when newval "Closed" \
{
  # If the user is a project leader, he or she can close it
  if {[lsearch $projectLeaders $user] != -1} {
    return ALLOW
  }
  # Get the note's properties
  url properties "sync:///Note/$system/$type/$id" props
  # If the user is the note's Author, he or she can close it
  if {$user == $props(Author)} {
    return ALLOW
  }
  # If the user is 'Responsible,' he or she can close it
  if {$user == $props(Resp)} {
    return ALLOW
  }
  # The user is not allowed to close the HW-Defect-1
  return DENY
}
```

See the ENOVIA Synchronicity Command Reference for further information on the `url properties` command and other commands that you can use in Tcl scripts for access controls.

Defining Custom Actions to Conditionalize Tcl Code

In this example, you want to conditionalize segments of stcl code, so that particular users have access to that code while other users are restricted. For example, you might want a ProjectSync panel to be viewable by only a particular set of users.

You can create custom actions that you later use to conditionalize your code. To do so, you define the new action in your site or server `AccessControl` file using the `access define` statement. You also specify the users who will have access to this conditionalized segment of stcl code using the `access allow` statement:

```
# Define new actions.
access define DisplayCustomPanel
access allow DisplayCustomPanel only users {syncmgr}
```

Then you can conditionalize your stcl code, using the `access verify` statement, as in this excerpt of a server-side script:

```
set allowDef [access verify DisplayCustomPanel $SYNC_User]
...

if {$allowDef} {
  ...
} else {
  ...
}
```

This example uses the `access verify` command to verify that a user is allowed access to a particular segment of code. The `access` commands are server-side-only commands; thus, this method of conditionalizing code works for server-side scripts, not for client-side scripts. As with all server-side scripts, you can invoke the script from a client using the `rstcl` (remote stcl) command.

This example shows how to create custom action definitions; however, you can set up most access rights based on the predefined action definitions located in the default access control files. (See Introduction to Access Control for a description of these files.) For a description of the ProjectSync predefined actions, see ProjectSync Action Definitions. For a description of the DesignSync predefined actions, see Revision Control Action Definitions; for server-related predefined actions, see User Authentication Action Definitions and Access Controls for Secure Communications.

Related Topics

Access Control Scripting

Using Access Commands

ProjectSync Action Definitions

Sample Access Controls (DesignSync)

Sample Server Access Controls

Setting Up Access Controls

Access Controls for the Data Replication System

Access Control for Replication

The default DesignSync access control file:

```
$SYNC_DIR/share/AC_Components/AccessControl.ds
```

defines the following actions to control management of the data replication system:

Note: Viewing DRRs on an MAS is controlled by the Access Controls for Browsing the Server.

Important: On CAS-enabled servers with AuthUnm enabled, the Data Replication cannot use the default username and password. If you are using data replication with CAS-enabled DesignSync servers, you must use AuthPwd. For more information on setting AuthPwd, see **User Authentication Access Controls.**

Replicate

Replicate provides browse access to the Replicate menus in the DesignSync Web interface and the replicate commands within the non-graphical DesignSync clients. Users who need to create or manage data replications must be granted Replicate access.

```
access define Replicate
```

ReplicateAdd

ReplicateAdd determines who can add new data replication repository (DRR) definitions.

Note: In addition to this permission, users adding new replications must also have `Replicate` access.

`access define ReplicateAdd {`*`Object Name ReadMode`*`}`

Where:

Object is the path for the 'drr'

Name is name being used for the 'drr'

ReadMode is the read mode specified. Possible values are 'all' or 'group'

For information on setting up Data Replication,, see the *ENOVIA Synchronicity DesignSync Administrator's Guide* : Setting Up Data Replication.

ReplicateData

ReplicateData determines who can add, reset, and enable replication for modules in the DRR.

Note: In addition to this permission, users managing data replications must also have `Replicate` access.

```
access define ReplicateData {<Object> <Selector> <Name> <Root>
<Action>}
```

Where:

Object: is the vault url of the data being replicated

Selector is value of the selector for the data being replicated

Name: is the specified name (if no name is specified, then this is an empty string)

Root: is the name of the DRR.

Action is the permitted actions. If this value is omitted in the access control, the permissions granted or denied apply to all possible actions. the Possible values are

- add - grants/denies permission to add a module to the DRR.
- enable - grants/denies permission to turn on replication for a module in the DRR
- disable - grants/denies permission to turn off replication for a module in the DRR
- reset - grants/denies permission to reset the DRR.

For information on controlling Data Replication, see the *ENOVIA Synchronicity DesignSync Administrator's Guide* : Introduction to Data Replication.

ReplicateRemove

ReplicateRemove determines who can replicated modules from a DRR and remove DRR definitions.

Note: In addition to this permission, users removing replications must also have `Replicate` access.

```
access define ReplicateRemove {<Object> <Root> <Name> <Vault>
<Selector>}
```

Where:

Object is the url of the path to the 'drr'

Root is the name of the 'drr'

Name is the name of the replicated data hierarchy. This option is only applicable to removing a replicated module from a DRR and should be left empty when removing a DRR from a MAS..

Vault is the vault url of the data being replicated. This option is only applicable to removing a replicated module from a DRR and should be left empty when removing a DRR from a MAS..

Selector is the selector of the data being replicated. This option is only applicable to removing a replicated module from a DRR and should be left empty when removing a DRR from a MAS.

For information on removing Data Replications, see the *ENOVIA Synchronicity DesignSync Administrator's Guide:* Cleaning and Removing Data Replication.

Related Topics

Access Controls for Browsing the Server

Overview of DesignSync Project Management

Access Controls for Groups of Data Replication Commands

You may want to grant or deny complete access to data replication actions, to some set of users. Variables defined in the default DesignSync access control file:

```
$SYNC_DIR/share/AC_Components/AccessControl.ds
```

make it more convenient to set the same access rights on data replication actions.

Predefined Variables

The default `AccessControl.ds` file defines the variable $ReplicateActions`,` to control most of the mirror actions:

```
set ReplicateActions "ReplicateAdd ReplicateData ReplicateRemove
Replicate"
```

For example, you could use this variable to ensure that only your Admin can add, edit, delete, modify, or view data replication repositories:

```
access allow $ReplicateActions everyone
access allow $ReplicateActions only users $admin
```

Related Topics

Access Control Scripting

Sample Access Controls

Setting Up Access Controls

Using Access Commands

Access Controls for the Mirror System

Access Controls for Mirrors

The default DesignSync access control file:

```
$SYNC_DIR/share/AC_Components/AccessControl.ds
```

defines the following actions to control management of mirrors:

```
access define Mirrors

access define AddMirror <Object> <MirrorDir> <Vault> <Category>
<Type> <Description>

access define EditMirror <Object> <OldMirrorDir> <NewMirrorDir>
<OldVault> <NewVault> <Category> <Type> <Description>

access define ModifyMirror <Object> <MirrorDir> <Vault>
<Category>

access define DeleteMirror <Object> <MirrorDir> <Vault>
<Category>

access define ViewMirror

access define PrimaryMirrorFetch <Object> [<FullCacheName>]
```

Where:

- `<Object>` is the name of the mirror.
- `<MirrorDir>` is the absolute path of the mirror directory. For example: `/home/projects/mirror/ASIC/layout`.
- `<Vault>` is the URL of the repository vault. For example: `sync://host.mycom.com:5000/Projects/ASIC/layout`.
- `<Category>` is the category defined for the mirror.
- `<Type>` is the type of the mirror, either `Normal`, `Primary`, or `Secondary`.
- `<Description>` is the description given to the mirror.
- `<OldMirrorDir>` is the absolute path of the old mirror directory when you are controlling who can move a mirror. For example: `/home/projects/mirror/ASIC/layout1`.
- <`NewMirrorDir>` is the absolute path of the new mirror directory when you are controlling who can move a mirror. For example: `/home/Projects/ASIC/layout2`.

- `<OldVault>` is the URL of the old repository vault when you are controlling who can move a vault. For example: `sync://host.mycom.com:5000/Projects/ASIC/layout1`.
- `<NewVault>` is the URL of the new repository vault when you are controlling who can move a vault. For example: `sync://host.mycom.com:5000/Projects/ASIC/layout2`.

These parameters are used in `when` clauses in `access allow` and `access deny` statements. (See Using Access Commands for details.)

The `$MirrorActions` variable can be used to control access to most of these actions. See Access Controls for Groups of Commands for details.

The Mirrors Action

The Mirrors action controls whether users can access the mirror commands within the DesignSync web interface or use them within the DesignSync command line clients. Any users who create or maintain mirrors must be granted access to this action.

The AddMirror Action

The `AddMirror` action controls whether users can create mirrors. Users without `AddMirror` access cannot create mirrors using the Create Mirror panel in the ProjectSync GUI and cannot execute the `mirror create` command.

Note: In addition to this permission, users adding new mirrors must also have `Mirrors` access.

The EditMirror Action

The `EditMirror` action controls whether users can edit a mirror definition. Users without `EditMirror` access cannot edit mirrors using the Edit Mirror panel in the ProjectSync GUI and cannot execute the `mirror edit` command.

Note: In addition to this permission, users editing mirrors must also have `Mirrors` access.

The ModifyMirror Action

The `ModifyMirror` action controls whether users can enable, disable, rename, or reset a mirror. Users without `ModifyMirror` access cannot perform these operations from the ProjectSync GUI and cannot execute the `mirror enable`, `mirror disable`, `mirror rename`, or `mirror reset` commands.

Note: In addition to this permission, users modifying mirrors must also have `Mirrors` access.

The DeleteMirror Action

The `DeleteMirror` action controls whether users can delete a mirror. Users without `DeleteMirror` access cannot perform this operation from the ProjectSync GUI and cannot execute the `mirror delete` command.

Note: In addition to this permission, users deleting mirrors must also have `Mirrors` access.

The ViewMirror Action

The `ViewMirror` action controls whether users can view a mirror definition or status. Users without `ViewMirror` access cannot view mirrors or get mirror status from the ProjectSync GUI and cannot execute the `mirror ismirror`, `mirror isenabled`, `mirror list`, `mirror get`, `mirror status` or `mirror wheremirrored` commands.

Note: In addition to this permission, users viewing mirrors must also have `Mirrors` access.

The PrimaryMirrorFetch Action

The `PrimaryMirrorFetch` action controls how secondary mirrors fetch versions from the cache. The optional `FullCacheName` option is reserved for future use.

Note: In addition to this permission, users fetching mirrors must also have `Mirrors` access.

Example

The following access rules ensure that only the project leaders can create a mirror of the ASIC project:

```
access init {
   set projectLeaders { chan kapoor }
}

access deny AddMirror everyone when Vault
"sync://host.mycom.com:5000/Projects/ASIC/*"

access allow AddMirror only users $projectLeaders \
  when Vault "sync://host.mycom.com:5000/Projects/ASIC/*"
```

The first rule ensures that no one can create a mirror for the ASIC project. The second rule builds on the first and allows project leaders, as defined using `access init`, to create mirrors of the project. (See Using Access Commands for details on using `access init` to create custom variables.)

See Setting Up Access Controls for information on creating your own custom `AccessControl` files. See Using Access Commands for a list of the access commands and examples of how to use them. See the `$SYNC_DIR/share/examples/ExampleAccessControl` file for more examples of using access controls with mirrors.

Related Topics

Access Control Scripting

Sample Access Controls

Setting Up Access Controls

Using Access Commands

Access Controls for Groups of Commands

You may want to grant or deny complete access to Mirror actions, to some set of users. Variables defined in the default DesignSync access control file:

```
$SYNC_DIR/share/AC_Components/AccessControl.ds
```

make it more convenient to set the same access rights on Mirror actions.

Predefined Variables

The default `AccessControl.ds` file defines the variable `$MirrorActions,to` control most of the mirror actions:

```
set MirrorActions "AddMirror EditMirror DeleteMirror
ModifyMirror ViewMirror Mirrors"
```

For example, you could use this variable to ensure that only your Admin can add, edit, delete, modify, or view mirrors:

```
access deny $MirrorActions everyone
access allow $MirrorActions only users $admin
```

Related Topics

Access Control Scripting

Sample Access Controls

Setting Up Access Controls

Using Access Commands

Access Controls for DesignSync Projects

Access Controls for DesignSync Projects

The default DesignSync access control file:

```
$SYNC_DIR/share/AC_Components/AccessControl.ds
```

defines the following actions to control management of DesignSync Projects.

DesignSyncProjects

DesignSyncProjects provides browse access to the DesignSync Project menus in the DesignSync Web interface and the sws commands within the OS shell. Users who need to create or manage DesignSync Projects must be granted DesignSyncProjects access.

```
access define DesignSyncProjects
```

By default, DesignSyncProjects access is enabled for everyone.

AddProjectInstance

AddProjectInstance determines who can add a new DesignSync Project instance.

Note: In addition to this permission, users adding DesignSync Project instances must also have `DesignSyncProjects` access.

```
access define AddProjectInstance {Object Vault Path}
```

Where:

Object is the project instance name

Vault is the URL of the vault/module

Path is the path to the project instance

By default, everyone is allowed to add DesignSync Project instances.

ModifyProjectInstance

ModifyProjectInstance determines who can modify a DesignSync Project instance.

Note: In addition to this permission, users modify DesignSync Project instances must also have `DesignSyncProjects` access.

```
access define ModifyProjectInstance {Object Vault Path}
```

Where:

Object is the project instance name

Vault is the URL of the vault/module

Path is the path to the project instance

By default, everyone is allowed to modify DesignSync Project instances.

DeleteProjectInstance

DeleteProjectInstance determines who can delete a DesignSync Project instance.

Note: In addition to this permission, users delete DesignSync Project instances must also have `DesignSyncProjects` access.

```
access define DeleteProjectInstance {Object}
```

Where:

Object is the project instance name

By default, everyone is allowed to modify DesignSync Project instances.

Access Controls for Groups of DesignSync Projects Commands

You may want to grant or deny complete access to DesignSync Projects actions, to some set of users. Variables defined in the default DesignSync access control file:

```
$SYNC_DIR/share/AC_Components/AccessControl.ds
```

make it more convenient to set the same access rights on DesignSync Project actions.

Predefined Variables

The default `AccessControl.ds` file defines the variable $DesignSyncProjectActions, to control most of the mirror actions:

```
set DesignSyncProjectActions "DesignSyncProjects
AddProjectInstance ModifyProjectInstance DeleteProjectInstance"
```

For example, you could use this variable to ensure that only your Project Administrators can access, add, delete, or modify, DesignSync Projects.

```
access allow $DesignSyncProjectActions everyone
access allow $DesignSyncProjectActions only users $ProjAdmin
```

Related Topics

Access Control Scripting

Sample Access Controls

Setting Up Access Controls

Using Access Commands

Access Controls for the Server

Server Action Definitions

Server Action Definitions

You set up access controls on particular **actions**, or operations related to server processes. To set up an access control on an operation, the operation must have an action definition specified with an `access define` command. If an action definition exists for an operation, you can control access to that operation using the stcl `access allow`, `access deny`, and `access filter` commands. You can control access to operations on module data by using the `access decline` command.

DesignSync provides predefined actions corresponding to most server operations you might want to access control. These actions are defined in the default access control file for sever authentication:

```
$SYNC_DIR/share/AC_Components/AccessControl.auth
```

See Introduction to Access Control for details on the individual files used to define access controls.

Important: Do not edit the `$SYNC_DIR/share/AC_Components/AccessControl.auth` file; changes will be lost upon upgrading. Instead, edit your site or server custom `AccessControl` file (see Setting Up Access Controls).

Server Actions Governed by access controls

To control access to this operation...	Customize this access control...
`suspend`	`suspend` See Suspending Server Activity Access Control for details.
Server User Authentication	See User Authentication Access Controls and User Authentication Action Definitions for details.
Non-SSL Server Authentication	See Access Controls for Secure Communications for details.

Related Topics

ProjectSync Action Definitions

DesignSync Action Definitions

Modules Action Definitions

Suspending Server Activity Access Control

The default server-level access control file:

```
$SYNC_DIR/share/AC_Components/AccessControl.auth
```

defines a single access-control actions for suspending server activity.

Do not edit the `AccessControl.auth` file; changes will be lost upon upgrading. Instead, edit your site or server custom `AccessControl` file (see Setting Up Access Controls).

The access control action for suspending server activity is:

```
access define Suspend {Mode}
```

By default, all users are granted suspend access:

```
access allow Suspend everyone
```

Note: If you restrict the suspend access and you run automated backups, you must allow suspend access for the user 'nobody,' the "user" who runs the automated backups. For more information see the *ENOVIA Synchronicity DesignSync Administrator's Guide*: Procedure for Defining Automatic Backups

Example

This examples allows access for 'nobody' who runs the automated backup scripts, and the syncmgr account.

```
access allow Suspend only users {nobody,sysmgr} -because "Only
Administrators are allowed to suspend the server"
```

User Authentication Access Controls

You can use access controls to define the level of user authentication required to access project data on a SyncServer. You cannot set up user authentication to control access to client vaults.

User authentication is defined on a per-SyncServer basis. The first time a user tries to access server data, the user must satisfy the level of authentication required by that server. Authentication takes place each time a user starts a new server session or accesses a different server.

The default server-level access control file is:

```
$SYNC_DIR/share/AC_Components/AccessControl.auth
```

For DesignSync, the default access controls allow username-only authentication for any user, from any IP address. For ProjectSync, the default access controls allow username/password authentication for any user, from any IP address.

DesignSync provides the following access-control actions for user authentication:

- `AuthUnm` - Controls username-only authentication.

  In username-only mode, users are never prompted for authentication information. The user's login account name is automatically passed to the server. Users do not need a ProjectSync user profile on the server for username-only access.

  ProjectSync does not allow username-only authentication.

  If `AuthUnm` is allowed only for certain users and if a user's operating system login name is not on the allowed list, then that user is prompted to log in. The login username and password must match the username and password in that user's ProjectSync user profile.

- `AuthPwd` - Controls username/password authentication.

  In username/password mode, users are prompted for their username and password the first time they attempt to access a server. The user's username and password must correspond to a ProjectSync user profile. All users accessing the server must have user profiles, as created from ProjectSync, on that server.

- `AuthAll` - Lets you set the same access control for both username-only and username/password authentication.

  For both username-only and username/password authentication, you can further restrict access to a server based on the IP address of the users' machines.

The sample access control file, `<SYNC_DIR>/share/examples/ExampleAccessControl`, contains examples of user-authentication access controls. See Sample Server Access Controls for additional examples.

Related Topics

Sample Server Access Controls

Setting Up Access Controls

User Authentication Action Definitions

Using Access Commands

User Authentication Action Definitions

The default server-level access control file:

`$SYNC_DIR/share/AC_Components/AccessControl.auth`

defines the access-control actions for user authentication.

Do not edit the `AccessControl.auth` file; changes will be lost upon upgrading. Instead, edit your site or server custom `AccessControl` file (see Setting Up Access Controls).

The following are the access-control action definitions for user authentication:

- Username-only authentication:

  ```
  access define AuthUnm {client_ip}
  ```

- Username/password authentication:

  ```
  access define AuthPwd {client_ip}
  ```

- Both username-only and username/password authentication.

  ```
  set AuthAll {AuthUnm AuthPwd}
  ```

  The `AuthAll` variable lets you set the same access control on both `AuthUnm` and `AuthPwd` actions.

where `<client_ip>` is the IP address for the client you want to grant or deny access.

You can use these actions in your custom `AccessControl` file to specify how users are authenticated.

Example

The following access rules ensure that only users from a trusted IP address can access the DesignSync server:

```
access deny $AuthAll everyone
access allow AuthPwd everyone when client_ip 10.1.1.*
```

In this example, the first rule denies access to all users for all types of authentication. The second rule then opens username and password access to all users who access the server from an IP address beginning with `10.1.1`.

See the sample access control file, `$SYNC_DIR/share/examples/ExampleAccessControl`, for more examples of how to set up user authentication. See Sample Server Access Controls for additional examples.

Related Topics

Access Control Scripting

DesignSync Action Definitions

Sample Access Controls

Sample Server Access Controls

Setting Up Access Controls

User Authentication Access Controls

Using Access Commands

Access Controls for Secure Communications

The default server-level access control file:

```
$SYNC_DIR/share/AC_Components/AccessControl.auth
```

defines a single action to control cleartext (Non-SSL) communications:

```
access define NonSSL <client_ip>
```

Where `<client_ip>` is the IP address of the system accessing the ProjectSync server.

By default, SyncServers allow cleartext (non SSL) communications for every user, from any IP address. When the non-SSL access control is set and communications require the use of a secure port, the SyncServer automatically redirects the communications to the SSL port.

Example

The following access rules require that communications from outside networks be SSL (secure) communications, except for communications from a local network address:

```
access deny NonSSL everyone

access allow NonSSL everyone when client_ip 10.1.1.*
```

The first rule ensures that all system accessing the ProjectSync server must use SSL. The second rules builds on the first to allow cleartext access from within the local network.

See Sample Server Access Controls for additional examples.

For additional information on secure communications, see *ENOVIA Synchronicity Administrator's Guide*: Overview of Secure Communications.

See Setting Up Access Controls for information on creating your own custom `AccessControl` files. See Using Access Commands for a list of the access commands and examples of how to use them.

Related Topics

Access Control Scripting

DesignSync Action Definitions

Sample Access Controls

Sample Server Access Controls

Setting Up Access Controls

User Authentication Access Controls

Using Access Commands

Example Server Access Controls

Sample Server Access Controls

Sample access controls are provided in:

```
$SYNC_DIR/share/examples/ExampleAccessControl
```

You can use this sample `ExampleAccessControl` file as a template for creating access control files.

This topic provides examples of server-level access controls and scripts that you might put in your custom `AccessControl` file:

Setting Up User Name and Password Authentication

Requiring Log Ins and Restricting Revision Control to Project Leaders

Denying or Allowing Access by Users

Granting Access to Specific Clients

Restricting Operations Based on Client IPs

Setting Up User Name and Password Authentication

The following statement in your site or server `AccessControl` file turns on user name and password authentication. The server's administrator creates a ProjectSync user profile for each user who is allowed access to the server. Only users whose name and password match a user profile are allowed to access the server.

```
access deny AuthUnm everyone
```

Requiring Log Ins and Restricting Revision Control to Project Leaders

Including the statements below in your site or server `AccessControl` file restricts server access to a selected list of users. Then, of those users, only those listed as project leaders can perform revision control operations.

```
# Define a list of privileged users
access init {
  set projectLeaders user1 user2 user3
}

# Require everyone to log in based on their ProjectSync
# User Profile by preventing user name-only authentication
access deny AuthUnm everyone

# Once successfully logged in, only project leaders can
# access the server for revision control operations
access allow $DesignSyncActions only users $projectLeaders
```

Denying or Allowing Access by Users

To use the `purge` command, users must have access to the `Delete` action when the type is `VERSION`. To set up this access, enter an access control like the following in your site or server `AccessControl` file:

```
# Only users specified on the delversusrs list can delete
versions
access init {
set delversusrs [list syncmgr jane tjones]
}
access allow Delete only users $delversusrs when Type VERSION
```

This statement permits only the users syncmgr, jane, and tjones to delete versions,

Granting Access to Specific Clients

The following statements in your site or server `AccessControl` file turn off all access to a server and then selectively grant access to specific clients:

```
access deny AuthAll everyone
access allow AuthPwd everyone when client_ip 10.1.1.*
```

Restricting Operations Based on Client IPs

Problem

To secure your server's data, you might want to restrict particular operations based on the client's IP address, specified by the <`client_ip`> parameter. However, most actions you might try to restrict, such as `Checkin` and `Checkout`, do not pass the <`client_ip`> parameter; only the `AuthUnm` and `AuthPwd` actions pass the <`client_ip`> parameter.

Solution

The following script restricts check-ins and checkouts on objects within the Asic project to the client whose IP address is 9.9.9.9:

```
# Allow only IP number 9.9.9.9 for project Asic
access filter {Checkin Checkout} {
 puts "CLIENTIP is $::SyncAC::CLIENTIP, Object is $Object"
 if {[string match *:///Projects/Asic/* $Object]} {
     if {![string match 9.9.9.9 $::SyncAC::CLIENTIP]} {
         return DENY
     }
 }
 return UNKNOWN
}
```

```
# Set up filter for AuthUnm which sets a var CLIENTIP
access filter [ list AuthUnm AuthPwd ] {
set ::SyncAC::CLIENTIP $client_ip
return UNKNOWN
}
```

Discussion

The `AuthUnm` and `AuthPwd` access filter is a user and password authorization access filter that passes the <`client_ip`> variable to the filter script. This access filter script creates the `CLIENTIP` global variable within the `SyncAC` namespace (denoted with the `::SyncAC` Tcl syntax) . You use the `::SyncAC` namespace rather than the global namespace (denoted with the `::` Tcl syntax) because the global namespace is reset upon every request whereas the `SyncAC` namespace persists.

Note that this `AuthUnm` and `AuthPwd` access filter must be the last access filter processed for the `AuthUnm` and `AuthPwd` actions in order for the `CLIENTIP` global variable to be set. All access filters with side effects such as setting this global variable must be the last filter of their type processed. Thus, you might want to include a filter with side effects such as this at the end of the server AccessControl file, rather than the site AccessControl file since the server AccessControl has precedence over the site AccessControl file.

Also note that `UNKNOWN` is used instead of `ALLOW`.

The `Checkin` and `Checkout` access filter script checks this `CLIENTIP` global variable against the allowed client IP address, 9.9.9.9. The script checks whether the <`Object`> (passed from the `Checkin` and `Checkout` actions) is included in the Asic project.

You can enhance this script by setting up an `access init` statement to specify a list of valid client IP addresses:

```
access init {
  set ValidClientIPs "9.9.9.9 20.6.5.3"
}
```

You can then use a Tcl `lsearch` statement within the access filter script to verify that the <`client_ip`> parameter is included in the <`ValidClientIPs`> namespace variable.

Access Control for ACAdmin

ACAdmin Action Definitions

Access Controls for Groups of Commands

You might want to grant or deny complete access to ACAdmins actions for users or groups who set up and maintain ACAdmin definitions. ACAdmin provides a a command group that contains all the Access rights needed for full access to ACAdmin functionality.

You also can create your own variables for sets of commands in your custom `AccessControl` file.

Predefined Command Groups

The default command groups defines these command to control ACAdmin actions:

```
set ACAEditActions {AcaCatPrmDef AcaCmdCatDef AcaCmdDef
AcaCmdFilterDef AcaUserGroupDef}

set ACAActions "$ACAEditActions AcaViewDef ResetAccessControls"
```

Another variable, `$AllActions`, controls all ProjectSync actions, as well as all DesignSync and module actions that have an Object parameter (`$CmdActions)`. The `$AllActions` variable is defined in the `AccessControl.ps` file. See Access Controls for Groups of Commands in the ProjectSync section for details.

Custom-Defined Variables

In your custom `AccessControl` file, you can use the `set` command to create variables that represent your own groupings of access control actions.

For example, to create a variable that controls the ability to move and remove module members, you could specify:

```
set ModuleMoveActions {MemberRename MemberRemove}
```

You can then use the variable `$ModuleMoveActions` in your access control rules.

You also can create variables that include other variables. However, when declaring a list that requires variable substitution, you must enclose the list in quotation marks, not curly braces. (Curly braces prevent variable substitution.) For example:

```
set ModuleDesignerActions "$CmdActions $ModuleMoveActions Tag"
```

Related Topics

Modify User Groups

Modify Command Category

Customizing ACAdmin

Access Control Commands

Access Command Details

The following table lists the access commands. Select a linked access command name to view the description of the command.

Action Definition Commands

access define	Defines new actions to be access controlled. Most actions are predefined in the `AccessControl` file included with DesignSync.
access init	Defines namespace variables and procedures for use in `access filter` statements.
access global	Defines global variables and procedures that can be called within `access filter` statements.
set <group_action>	Defines a variable used to represent a group of individual actions.

.

Access Rule Commands

access allow | deny	Allows or denies access to a specified list of actions.
access decline	Causes additional access rules to be invoked.
access db_filter	Allows or denies access to viewing or editing notes based on specified filters.
access filter	Allows or denies access based on specified filters. (Use `access db_filter` for viewing or editing more than one note.)

Access Commands Used Outside the AccessControl File

access reset	Updates access controls without forcing you to stop and restart the SyncServer.
access verify	Checks whether the specified user is allowed to perform an action.

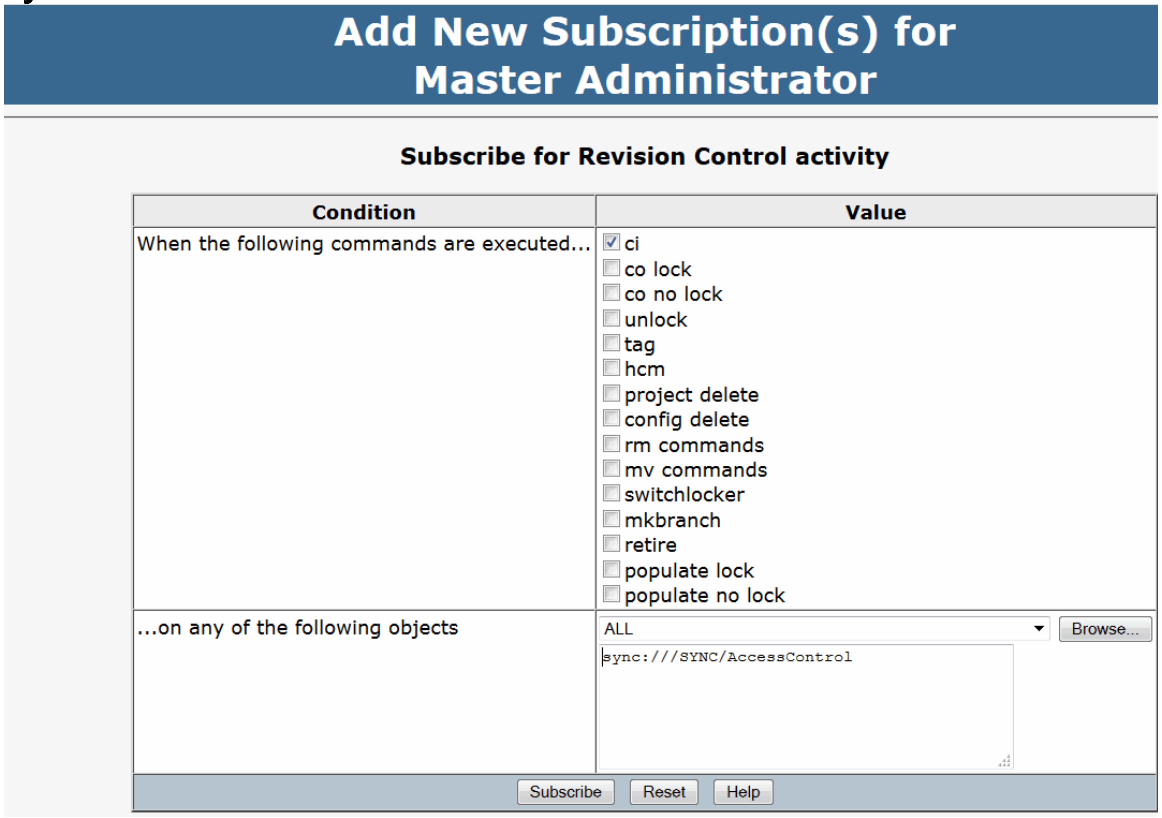
Secure Access Control

Subscribing to Secure Access Control Revision Control Notes

Whenever an Access Reset is performed, a new version of the consolidated rules set is checked in to the Access Control Vault, and ‘ci’ Revision Control (RC) Note will be created. You can subscribe to this RC note to receive an email update whenever the Access Control vault is updated.

Subscribing to Secure Access Control RC Notes:

1. Open the DesignSync WebUI and log in as the administrator or user with permission to create RC note subscriptions.
2. Select the **Admin Men**u and open the U**ser Profiles** section.
3. Select **Email Subscriptions** | **Add New Subscriptions.**
4. Select the following options as shown in the image
   When the following command are executed... **ci**
   ...on any of the following objects
   sync:///SYNC/AccessControl



Add New Subscription(s) for
Master Administrator

Subscribe for Revision Control activity

Condition	Value
When the following commands are executed...	☑ ci ☐ co lock ☐ co no lock ☐ unlock ☐ tag ☐ hcm ☐ project delete ☐ config delete ☐ rm commands ☐ mv commands ☐ switchlocker ☐ mkbranch ☐ retire ☐ populate lock ☐ populate no lock
...on any of the following objects	ALL sync:///SYNC/AccessControl

   Tip: If you Browse for The Access Control vault, it is located in the **SYNC** object.
5. Click **Subscribe** to save the subscription.

Unsubscribing to Secure Access Control RC Notes:

1. Open the DesignSync WebUI and log in as the administrator or user with permission to create RC note subscriptions.
2. Select the **Admin Men**u and open the U**ser Profiles** section.
3. Select **Email Subscriptions.**
4. Your subscription will appear in the Subscription status list. Click the checkbox next to **Delete**.
5. Submit Deletions to remove the subscription.

Enabling/Disabling Secure Access Control

Secure Access Control is enabled when the server is started or reset with a .ac file in the Access Control directory. When DesignSync sees this file, it starts Access Control in secure mode.

IMPORTANT: The permissions on the file should be set so that only authorized administrators can remove the file.

To enable secure access control, create the .ac file in the syncdata directory:

```
$> touch
syncdata/<host>/<post>/server_vault/SYNC/AccessControl/.ac
```

where:

- host is the simple server name of the server.
- port is the port number configured for the server.

To disable secure access control, remove the .ac file from the syncdata directory:

```
$> rm
syncdata/<host>/<post>/server_vault/SYNC/AccessControl/.ac
```

where:

- host is the simple server name of the server.
- port is the port number configured for the server.

Format of Consolidated Rule Set

The consolidated rules set includes all the components of the AccessControl files:

- access definitions
- global procedures/variables
- initialization procedure/variables
- allow/deny/decline/filter/db_filter rules

The majority of the changes to the access control rules will be made to the original files and the system will then create the consolidated file, however the format of the file is provided here for reference purposes.

The consolidated rule set users the following format:

```
<GLOBALS SECTION>

<ACTION SECTION> [...]
```

Globals

The Globals section contains any global procedures or variables defined using the `access globals` coimmand.

Actions

Each action section includes the following:

- Name of the Action
- Access Definition, including a comment showing the full path of the file from which the definition comes.
- Initialization Procedures, including a comment showing the full path of the file from which this initialization procedure comes.
- Rules for Processing including the full path of the file from which the rule comes, and, if the access control rule loads another file, the path to that file as well.

The order of the set of rules as read from the original Access Control files is significant and is preserved in the consolidated rule set so that the rules are correctly applied.

Related Topics

Sample Consolidated Rule Set

Sample Consolidated Rule Set

This is a sample of what the consolidated rule set file might look like. It is not intended as an example of how you should set up your Access Control rules. Your access control rules should be set up as usual, in the custom Access Control files..

```
# Source the common utilities for access db_filter

# the individual acLeaves files also defined there

foreach ACtcl [locate -all -reverse share/tcl/AC.tcl] {

    source $ACtcl

}

# File: /home/syncadmin/syncinc/linux_a64/share/tcl/AC.tcl

access global {
  # Setup db_filter API for all actions
  access init {
    if { [info exists __accName] } {
      # Alias all API functions
      set api [list ALLOW_ALL DENY_ALL ALLOW DENY FOREACH_NOTE
FOREACH_UNKNOWN SINGLE_NOTE]
      foreach fn$api {
        interp alias {} $fn {} ::SyncAC::$fn $__accName
      }
      set api [list GET_NOTES FILTERED_IDS]
      foreach fn $api {
        interp alias {} $fn {} ::SyncAC::$fn
      }
    }
  }
}

#########################
# Action: MemberCheckoutLock
#########################

# File:
/home/syncadmin/syncinc/linux_a64/share/AC_Components/AccessCont
rol.hcm

access define MemberCheckoutLock {Object NaturalPath}

# File:
/home/syncadmin/syncinc/linux_a64/share/AC_Components/AccessCont
rol.hcm

access allow MemberCheckoutLock  everyone
```

```
#########################

...

#----------------------------

# Action: Mkmod

#-------------------------------

# File:
/home/syncadmin/syncinc/linux_a64/share/AC_Components/AccessCont
rol.hcm

access define Mkmod {Object}

# File:
/home/syncadmin/syncinc/linux_a64/share/AC_Components/AccessCont
rol.hcm

access allow Mkmod  everyone

# File:
/home/syncadmin/syncinc/linux_a64/custom/servers/serv1/2647/shar
e/AccessControl.freeze

access deny Mkmod  everyone when Object
{sync:///Modules/SYNC/DevelopmentSettings/Templates/ENOVIA/Analo
g;*} -because {The module is frozen so cannot be modified.}

#----------------------------

...
```

Troubleshooting Access Controls

Troubleshooting Access Controls

The following sections contain hints for debugging problems with your custom access controls

Resetting Access Controls

When you customize the custom `AccessControl` files, you must reset the access controls to reapply the changes. So, if you have recently modified an `AccessControl` file and you do not see the changes, reset your access controls using one of the following methods:

- Execute the `access reset` command in a server-side Tcl script.
  An `access reset` lets you update access controls without forcing you to stop and restart the SyncServer
- Click the **Access Reset** option in the ProjectSync menu if you are a ProjectSync user.
  The **Reset Server** option (in the ProjectSync menu) also updates access controls, along with other customizations.

  Note: These options appear in the ProjectSync menu only if you have permissions to reset the access controls. (See Access Controls for Server Administration for details.)

  Since AccessControl files are also read when server child processes are spawned, you must **Access Reset** or **Reset Server** after modifying custom access controls. Otherwise the end users might see inconsistent access control behavior.
- Stopping and restarting the SyncServer.

Checking Access Control File Syntax

If your access control rules are not working, review the syntax and check for typos. The elements of `AccessControl` files are case sensitive, so take care when typing in command strings.

If one of your custom `AccessControl` files contains a syntax error, you see an Operation Failed panel with some Tcl stack information. This information includes path to the `AccessControl` file and the line number where the error was encountered. For example:

```
Tcl Stack:
access deny: action everyone is not defined
   while executing
"access deny everyone AddUser"
   (file "/u1/ProjSync/custom/site/share/AccessControl" line
368)
```

ProjectSync sometimes passes in the asterisk (`*`) qualifier to the `access verify` command when the value of a parameter is not yet known; therefore, your `access` commands must account for this value. See Using Access Command Qualifiers for more information.

See Setting Up Access Controls for information on creating your own custom `AccessControl` files. See Using Access Commands for a list of the access commands and examples of how to use them.

Reviewing Log Files

When you first create or later change a custom `AccessControl` file, check for errors in the error log files after invoking the server for the first time. You can check both the server metadata log file and the main Apache server log file.

Server Metadata Log File

You can find access control error messages in the SyncServer's error_log file. The default location for `<ServerMetaDataRoot>` is:

```
$SYNC_DIR/../syncdata/<host>/<port>/server_metadata
```

If the log file is not at the above default location, look in the `PortRegistry.reg` file for the server metadata directory:

```
$SYNC_CUSTOM_DIR/servers/<host>/<port>/PortRegistry.reg
```

This file contains the registry key:

```
HKEY_LOCAL_MACHINE/Software/Synchronicity/Directories/ServerMeta
DataRoot
```

The value of this key is the location of the server metadata directory containing the log file.

Apache Server Log File

The SyncServer is built on top of the Apache server. Therefore, access control error messages also are written to the default Apache server log file:

```
$SYNC_CUSTOM_DIR/servers/<host>/<port>/logs/error_log
```

Using Filter Scripts to Get Access Control Parameter Values

If an access control is not behaving as intended, you can print out the values of the parameters of the access control action.

For example, the following access control rules are intended to allow only users in the `teamleaders` group to check out and lock a `Makefile` for editing.

```
access init {
  set teamleaders {george linda}
}

access allow Checkout only users $teamleaders \
  when Object "*/Makefile" \
  when Lock "yes"
```

Unfortunately, this `access allow` rule fails to prevent team members who are not team leaders from checking out `Makefile` objects with locks. To discover why the access rule is not working, use the `access filter` command to print the values of the `Checkout` action parameters to the SyncServer's error_log file.

For the `Makefile` example, the filter used to print out the `Checkout` parameters is:

```
access filter Checkout {
  puts "Lock = $Lock"
  puts "Object =$Object"
  return UNKNOWN
}
```

Note that the filter returns `UNKNOWN` instead of `ALLOW`. The `ALLOW` return value overrides other rules. Instead, the `UNKNOWN` return value simply does not deny the action; `UNKNOWN` defers to existing access control results for the action.

After resetting the access controls to apply the access filter above and having a team member who is not a team leader successfully check out a `Makefile` with a lock, the SyncServer's error_log file contains the following results:

```
Wed Nov 7 14:49:18 2001 | AccessControl: Lock = yes

Wed Nov 7 14:49:18 2001 | AccessControl: Object
=sync:///Projects/Asic/Makefile;1.1
```

Notice that the "`Object`" value printed to the SyncServer's error_log file is of the form "`sync:///Project/<ProjName>/Makefile;1.1`". Because the "`;1.1`" version is specified at the end of the object's path, the `when` clause of the Checkout access control (`when Object "*/Makefile"`) does not match the actual `Makefile` object being checked out.

To fix the `Checkout` access control, the object value must be edited as follows:

```
access allow Checkout only users $teamleaders \
  when Object "*/Makefile;*" \
  when Lock "yes"
```

Related Topics

access filter

Setting Up Access Controls

Using Access Command Qualifiers

Using Access Commands

Additional Information

Command Buttons

Button	Description
Reset	Resets the settings on the form back to their original default values and clears any text fields.
Submit	Sends the information on the form to the server.
Back	Reloads the previous page without processing any of the settings in the dialog.
Help	Invokes help information for the dialog in a separate browser window.
Show [type]	Shows you a list of existing user groups or command categories, as appropriate for the calling page.

Getting Assistance

Using Help

ENOVIA Synchronicity DesignSync Data Manager Product Documentation provides information you need to use its products effectively. The Online Help is delivered through WebHelp® , an HTML-based format.

When the Online Help is open, you can find information in several ways:

- Use the **Contents** tab to see the help topics organized hierarchically.
- Use the **Index** tab to access the keyword index.
- Use the **Search** tab to perform a full-text search.

Within each topic, **Show** and **Hide** buttons toggle the display of the navigation (left) pane of WebHelp, which contains the Contents, Index, and Search tabs. Hiding the navigation pane gives more screen real estate to the displayed topic. Showing the navigation pane givens you access to the Contents, Index, and Search navigation tools.

You can also use your browser navigation aids, such as the **Back** and **Forward** buttons, to navigate the help system.

Related Topics

Getting a Printable Version of Help

Getting a Printable Version of Help

The *ENOVIA Synchronicity Access Control Guide* is available in book format from the ENOVIA Documentation CD or the DSDocumentation Portal available on the 3ds support website (http://media.3ds.com/support/progdir/). The content of the book is identical to that of the help system. Use the book format when you want to print the documentation, otherwise the help format is recommended so you can take advantage of the extensive hyperlinks available in the DesignSync Help.

You must have Adobe® Acrobat® Reader™ Version 8 or later installed to view the documentation. You can download Acrobat Reader from the Adobe web site.

Contacting ENOVIA

For solutions to technical problems, please use the 3ds web-based support system:

http://media.3ds.com/support/

From the 3ds support website, you can access the Knowledge Base, General Issues, Closed Issues, New Product Features and Enhancements, and Q&A's. If you are not able to solve your problem using this information, you can submit a Service Request (SR) that will be answered by an ENOVIA Synchronicity Support Engineer.

If you are not a registered user of the 3ds support site, send email to ENOVIA Customer Support requesting an account for product support:

enovia.matrixone.help@3ds.com

Related Topics

Using Help

Index