cādence®

Cadence Application Infrastructure User Guide

Product Version IC23.1
November 2023

Printed in the United States of America.

Cadence Design Systems, Inc. (Cadence), 2655 Seely Ave., San Jose, CA 95134, USA.

Disclaimer: Information in this publication is subject to change without notice and does not represent a commitment on the part of Cadence. Except as may be explicitly set forth in such agreement, Cadence does not make, and expressly disclaims, any representations or warranties as to the completeness, accuracy or usefulness of the information contained in this document. Cadence does not warrant that use of such information will not infringe any third party rights, nor does Cadence assume any liability for damages or costs of any kind that may result from use of such information.

Cadence is committed to using respectful language in our code and communications. We are also active in the removal and replacement of inappropriate language from existing content. This product documentation may however contain material that is no longer considered appropriate but still reflects long-standing industry terminology. Such content will be addressed at a time when the related software can be updated without end-user impact.

Contents

6

Cadence Data Registry File: data.reg

7

Name Mapping

1

Cadence Application Infrastructure

This section describes a set of mechanisms common to Cadence® applications.

The following mechanisms support consistent operations between applications.

- Name mapping between applications: The Cadence approach to name mapping, applied when applications use data from other applications with non-compatible naming conventions. Each Cadence application interprets names according to a consistent set of rules. Data is interoperable across many applications and data formats. Also, data is interoperable between UNIX and NT operating systems.
- Generic Design Management (GDM) facility: It is an interface between Cadence applications and any design management system. This facility implements an interface that CAD applications, which make direct DM system calls, can use so that they can work with many different design management systems without having any special knowledge of those systems. Part of the GDM facility is a set of shell commands that can be used in running design management operations. These commands are also used by some Cadence applications that perform these operations from shell scripts.
- Copying Libraries: Cadence provides the cdsCopy system to help you copy, merge, or rename libraries. A cdsCopyShell, a set of SKILL functions, and a cross-reference updater system are part of cdsCopy.
- Cadence Locking System (CLS): It is the mechanism used by Cadence applications to lock files on all platforms. A Lock-Stake file (fileName.cdslck) in the same directory as a file indicates that the file is locked. Applications release locks when they no longer need them. CLS also provides an administrative tool that lets you view and release any locks.
- Occurrence property dictionary: It is a central repository for simulation-control property definitions.
- Cadence Library Structure: It uses your computer directory structure to organize data. Data is stored in files in a directory hierarchy. Most Cadence applications use the same library model. The library structure provides data interoperability and consistent behavior among applications—a common library model using common file-naming conventions.

Related Topics

Name Mapping

Generic Design Management (GDM) Commands

cdsCopy

Occurrence Property Dictionary

Cadence Locking System

Cadence Library Structure

Cadence Application Infrastructure Files

Various configuration mechanisms are used in Cadence application, such as defining data types, locating libraries, configuring key capabilities. These configuration mechanisms are controlled by specific file formats and associated utilities.

- The Cadence Setup Search File (`setup.loc`)
- The Cadence System Information File (`cdsinfo.tag`)
- The Cadence Library Definition File (`cds.lib`)
- The Cadence Data Registry File (`*.reg`)

The Cadence Setup Search File

This file supports a configuration to search for application setup and configuration information.

The Cadence System Information File

This file supports configuration of several key capabilities, including the type of design management system used to manage a library and whether a more strict library checking mechanism should be used to identify Cadence libraries.

The Cadence Library Definition File

This file supports defining the locations of Cadence libraries and models related to organizing library definition files so that teams of users can consistently share such definitions.

The Cadence Data Registry File

This file supports defining various data types (especially views), associating default editors with data, and is also a general registry facility.

Related Topics

Cadence Setup Search File: setup.loc

Cadence System Information File: cdsinfo.tag

Cadence Application Infrastructure Commands

The following commands are useful in configuring data files, debugging files, and in reporting the status of the Cadence application infrastructure.

Command	Usage
`cdswhich`	A command available as a test, debug, and administration tool for use in managing the search mechanism.
`cdsinfo`	A command used as a test, debug, and administration tool for the system information file mechanism.
`nmp`	A command that assists you in determining the difference in a name as it is mapped to different name spaces.
`cds_root`	A utility, typically used in startup scripts, for identifying the location of Cadence installation hierarchies.
`dregprint`	A command that displays the contents of the data registry.
`cdsLibDebug`	A command that helps you to test and debug your library definition files. `cdsLibDebug` is only available with Virtuoso Studio design environment applications.
`clsAdminTool`	An administrative tool for viewing and releasing file locks.

Related Topics

The cdswhich Command

The cdsinfo command

The nmp Command

The cds_root Utility

The dregprint Command

The cdsLibDebug Command

CLS Administrative Tool

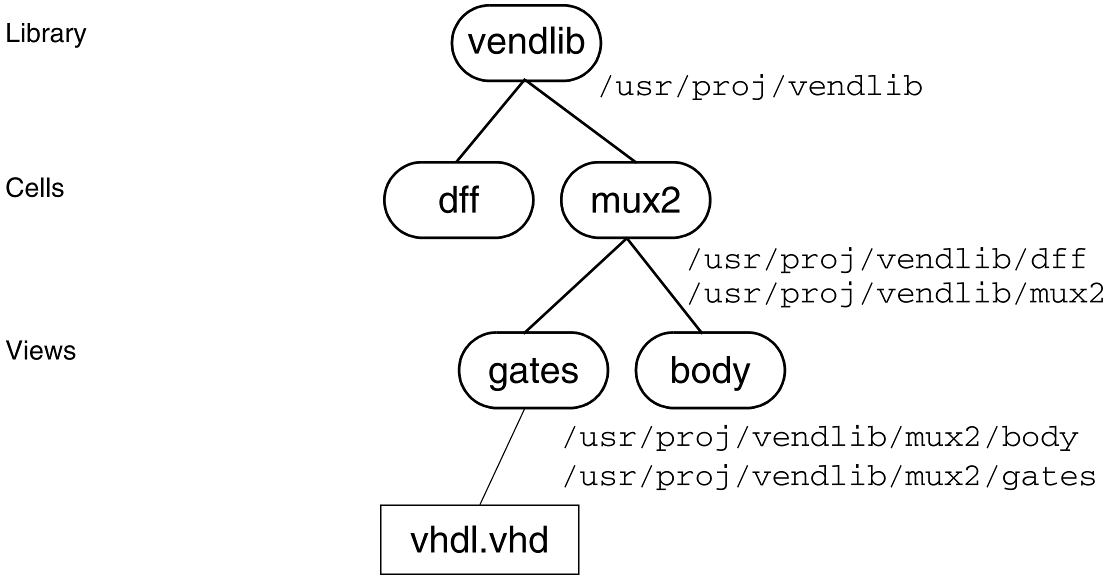
2

Cadence Library Structure

The Cadence library structure is the standard structure in which Cadence applications organize design data. The structure stores data in directories based on the UNIX file system.

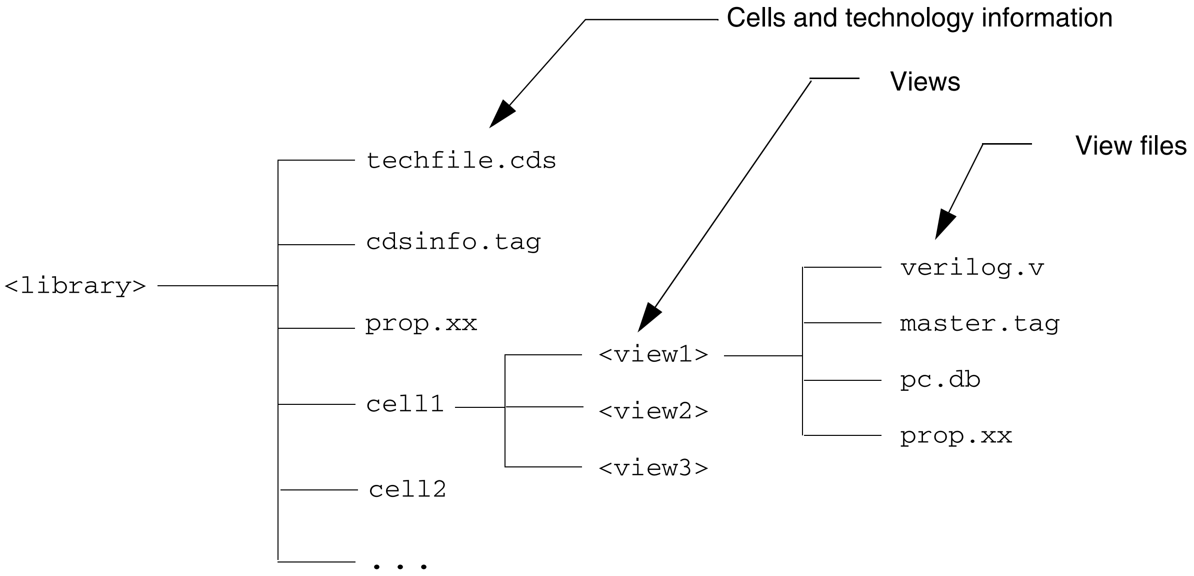
The hierarchy of these directories starts with a library, which is a directory containing both files and other subdirectories. The subdirectories of a library are cells and these cells also contain files or other subdirectories. The subdirectories of cells are views of the cell.

Files are not hidden. You can find the UNIX or NT file that contains the data you need as follows:



Libraries

A library consists of cells, views, and technology information.



A library is a logical collection of design data implemented as a physical collection of directories and files that can reside anywhere in the file system. A library can be shared by all users or controlled by a single person.

Reference libraries store data used by many designs. Whereas, Design libraries or working libraries store data of a single design and derived data about that design.

Reference libraries have read-only access to avoid accidental modification of the master building blocks. Design libraries have read and write access so you can edit and save designs.

A library can have a temporary directory associated with it. For more information, see Temporary Directory for a Library.

In the Cadence infrastructure, each library must have an entry in the `cds.lib` file. You can find the `cds.lib` file by using the setup search mechanism. The search mechanism has a verification option.

Cells and Views

Each library has associated data called cells, and each cell has associated data called views.

- A cell is a collection of views that describe an individual building block of a chip or system, such as a `nand2` gate. Each cell within a library is a separate file system directory with a unique name.
- A view is a collection of files that are related. Each view within a cell is a separate file system directory in which Cadence locates all of the files pertaining to a particular view of a given cell.

You might use different views to represent different levels of abstraction of a design (behavior, gate) or different stages in the design process (rtl, postsynthesis). Views can also be used to contain different types of data about a cell (schematic, symbol, layout, vhdl).

Related Topics

Cadence Setup Search File: setup.loc

Cadence System Information File: cdsinfo.tag

master file

derived data

derived files

also-managed files

View Files and a Co-Managed Set

Views are subdirectories of a cell and each view might contain a variety of files such as the master file specified by the `master.tag` file, the co-managed set of data specified in the data registry, and perhaps derived files and also-managed files.

The application you use determines the names of the files under the view directory. Files under view directories created by Cadence applications are controlled by the application, not by the user. View files are stored in a view directory. Some common view files are as follows:

- Master files
- Co-master files
- Derived files

Co-managed set

- master.tag file
- Also-managed file

Master Files

Master files contain the primary data, such as schematic database or the Verilog source for a view.

Co-Master Files

Co-master files contain information that is not the primary data, but which cannot be derived from the primary data, such as the view property files created by Virtuoso applications and stored in the `data.dm` file.

Derived Files

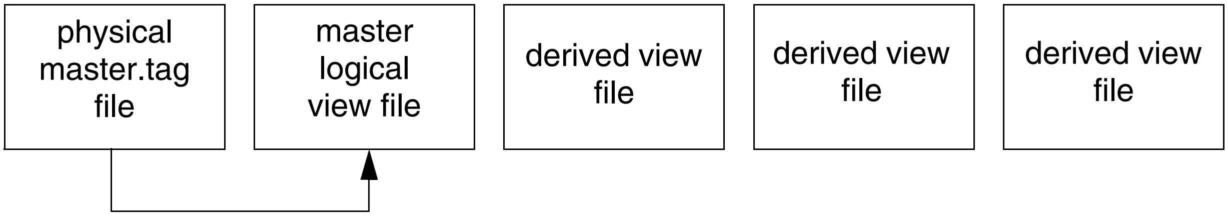
Derived files contain data derived from source data, such as an `ast` file for Verilog or the `pc.db` file.

A Co-Managed Set

The master file, co-master files, and derived files together make up as a co-managed set for the view. Many operations, especially design management operations, need to treat these files as a set.

master.tag File

For certain operations, such as launching an editor on view data, the master file must be identifiable. This is done algorithmically based on the presence of various files in the view directory as described below.



The `master.tag` file, if present, defines which view file inside each cell directory is the master. It records which physical file is the master logical view file for any given view. A given view always has one master representation and zero or more derived representations. When derived representations exist for a view, some applications might need additional information in the library to indicate which data is master and which data is derived.

The following rules apply in the order listed to determine the master representation:

- If the `master.tag` file exists, the file specified by the `master.tag` file is the master representation.
- If there is only one file in the directory, it is the master.
- If a file named `vhdl.vhd` or `verilog.v` is present in the directory, it is the master.

Also-Managed Files

In several cases, it is appropriate to create additional data that is stored in files located under the library, cell, or view directories. These files are not members of a co-managed set, but need to be considered for some design management operations. For example, these files should be checked in or checked out along with other library data when the entire library is checked in or checked out. This checkout procedure is handled by identifying these files to the data registry as also-managed files.

Related Topics

Cadence Data Registry File: data.reg

Temporary Directory for a Library

A library can have a temporary directory associated with it to store derived data from the source library. This is useful if you want to create derived data for a read-only reference library in another directory. Applications reading a library that has a temporary directory assume that the library contains combined contents of the source library and the temporary directory. If there are any files in common, the files in the temporary directory take precedence.

A temporary directory can contain only derived data; it cannot contain source data. If a library has a temporary directory, applications write derived data to the temporary directory and source data (master files and co-master files) to the library.

Files in a temporary directory are not managed. if you want to check in a derived file that is in a temporary directory, you must copy it to the library and then check it in.

Temporary directories have the same directory structure (library/cell/view) as libraries. While writing derived data to a temporary directory, applications create cell and view directories in the temporary directory as needed.

Each library can have only one temporary directory associated with it.

You can specify a temporary directory for a single library or a global temporary directory for all libraries.

To specify a temporary directory for a library:

➤ Add the following statement to your `cds.lib` file:

```
ASSIGN libName TMP TempDirPath
```

where *`libName`* is the library to which you want to assign a temporary directory and *`TempDirPath`* is the path to the temporary directory.

For example:

```
ASSIGN LSTTL TMP /tmp/lsttl_tmp
```

The library must already be defined. The `ASSIGN` statement does not have to be in the same `cds.lib` file as the library definition.

You can remove a `TMP` attribute with the `UNASSIGN` statement.

To specify a global temporary directory for all libraries:

➤ Add the following statement to your `cds.lib` file:

```
ASSIGN AllLibs TmpRootDir path
```

where *`path`* is the path to the root of the temporary directory.

For example:

```
ASSIGN AllLibs TmpRootDir $HOME/myTempLibs
```

In this case, if you have a library `LSTTL`, its temporary directory is `$HOME/myTempLibs/LSTTL`.

The `AllLibs` directive is overriden by the `TMP` directive for libraries that have a `TMP` assigned.

Related Topics

Cadence Library Definition File

File Naming Conventions in Library Directory

Directories and files stored in Cadence library directories follow a naming convention. This lets you consistently identify library and directory types without having to examine their contents.

- The dot character (.) is not allowed in a cell or view name.
- Library files (any file in a library that is not a cell) and cell files (any file in a cell that is not a view) should have a `.`*`fileExtension`* to distinguish them from cells or views. For example, if you want to keep a text file `myfile` in a library, name it `myfile.txt` to distinguish it from the cells contained in the library. If you have files without extensions at the cell or view level, applications such as the Cadence library manager treats them as cells or views, leading to confusing error messages.
- View names are not reserved. It is possible to have a state diagram whose view name is `schematic`. However, many applications follow conventions for view names.
- Lowercase alphanumeric names starting with a letter are recommended because files can be written to both case-sensitive and case-insensitive file systems. Name length restrictions are only those enforced by the underlying file system.

Related Topics

View Files and a Co-Managed Set

3

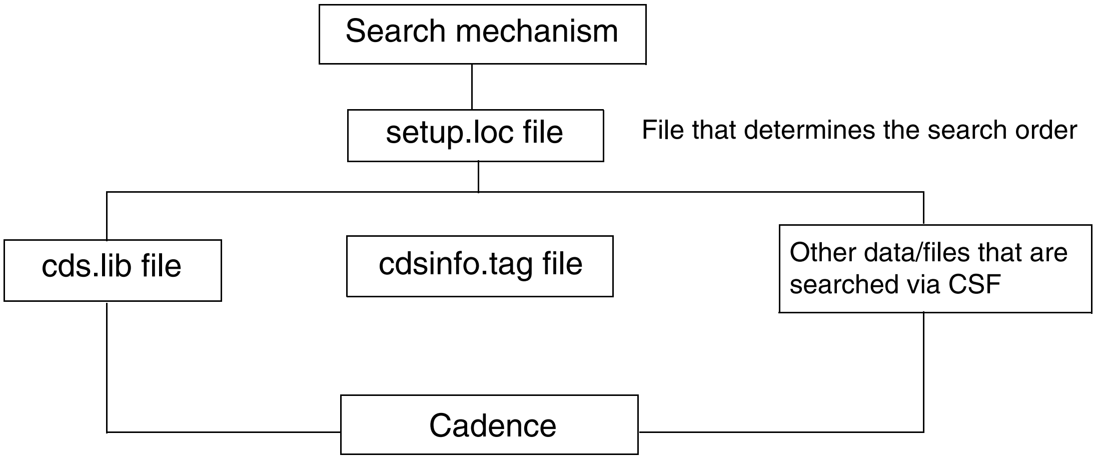
Cadence Setup Search File: setup.loc

Cadence software includes the Cadence Setup Search File (CSF) mechanism to search for application setup and configuration information such as the following:

- `cds.lib` files, which define the design libraries that you use
- `cdsinfo.tag` files, which contain information such as the `DMTYPE` for a library
- `.cdsplotinit` files, which contain plotting configuration

  The CSF mechanism is used to find the `.cdsplotinit` file if the file name is specified in the `csfLookupConfig` file.

To find the application setup and configuration information, CSF uses the `setup.loc` file. The `setup.loc` file is an ASCII file that specifies the locations to be searched and the order in which they should be searched.


Search mechanism
setup.loc file
File that determines the search order
cds.lib file
cdsinfo.tag file
Other data/files that are searched via CSF
Cadence

Search Order to Locate the setup.loc File

To find the `setup.loc` file or to search for setup and configuration information if no `setup.loc` file is found, Cadence Setup Search File (CSF) uses the following search order. It uses the first `setup.loc` file that is found.

- `.`
- `@LIBRARY`
- `$CDS_WORKAREA` (if defined)
- `$CDS_SEARCHDIR` (if defined)
- `$HOME`
- `$CDS_PROJECT` (if defined)
- `$CDS_SITE` (if not defined, defaults to *`your_install_dir`*`/share/local`)
- `$(compute:THIS_TOOL_INST_ROOT)/share`

For each of these locations, CSF looks in the following directories, in this order:

- *`location`*`/.cadence`
- *`location`*
- *`location`*`/cdssetup`

Cadence provides a default `setup.loc` file in the *`your_install_dir`*`/share/cdssetup` directory.

Related Topics

Locating Additional Files Using CSF

Cadence System Information File: cdsinfo.tag

Cadence Library Definition File

Cadence Data Registry File: data.reg

Search Mechanism in Cadence Setup Search File

To find setup and configuration information, Cadence Setup Search File (CSF) searches the locations specified in the `setup.loc` file, in the order in which they are listed, until a match is found.

For each location specified in the `setup.loc` file, except `@LIBRARY`, CSF looks in the following directories, in this order:

- In a subdirectory named `.cadence`
- In the same location specified in the `setup.loc` file
- In a subdirectory named `cdssetup`

If the file being searched for is found in *location*`/.cadence`, the other directories are not searched. If the file is not found in *location*`/.cadence`, the location specified in the `setup.loc` file is searched. If the file is not found, then *location*`/cdssetup` is searched. If the file is not found there either, CSF looks at the next entry in the `setup.loc` file. It continues to search each entry in the `setup.loc` file in the same way until a match is found.

For an `@LIBRARY` entry in a `setup.loc` file, CSF searches design libraries listed in the library definition file, in the order in which they are listed. The `@LIBRARY` entry is ignored if the file being searched for is not a library file. For more information, see `@LIBRARY`.

With this search mechanism, you do not have to store your customized Cadence application configuration information directly in `$HOME`. Instead, these files can be stored under a `cdssetup` or `.cadence` subdirectory so that they can be accessed without overcrowding your home directory. Similarly, site information can be stored in a `$CDS_SITE` directory.

Default Search Order

Cadence provides a default `setup.loc` file in *your_install_dir*`/share/cdssetup`. This file specifies the following default search order, which includes commonly-used storage locations:

Location	Description
`.`	Current directory
`@LIBRARY`	Libraries listed in the library definition file
`$CDS_WORKAREA*`	User workarea
`$CDS_SEARCHDIR*`	No longer set by applications

Location	Description
`$HOME`	Home directory
`$CDS_PROJECT*`	Project storage area
`$CDS_SITE**`	Site setup information
`$(compute:THIS_TOOL_INST_ROOT)/share`	Cadence default setup information

* Ignored if undefined

** Defaults to *`your_install_dir`*`/share/local`

If the value of `CDS_LOAD_ENV` is set to CSF, then the first directory found in `setup.loc` is considered to be the default directory to which `.cdsenv` to be saved.

Commonly Searched Locations

Commonly-used storage locations for setup and configuration files are described below. Most of these locations are defined in the default `setup.loc` file.

Location	Description
`.`	The current directory.
`$CDS_WORKAREA`	A workarea stores your project-specific customizations. *Important* You should not to set the `$CDS_WORKAREA` environment variable in `.cdsinit`. It is always recommended to set the `$CDS_WORKAREA` shell environment variable before starting Virtuoso. The value of `$CDS_WORKAREA` is a path to the workarea directory.
`$CDS_SEARCHDIR`	`$CDS_SEARCHDIR` is no longer set by Cadence applications. Do not set this variable yourself.
`$HOME`	The home directory is the primary location for user-specific customization.
`$CDS_PROJECT`	A project contains information that is specific to the project and is not likely to change over the life of the project.

Location	Description
`$CDS_SITE`	Your site configuration is the primary location for you to customize the software for your site. Customization at this level is independent of any project and can be overridden by individual projects. You can also install default methodologies in this location. Other environment variable values are overwritten at startup with the computed values of this configuration. `$CDS_SITE` is the only environment variable that you use without setting it explicitly in your `.cshrc` or `.login` files. If `$CDS_SITE` is not defined, it defaults to *`your_install_dir`*`/share/local`.
`$(compute:THIS_TOOL_INST_ROOT)/share`	This directory contains the application default setup information.
`@LIBRARY`	When the reserved keyword `@LIBRARY` is specified in a `setup.loc` file, CSF searches design libraries listed in the library definition file, in the order in which they are listed. Only the library directory is searched. Unlike for other entries in the `setup.loc` file, CSF does not look for the subdirectories `.cadence` and `cdssetup` in libraries. Libraries are searched only if the file being looked for contains library information, for example, a Virtuoso XL mapping file. Otherwise, the `@LIBRARY` entry is ignored.

Related Topics

Specifying Environment Settings

Customizing the Search Mechanism

Customizing the Search Mechanism

You can customize the search order of the setup and configuration files by creating your own setup.loc file that specifies the locations to be searched and the order in which they should be searched.

Important

Do not modify the default setup.loc file provided by Cadence.

To create a setup.loc file:

1. Create a file named setup.loc in any directory Cadence Setup Search File (CSF) searches for the setup.loc file. You can do this by copying the default setup.loc file *your_install_dir*/share/cdssetup/setup.loc to the required location or by using a text editor to create a new ASCII file.
2. Edit the file in following ways:
   - ❑ Change the order of the default search locations. See Default Search Order for a list of default locations and their suggested use.
   - ❑ Delete any locations that you do not want CSF to search.
   - ❑ Add new locations that you want CSF to search.

     Specify the paths in the order in which you want them searched. Separate paths with newlines. Paths can contain environment variables, ~, and installation root expressions.

   CSF searches the locations in the setup.loc file in the order in which they are listed. Also, for each location, CSF searches location/.cadence, location, and location/cdssetup.

Specifying Custom Search Order for .cdsenv

The CDS_LOAD_ENV shell environment variable is used to specify a custom search order for .cdsenv. This variable does not affect the search order of other setup files, such as cds.lib, cdsinfo.tag, setup.loc, and csfLookupConfig.

Some setup files are searched for the first match found, while others are searched for all matches found.

Related Topics

Syntax and File Formats in the setup.loc File

Search Order to Locate the setup.loc File

Syntax and File Formats in the setup.loc File

The `setup.loc` file consist of a set of lines with a simple syntax:

- A line is either empty, a comment, or a search location.
- There is one entry per line. Everything after the first blank space or tab is a comment.
- A comment line begins with either a pound sign (`#`) or a double hyphen (`--`). For example:

  ```
  # this is a single line comment
  -- this is a single line comment
  ```
- A search location is a path that ends with a blank space or new line. If a search location path ends with a blank space, additional comment characters can follow it on the same line.
- A search location path is a relative path or a fully qualified path. A relative path is processed relative to your current working directory.
- A search location path can include environment variables in the following form: `$envvar` or `${envvar}`. It can include ~ and ~*username*, which are expanded as needed. It can also include installation root expressions such as: `$(compute:THIS_TOOL_INST_ROOT).`

Installation Root Expressions

You can use the following expressions in any path you specify in the `setup.loc` file to refer to the installation root of Cadence hierarchies. These are not environment variables and you do not need to set them in `.cshrc` or `.login` files.

Command	Description
`$(compute:THIS_TOOL_INST_ROOT)`	Interpreted as the root of the installation hierarchy of the application that is reading the `setup.loc` file. If the application is not in a Cadence installation hierarchy, an error occurs.
`$(csf_search:somePathlette)`	Specifies that a search for a path fragment is to be performed in each location specified in `setup.loc`. The path being resolved to the first match found.

Command	Description
`$(compute:THIS_FILE_INST_ROOT)`	Interpreted as the root of the installation hierarchy of the `setup.loc` file that is being read. If the `setup.loc` file is not in an installation hierarchy, an error occurs.
`$(inst_root_with:`*`pathRelativeToRoot`*`)`	where *`pathRelativeToRoot`* is the path (relative to the root of the installation hierarchy) to a file or directory. This expression is interpreted as the root of the first installation hierarchy found in `$PATH` that contains *`pathRelativeToRoot.`* If you are using applications in the Stream Manager environment, this expression is interpreted as the root of the first installation hierarchy found in `$CDS_STRM_DIR_LIST` that contains *`pathRelativeToRoot`*. If *`pathRelativeToRoot`* is not found, an error occurs.

Related Topics

Commonly Searched Locations

Search Mechanism in Cadence Setup Search File

Locating Additional Files Using CSF

Some common configuration files, such as `cds.lib` and `cdsinfo.tag`, can be easily found while running a search using Cadence Setup Search File (CSF). You can specify that Cadence applications also locate other configuration files, such as `.cdsinit` or `.cdsenv`, through Cadence Setup Search File mechanism. This enables you to customize the search order for these files.

To specify files that are to be found through CSF,

1. Create an ASCII file named `csfLookupConfig` in any directory that is listed in your `setup.loc` file, for example `$CDS_SITE` or `$HOME`. The `csfLookupConfig` file itself is found through CSF.
2. In the `csfLookupConfig` file, specify the files that you want found through CSF.

Tip

A sample file, `csfLookupConfig.sample`, is located in *`your_install_dir`*`/share/cdssetup`. You can copy this file and modify it for your needs.

csfLookupConfig File Format

Use the following keywords in your `csfLookupConfig` file to specify the files to be found through CSF:

Keyword	Description
`ALL`	Use CSF search for all configuration files
`NONE`	Do not use CSF search for any configuration file
`INCLUDE` *`fileName`*	Use CSF search to find *`fileName`*
`fileName`	Use CSF search to find *`fileName`*
`EXCLUDE` *`fileName`*	Do not use CSF search to find *`fileName`*

Keyword	Description
`LOAD` *`fileName`*	Use to specify that a file that contains `csfLookupConfig` statements should be loaded. If a relative filepath is specified, it is resolved relative to the location of the file that contains the `LOAD/SOFTLOAD` statement. The filepath can be specified as a string or an expression using either `$(csf_search:somePathlette)` or `$(compute:THIS_TOOL_INST_ROOT).` An error is displayed if the specified file is not found.
`SOFTLOAD` *`fileName`*	As above (`LOAD`), but no error is displayed if the specified file is not found.

The following rules apply to the `csfLookupConfig` file:

- You can specify only one *`fileName`* per line.
- Comment lines begin with a pound sign (#).
- Keywords are case-insensitive.
- Commands listed later in the file override any previous commands. For instance, if an `ALL` command is followed by a `NONE` command, the `ALL` command will be ignored and CSF search will not be applied to any configuration files.

Caution

Do not use the csfLookupConfig file to modify search for files that are always found through CSF, such as cds.lib or cdsinfo.tag.

Checking Files Searched Using CSF

You can check whether a specific file is found through CSF by using the `cdswhich` command:

```
cdswhich -lookupconfig fileName ...
```

If a `csfLookupConfig` file specifies that a *`fileName`* can be found through CSF, then the path of the file is returned, for example:

```
Info (cdswhich): '.cdsinit' found at:
```

```
./.cdsinit
```

If a `csfLookupConfig` file is not found or does not specify that *`fileName`* to be found through CSF, then the following message is returned:

```
Info (cdswhich): Cadence software is not configured to locate file '.cdsinit' via
Cadence File Search mechanism
```

Related Topics

CSF Support For Third-Party Simulators

The cdswhich Command

CSF Support For Third-Party Simulators

This topic contains sample information, for both end-users and EDA vendors, regarding the integration of third-party tools using Cadence Setup Search File (CSF).

For example, your `csfLookupConfig` contained:

```
SOFTLOAD $(csf_search:mentor.conf)
SOFTLOAD $(csf_search:agilent.conf)
SOFTLOAD $(csf_search:bda.conf)
SOFTLOAD $(csf_search:silvaco.conf)
```

Then third-party vendors can maintain their own files, and it always be possible to pick up the latest versions.

The `data.reg` command can also be used to perform something similar:

```
SOFTINCLUDE $(csf_search:mentor.reg);
SOFTINCLUDE $(csf_search:agilent.reg);
SOFTINCLUDE $(csf_search:bda.reg);
SOFTINCLUDE $(csf_search:silvaco.reg);
```

The idea in both of these example scenarios is that `$(csf_search:…)` would look for the correct file using the CSF mechanism, and consequently complete control over the specified locations could be performed using the `setup.loc` file. The `setup.loc` file could also either use hard paths, or environment variables to indicate a search order. Because `SOFT` versions of these commands have been used in the examples, the `data.reg` and `csfLookupConfig` files could remain unchanged, even if specific software was not enabled within your environment, for example, if a module had not been loaded.

Integration Recommendations for EDA Vendors

These guidelines contain recommendations from Cadence to third-party EDA vendors, for the integration of their tools using `SOFTLOAD` and `SOFTINCLUDE`:

- The correct structure should be provided, for example:
    - `.cdsenv` default files

      For each tool, you require a `cdsenv` file named “`<tool>.cdsenv`” (for example, `eldoD.cdsenv` or `adit.cdsenv`). These are the same registration files that were previously included in `<instdir>/tools/dfII/etc/tools/<tool>/.cdsenv`.

    - `hierEditor/templates`

      This directory contains the hierarchy editor templates that would previously have been placed in `<instdir>/share/cdssetup/hierEditor/templates`.

- ❑ `menus`

  This directory contains any menu customisation files (previously stored in `<instdir>/tools/dfII/etc/tools/menus`).

- ❑ `hnl` and `si/caplib`

  This directory contains the netlister "`ile`" files that were previously stored in `<instdir>/tools/dfII/etc/skill/hnl` and `<instdir>/tools/dfII/etc/skill/si/caplib`.

- ❑ `cdssetup/registry/{data,tools}`

  This directory contains data and tools information that was previously held in `<instdir>/share/cdssetup/registry/{data,tools}`.

- ❑ `data.reg`

  This contains `INCLUDE` statements used to reference the registry files.

- ❑ `csfLookupConfig`

  This containts statements that inform of the use of CSF mechanisms for `hnl` and `si/caplib` files - all of which are listed. It also instructs to use CSF for `.cdsenv lookup`.

- ■ Create a unique symbolic link to the `data.reg` file called, for example, `mentor.reg`.
- ■ Create a unique symbolic link to the `csfLookupConfig` file, for example, `mentor.conf`.

Example of Vendor Site Directory

- ■ The following is an example vendor site directory as referenced from `setup.loc`:

```
-andrewb_14> ls -lG mentor_site
total 300
-rw-r--r-- 1 andrewb 59659 Nov 19 13:24 ADVance_MS.cdsenv
-rw-r--r-- 1 andrewb 5528 Nov 19 13:24 adit.cdsenv
-rw-r--r-- 1 andrewb 6024 Nov 19 13:24 adit_sp.cdsenv
-rw-r--r-- 1 andrewb 10763 Nov 19 13:24 artist_link.cdsenv
-rw-r--r-- 1 andrewb 2111 Nov 19 13:24 auCore.cdsenv
drwxr-xr-x 4 andrewb 4096 Nov 19 12:20 cdssetup/
-rw-r--r-- 1 andrewb 1757 Nov 19 13:25 csfLookupConfig
-rw-r--r-- 1 andrewb 277 Nov 19 13:31 data.reg
-rw-r--r-- 1 andrewb 49261 Nov 19 13:24 eldoD.cdsenv
-rw-r--r-- 1 andrewb 53148 Nov 19 13:24 eldoD_sp.cdsenv
drwxr-xr-x 3 andrewb 4096 Nov 19 12:10 hierEditor/
drwxr-xr-x 2 andrewb 4096 Nov 19 12:34 hnl/
lrwxrwxrwx 1 andrewb 15 Mar 18 16:19 mentor.conf -> csfLookupConfig
lrwxrwxrwx 1 andrewb 8 Nov 19 16:53 mentor.reg -> data.reg
drwxr-xr-x 2 andrewb 4096 Nov 19 13:25 menus/
```

```
drwxr-xr-x 3 andrewb 4096 Nov 19 12:34 si/
-rw-r--r-- 1 andrewb 59537 Nov 19 13:24 spectre2eldoD.cdsenv
```

- Example structure of `csfLookupConfig`:

```
INCLUDE hnl/eldoD.ile
INCLUDE hnl/ADVance_MS.ile
INCLUDE hnl/adit.ile
INCLUDE hnl/eldoD_sp.ile
INCLUDE hnl/adit_sp.ile
INCLUDE si/caplib/eldoD.ile
INCLUDE si/caplib/ADVance_MS.ile
INCLUDE si/caplib/adit.ile
INCLUDE si/caplib/eldoD_sp.ile
INCLUDE si/caplib/adit_sp.ile
INCLUDE .cdsenv
```

- Example structure of `data.reg`:

```
INCLUDE cdssetup/registry/data/advance_ms.reg;
INCLUDE cdssetup/registry/data/netlist.reg;
INCLUDE cdssetup/registry/data/verilogAe.reg;
INCLUDE cdssetup/registry/tools/advance_ms.reg;
INCLUDE cdssetup/registry/tools/netlist.reg;
INCLUDE cdssetup/registry/tools/verilogAe.reg;
```

Related Topics

Locating Additional Files Using CSF

The cdswhich Command

The `cdswhich` command is a test, debug, and administration tool for managing the CSF search mechanism. It is located in *`your_install_dir`*`/tools/bin`.

`cdswhich` command has the following syntax:

```
cdswhich [-debug] [-all] filename
cdswhich [-debug] [-where path] [-formaldir path] filename
cdswhich [-debug] -lookupconfig filename ...
cdswhich [-debug] -findsetuploc
```

Command Options

Command	Description
`-debug`	Displays information about the file paths that are examined when searching for files.
`-all`	Lists all occurrences of *`filename`* in all search locations, not only the first occurance. The search locations are the locations listed in the `setup.loc` file found by CSF. `cdssetup` and `.cadence` subdirectories are also searched.
`-where` *`path`*	Searches a specific location for *`filename`*. Only the same path specified is searched; `cdssetup` and `.cadence` subdirectories are not searched. For example: `cdswhich -where mnt3/RunTools myFile`

Command	Description
`-formaldir` *`path`*	Searches for *`filename`* in the *`/path`* subdirectories of search locations if *`filename`* is not found in the search locations. *`path`* must be a subdirectory path. The search locations are the locations specified in the `setup.loc` file found by CSF. `cdssetup` and `.cadence` subdirectories are not searched by default. For example: `cdswhich -formaldir cdssetup myFile` searches for `myFile` in the `cdssetup` subdirectory of each search location listed in the `setup.loc` file, if the file is not found in the search location directory itself.
`-lookupconfig` *`fileName`*	Determines whether the `csfLookupConfig` file includes a valid value in *`fileName`* and the CSF search can be completed.
`-findsetuploc`	Writes *`filepath`* to the `setup.loc` file that specifies the CSF search locations. CSF uses the default search order to locate `setup.loc`. A customized `setup.loc` file can result in `% cdswhich setup.loc`, which can write a different filepath.

If you do not specify any options with the `cdswhich` command, only the first file found is reported.

Examples

```
cdswhich cds.lib
cdswhich -all cds.lib
cdswhich -where /usr1/mnt3/ns/Examples testFile
cdswhich -formaldir Examples testFile
cdswhich -lookupconfig .cdsinit
```

Related Topics

Locating Additional Files Using CSF

Locating Additional Files Using CSF

Syntax and File Formats in the setup.loc File

4

Cadence System Information File: cdsinfo.tag

The Cadence system information file, `cdsinfo.tag`, defines the configuration of several key capabilities, including:

- The type of design management system used to manage a library
- A strict library checking mechanism should be used to identify Cadence libraries
- The case sensitivity of the file system that contains the library

The `cdsinfo.tag` file is an ASCII file that contains entries for various Cadence applications, libraries, design management systems, and file system properties. Properties in the `cdsinfo.tag` files are found using the `cdsinfo.tag` search mechanism.

`cdsinfo.tag` file example:

```
CDSLIBRARY
CDSLIBCHECK ON
DMTYPE TDM
NAMESPACE LIBRARYUNIX
```

The cdsinfo.tag File Location

You might have more than one `cdsinfo.tag` file in your Cadence environment. The search mechanism looks for the file in the following order and uses the settings found last in the search order.

1. The default site-wide and user-wide `cdsinfo.tag` file in *`your_install_dir/`*`share/cdssetup` where *`your_install_dir`* is the location of the Cadence software installation. Entries in this file are the default, for example, the default `DMTYPE` for newly created libraries.
2. A `cdsinfo.tag` file in a directory referenced by the `$CDS_SITE` environment variable. It applies to all directories on your site.

3. A library-specific `cdsinfo.tag` file with settings for the library such as, the `DMTYPE` used for the library, whether the directory should be treated as a library, and so on.

Related Topics

Cadence Setup Search File: setup.loc

The cdsinfo.tag Search Mechanism

The cdsinfo.tag Search Mechanism

The data contained in `cdsinfo.tag` files is accessed by Cadence applications, such as the Virtuoso Studio Design Environment. These applications always look for system information about an object, a file or directory, in the context of the path of that object. For example, an application provides a file location when it needs to know which design management system is managing the file.

The `cdsinfo.tag` search mechanism uses a combination of that path and the Cadence search mechanism to locate the `cdsinfo.tag` file that contains the information requested by the application.

It first looks for a `cdsinfo.tag` file in the directory that is provided by the application or the directory that contains the file that is provided by the application. If a `cdsinfo.tag` file is found and it contains the property that the application requested, then the value of that property is used. But if the file is not found in the directory or if the file does not contain the property that the application requested, then the parent directory is searched. The mechanism continues to look for a `cdsinfo.tag` file with the required property upwards through the path, up to the root of the file system. If it is still not found, then the Cadence search mechanism (CSF) is used to search other locations for the `cdsinfo.tag` file. The Cadence search mechanism is determined by the `setup.loc` file.

If the property is not found in any `cdsinfo.tag` file, default values are used. The default value for each property is described in Entry Types.

To determine which `cdsinfo.tag` file is being used to obtain the value of a property, use the following command:

```
cdsinfo -lookup propertyname
```

An exception to the search order described above is the `CDSLIBRARY` property. When an application requests the value of the `CDSLIBRARY` property, only the current directory provided by the application is searched for the `cdsinfo.tag` file.

Entry Types

The `cdsinfo.tag` file contains Cadence application, library, design management system, and file system properties. For example, the library identification entry is the property `CDSLIBRARY`.

The first word of any entry is the property name. It is followed by the value of the property. The following properties can be in a `cdsinfo.tag` file:

Property	**Description**
`CDSLIBRARY`	Identifies a directory as being a `CDS` (Cadence) library. When `CDSLIBCHECK` is `ON`, this entry must exist in a `cdsinfo.tag` file in the library directory for the library to be considered a valid Cadence library. If no `CDSLIBRARY` entry is found, its value is assumed to be `NO`. When an application requests `CDSLIBRARY` information for a directory, only that directory is searched for a `cdsinfo.tag` file.
`CDSLIBCHECK`	Activates a mechanism that provides control over the definition of libraries. By default, libraries are defined as being libraries by their presence in a `cds.lib` file. By adding a `CDSLIBCHECK ON` entry to a `cdsinfo.tag` file, a site CAD administrator might want to stipulate that library certification be done by enforcing that a `cdsinfo.tag` file exists in each library directory with a `CDSLIBRARY` entry in it. If `CDSLIBCHECK` is off by default, `cdsinfo.tag` files need not exist and are not checked for `CDSLIBRARY` entries. The expected use model for `CDSLIBCHECK` is that it be set globally in the site `cdsinfo.tag` file. If no `CDSLIBCHECK` entry is found, its value is assumed to be `OFF`.

`DMTYPE`	Identifies the native design management system in use, if any. Generic Design Management (GDM) requests that cannot identify a design management system will fail. A value of `none` means that the directory is not managed. You can set up the site `cdsinfo.tag` file to set a default for your site.
	Examples:
	`DMTYPE tdm`
	`DMTYPE crcs`
	`DMTYPE none`
	If no `DMTYPE` entry is found, its value is assumed to be `none`.

Property	Description
NAMESPACE	Identifies the native file system so that proper name space handling can be done by application programs. NAMESPACE entries have a single additional meaningful word, which is either LibraryNT or LibraryUnix. If no name space entry is found, the default is the architecture of the machine on which the application is running.

The CDSLIBRARY and CDSLIBCHECK entry types together provide an alternative to the current library identification mechanism, which consists entirely of the presence of a DEFINE entry in a cds.lib file. The system reports as errors those libraries that are defined but that do not have cdsinfo.tag files with the CDSLIBRARY entry. The default (no CDSLIBCHECK entry type) acts as if a CDSLIBCHECK OFF entry was found, prevents damage of a site configuration upon installation of new software including this mechanism, and maintains the current behavior.

Related Topics

Cadence Setup Search File: setup.loc

Syntax and File Format of cdsinfo.tag

File syntax for entries in the `cdsinfo.tag` files consist of a set of lines with a simple syntax:

- When running on Windows, the backslash (\ ) and the forward slash (/) are treated synonymously.
- When running on Windows, *`<letter>:`* is a legal path, but not when running on UNIX.
- A comment line is any line in which the first nonwhitespace character is a double hyphen (--) or a pound sign (#). Examples are as follows:

  ```
  -- this is a single line comment
  # this is a single line comment
  ```
- Whitespace is any blank or tab character. Blank lines are acceptable.
- An entry consists of a keyword followed by zero or more words, terminated by a new line character or a pound sign (#).
- Keywords are case insensitive.

Sample Site and Library Files

The pound sign (#) denotes a comment. Remove the leading pound signs from entries to activate them.

A Sample Site cdsinfo.tag File

```
# This is a sample cdsinfo.tag file as it might be set for a
# site. Place this file in the directory referenced by the shell
# environment variable CDS_SITE, and have all Cadence users share
# the same setting for CDS_SITE.
#
# Select a site-wide DM system.
# Use ‘none’ to turn off use of DM for the site.
dmtype tdm
#
# Select (enable) the strict library checking mechanism
#
# ALL Cadence libraries must has a local cdsinfo.tag
# file with a CDSLIBRARY entry if this is used.
cdslibcheck ON
```

A Sample Library cdsinfo.tag File

```
# This is a sample cdsinfo.tag file as it might be set for a
# Cadence Library. This file needs to be located in the library
# directory (not under a cell or view).
#
```

```
# Override the site DM selection - mark this library as unmanaged;
# it is used for experimental work only.
dmtype none
#
# Indicate that this directory is a Cadence library.
cdslibrary
```

Related Topics

The cdsinfo.tag Search Mechanism

The cdsinfo command

Glossary

The cdsinfo command

The `cdsinfo` command is used as a test, debug, and administration tool for the `cdsinfo.tag` file mechanism. It is located in *`your_install_dir/tools/bin`*.

The command syntax is as follows:

```
cdsinfo [-verbose | -v] [-path filepath] [-configurelibrary | -checklibrary |-
     lookup entryname... | -show | -addentry entryname value | -clearentry
     entryname]
```

Argument	Description
`-verbose | -v`	Provides detailed information about the command progress and where it finds information. When used with the `-configurelibrary` argument, the `-verbose` option lists the path to your `cds.lib` file and a list of library directories selected for configuring.
`-path` *`filepath`*	Specifies the path to the file to use for the other arguments.
`-configurelibrary`	Examines your `cds.lib` file and validates all the entries, including those brought in by `INCLUDE` and `SOFTINCLUDE` statements. It examines every library directory defined in the `cds.lib` file and attempts to create a `cdsinfo.tag` with a `CDSLIBRARY` entry. The system issues a warning message if it cannot write to a library directory. Library directories that already contain a `cdsinfo.tag` file with a `CDSLIBRARY` entry are not changed. Those with a `cdsinfo.tag` file without a `CDSLIBRARY` entry have one added to the `cdsinfo.tag` file, if possible.

Argument	Description
`-checklibrary`	Performs multiple operations such as: ■ Finds a `cds.lib` file. ■ Checks libraries defined by the `cds.lib` file and the other `cds.lib` files it includes to find `cdsinfo.tag` files with `CDSLIBRARY` entries in them. Prints the names of the libraries and the paths to the `cdsinfo.tag` files. ■ Checks all subdirectories of the library directories identified by the `cds.lib` file to find the `cdsinfo.tag` files. Issues a warning for any sublibrary `cdsinfo.tag` files found that indicate an incorrect configuration. For a large library, this process could take some time. ■ Checks all `cdsinfo.tag` files that are found in step 2 for correct syntax and reports any syntax errors.
`-lookup` *`entryname`* `...`	Looks for a `cdsinfo.tag` file that contains an entry with the given name and prints the appropriate search rules for the given entry type. This command also gives the location of the `cdsinfo.tag` file that is used.
`-show`	Displays the path and contents of the `cdsinfo.tag` file in the current directory or in the directory specified with `-path`. If there is no `cdsinfo.tag` file in that directory, searches for it in the parent directory, continuing up to the root of the path.
`-addentry` *`entryName value`*	Adds the entry to the `cdsinfo.tag` file specified by`-path`. If the path is not specified, adds the entry to the `cdsinfo.tag` file that is in the current working directory. If the current working directory does not contain a `cdsinfo.tag` file, this command creates one.
`-clearentry` *`entryName`*	Removes the entry from the `cdsinfo.tag` file specified by `-path` or, if the path is not specified, from the `cdsinfo.tag` file that is in the current working directory.

You can also use the `ddSetLibUnmanaged` and `ddClearLibUnmanaged` SKILL functions to set or remove the `DMTYPE` entry from the `cdsinfo.tag` files.

Related Topics

ddSetLibUnmanaged

ddClearLibUnmanaged

Design Management Functions

5

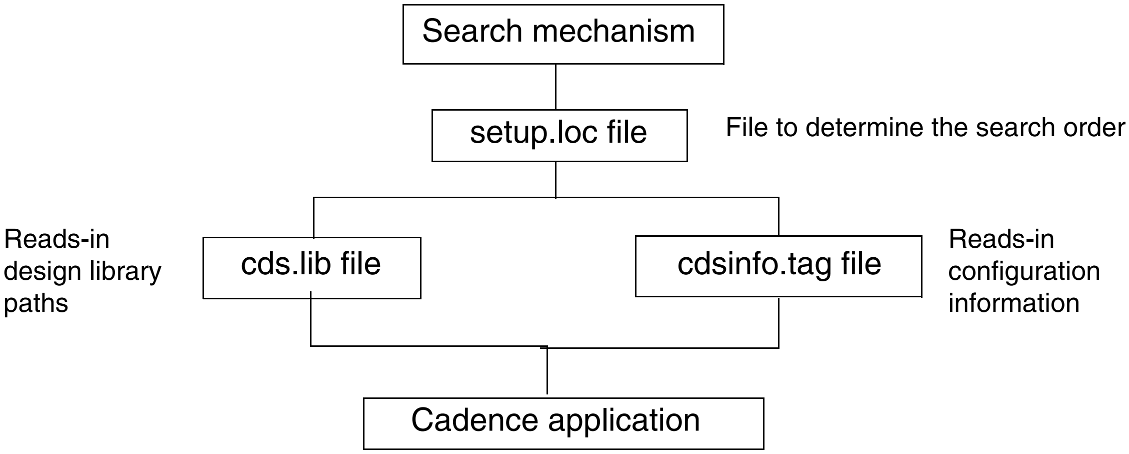
Cadence Library Definition File

A library definition file is used in Cadence software to define libraries. A library definition file maps library names to physical directory paths. Applications read this file to identify the libraries that they can use. Virtuoso Studio uses the `cds.lib` file as its library definition file.

The `cds.lib` file is an ASCII file used to define libraries. The file maps user library names to physical directory paths.

Applications read the `cds.lib` file to identify the libraries they can use. Usually, one `cds.lib` file, which might reference other files, determines which libraries are available to your application. Other `cds.lib` files can be included in the `cds.lib` file with the `INCLUDE` and `SOFTINCLUDE` statements. This allows you to customize the `cds.lib` files for specific projects or at different levels such as the site, group, or user level.

The Cadence search mechanism (CSF) is used to find the correct `cds.lib` file for your software. The first `cds.lib` file that is found is used.


Search mechanism
setup.loc file
File to determine the search order
Reads-in design library paths
cds.lib file
cdsinfo.tag file
Reads-in configuration information
Cadence application

Example

```
# The DEFINE statement defines library references.
DEFINE ttl /users/$USER/ttl
# The SOFTDEFINE statement is similar to DEFINE but doesn’t print errors.
SOFTDEFINE myDesign /users/$USER/parts
# The INCLUDE statement reads a file.
INCLUDE /users/$USER/cds.lib
```

```
# The SOFTINCLUDE statement is similar to INCLUDE but doesn't print errors.
SOFTINCLUDE $GOLDEN/cds.lib
# The UNDEFINE statement undefines the iclib library.
UNDEFINE iclib
DEFINE iclib ./ic_lib
```

The cds.lib File Location

Cadence provides a default `cds.lib` file in the *`your_install_dir`*`/share/cdssetup` directory. In addition, applications might create a `cds.lib` file in other directories such as your current working directory when you create a new library. You can also create a `cds.lib` file in any directory listed in the `setup.loc` file.

Multiple cds.lib Files

You can have multiple `cds.lib` files.Use the `INCLUDE` statement to include them in the primary `cds.lib` file. The primary file must be named `cds.lib` because that is the name the system searches for by default. Included files do not need to be named `cds.lib`.

You can have a user `cds.lib` file that contains library settings used to support all your projects. You can also have project-wide or local `cds.lib` files located in specific design directories that contain library settings specific to each project, such as technology or cell libraries. These can be combined in many ways with the `INCLUDE` statement.

Related Topics

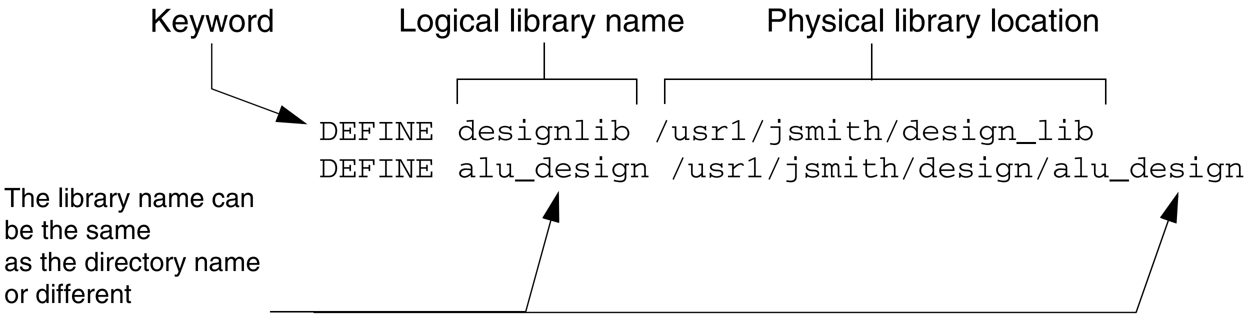
Cadence Setup Search File: setup.loc

Creating or Editing a cds.lib File

Included files

cds.lib File Statements

The categorisation of the `cds.lib` file statement is divided into a keyword, a valid library name, and the library location. For example,


Keyword
Logical library name
Physical library location
The library name can be the same as the directory name or different

```
DEFINE designlib /usr1/jsmith/design_lib
DEFINE alu_design /usr1/jsmith/design/alu_design
```

Use the following statements in a `cds.lib` file:

Statement	Description

`DEFINE` *`lib pathToLib`*	Defines *`lib`* as the logical reference to the directory specified as *`pathToLib`*. The same directory cannot be contained in multiple library definitions. An error message is printed if *`pathToLib`* does not exist.
	For example:
	`DEFINE ttl_lib /usr1/libraries/ttl_lib`
	`DEFINE ttl ./libraries/ttl`
	`DEFINE designabc ~user/designABC`
	`DEFINE companyabc $HOME/companyABC`

`SOFTDEFINE` *`lib pathToLib`*	Same as the `DEFINE` statement, except that no error message is printed if *`pathToLib`* does not exist. For example: `SOFTDEFINE myLib /usr1/libraries/parts_lib`

Statement	Description
`UNDEFINE` *`lib`*	Undefines the specified library. This command is useful for removing any libraries that were defined in other files. It is not an error if *`lib`* was not previously defined. For example, `UNDEFINE ttl`
`INCLUDE` *`file`*	Reads the specified file as a `cds.lib` file. Using `INCLUDE` is the same as incorporating the contents of *`file`* within the `cds.lib` file, except that file paths in the included file are relative to the directory containing the included file. An error message is printed if *`file`* is not found or if recursion is detected. *`file`* does not have to be named `cds.lib`. The following example reads the `cds.lib` file from `/users/$USER`: `INCLUDE /users/$USER/cds.lib`
`SOFTINCLUDE` *`file`*	Same as the `INCLUDE` statement, except that no error message is printed if the file does not exist. The following example reads the `cds.lib` from the `$GOLDEN` directory if it exists: `SOFTINCLUDE $GOLDEN/cds.lib`
`ASSIGN` *`libName attribute value`*	Assigns the specified attribute to a library. The library must already be defined; an error message is printed if the library has not been defined when the `ASSIGN` statement is read. An `ASSIGN` statement for a library does not have to be in the same file as the library definition.

Statement	Description

	Currently, you can use the attributes listed below with an `ASSIGN` statement; the first two define the temporary directory for libraries while `DISPLAY` and `COMBINE` let you customize the display of libraries in the Library Manager.
	`ASSIGN` *`libName`* **`TMP`** *`TempDirPath`*
	It defines the temporary directory for a library. *`libName`* is the library to which you want to assign a temporary directory and *`TempDirPath`* is the path to the temporary directory.
	For example:
	`ASSIGN LSTTL TMP /tmp/lsttl_tmp`
	`ASSIGN AllLibs TmpRootDir` *`TmpRootDirPath`*
	It defines the global temporary directory for all libraries. *`TmpRootDirPath`* is the path to the root of the temporary directory.
	For example:
	`ASSIGN AllLibs TmpRootDir $HOME/myTempLibs`
	In this case, if you have a library `LSTTL`, its temporary directory will be `$HOME/myTempLibs/LSTTL`.
	The `AllLibs` directive is overridden by the `TMP` directive for libraries that have a `TMP` assigned.
	A library can have only one temporary directory. Temporary directories contain only derived data; they cannot contain source data. Applications read both the library and its temporary directory to get library data; if there are any files in common, the files in the temporary directory have precedence. Applications write source data to the library and derived data to the temporary directory. For more information about temporary directories, see Cadence Library Structure

Statement	Description
	ASSIGN *libName* **DISPLAY** *displayAttributeName* It sets the specified attribute on the library. The `DISPLAY` attribute, which is only read by the Library Manager, is used to customize the display of libraries. For example, you can set: `ASSIGN LSTTL DISPLAY RefLibs` in your `cds.lib` file, and in the Library Manager define that all libraries tagged with the `RefLibs` attribute be displayed in blue. **ASSIGN** *combinedLibName* **COMBINE** *libA libB* **...** It defines a combined library, which is a set of libraries that are displayed together as a composite library in the Library Manager. *combinedLibName* is the name of the combined library and *libA* and *libB* are the libraries that comprise the combined library. The `COMBINE` attribute is only read by the Library Manager. Use the `UNASSIGN` statement to remove attributes from a library. You can also assign new values to attributes without having to unassign the attributes first.
`UNASSIGN` *libName attribute*	Removes the specified attribute from the library. The library must already be defined. No error message is printed if the attribute does not exist on the library. For example: `UNASSIGN LSTTL TMP` `UNASSIGN` is not supported for the `AllLibs` directive.

Related Topics

cds.lib File Statements

Creating Combined Libraries

Installation Root Expressions

You can use the following expressions in `cds.lib` file statements to refer to the root of the Cadence installation hierarchies.

Note: These are not environment variables. Do not need to set them in your `.cshrc` or `.login` files.

`$(compute:THIS_TOOL_INST_ROOT)`

Interpreted as the root of the installation hierarchy of the application that is reading the `cds.lib` file. If the application is not in a Cadence installation hierarchy, you will get an error.

For example:

```
DEFINE mixSigLib $(compute:THIS_TOOL_INST_ROOT)/
libraries/mixSigLib
```

However, when using the same `cds.lib` in the NC-Verilog environment, “`THIS_TOOL`” proves to be a wrong specification. You can use the following variant if you want the hierarchy with Virtuoso in it:

```
$(inst_root_with:tools/dfII/bin/virtuoso)/tools/dfII/etc/
cdsDefTechLib
```

Moreover, using this variant you are sure to point to the right installation root, which is independent of the tool flow/environment that you are using.

`$(compute:THIS_FILE_INST_ROOT)`

Interpreted as the root of the installation hierarchy of the `cds.lib` file that is being read. If the `cds.lib` file is not in an installation hierarchy, you will get an error.

For example:

```
DEFINE myLib $(compute:THIS_FILE_INST_ROOT)/libraries/
partsLib
```

`$(inst_root_with:`*`pathRelativeToRoot`*) where *`pathRelativeToRoot`* is the path (relative to the root of the installation hierarchy) to a file or directory.

This expression is interpreted as the root of the first installation hierarchy found in `$PATH` that contains *`pathRelativeToRoot`*. If you are using applications in the Stream Manager environment, this expression is interpreted as the root of the first installation hierarchy found in `$CDS_STRM_DIR_LIST` that contains *`pathRelativeToRoot`*. If *`pathRelativeToRoot`* is not found, you will get an error.

Related Topics

cds.lib File Statements

Syntax and File Format of cds.lib

The following rules apply to the `cds.lib` file:

- Only one statement is allowed per line.
- Blank lines are allowed.
- Use the pound sign (`#`) or the double hyphen (`--`) to begin a comment. For example:

  ```
  # this is a single line comment
  -- this is a single line comment
  ```

  You must precede and end the comment character with a blank space, a tab, or a new line.
- Keywords are identified as the first non-whitespace string on a line.
- Keywords are case insensitive.
- Directory and file names cannot contain shell wildcards, such as an asterisk (*) or a question mark (?).
- You can include environment variables such as `$HOME` or shell-style home directory references such as ~ and `~user`. Symbolic variables and library paths are in the file system name space.
- You can enter either absolute or relative file paths.

  Relative paths are relative to the location of the current `cds.lib` file being read. Use of absolute paths with design management systems might cause problems in large team environments.
- Logical library names are always in the LibraryUnix name space and are case sensitive both on UNIX and Windows.
- Any reference to `.` means the directory that was represented by the path used to find the `cds.lib` file. If you are in a work area that has a symbolic link to a `cds.lib` file, refers to the directory that the original `cds.lib` file is in. For example:

work area			**repository**
`file`	`cds.lib`	`->`	`cds.lib.6`
`contents`	`INCLUDE ./foo.lib`	`->`	`foo.lib.4`

Related Topics

cds.lib File Statements

Library Definitions in Virtuoso Applications Using the cds.lib Files

Creating or Editing a cds.lib File

You can create and update the `cds.lib` file using one of the following editors:

- Cadence Library Path Editor

  Many Cadence applications include the Library Path Editor.

- Any text editor

  The New Library form from the Command Interpreter Window or the Cadence Library Manager (Virtuoso users only) includes a text editor. When you create a new library, a new `cds.lib` file is created or the existing file is updated. Creating a new or temporary library inside of an existing library is not allowed as any directories found inside the library are considered as cells.

Using the Cadence Library Path Editor to Create or Edit the cds.lib File

To start the Cadence Library Path Editor,

➡ In a shell window, type `cdsLibEditor`. To use a namespace other than the default CDBA namespace, use the `-namespace` option with the command. For example:

```
cdsLibEditor -namespace VHDL
```

For a list of Cadence name spaces, see Name Mapping

Using a Text Editor to Create or Edit the cds.lib File

To create a new `cds.lib` file or to edit your `cds.lib` file to add libraries or include other `cds.lib` files,

- To create a new `cds.lib` file, create an ASCII file named `cds.lib` in any directory that is listed in your `setup.loc` file, for example, `$HOME`.

  The search order specified in the `setup.loc` file determines which `cds.lib` file will be used.

- To edit a `cds.lib` file, open the file in a text editor.

- To add a library, add the following statement:

  ```
  DEFINE logicalNameForLib pathToLib
  ```

  For example:

  ```
  DEFINE myLib ../libs/designLib
  ```

- To include another `cds.lib` file in your file, add the following statement:

  ```
  INCLUDE path_to_file
  ```

  For example:

  ```
  INCLUDE /net/cds/user/libs/samples/cds.lib
  ```

After making the changes, save the `cds.lib` file.

Important

You must specify library names in the `cds.lib` file in the LibraryUnix name space.

Related Topics

cds.lib File Statements

Mapping Library Names

Mapping Library Names

Library names in a `cds.lib` file are in the LibraryUnix name space. Applications map these names to their own name space. This enables all applications to share library definitions because they always interpret the library names in a `cds.lib` file in the LibraryUnix name space.

If you add a library definition to the `cds.lib` file with a text editor (instead of through an application), you must specify the logical library name in the LibraryUnix name space. You can use the `nmp` command to determine the name to include in the `cds.lib` file.

To use the `nmp` command,

➤ Type the following in a UNIX shell:

```
nmp mapName appNamespace LibraryUnix libName
```

When you specify *`libName`*, you need to escape names that contain special characters. For example, to use the library named `!Lib!` in Verilog, you need to escape the exclamation mark (!) with a backslash, because the exclamation mark requires an escaped names So, you would use the escaped name `\!Lib!␣`.

You also need to enclose names with special characters in single quotes so that the shell does not delete the special characters. Also, in some shells, you might need to escape the backslash that is part of the escaped name with another backslash.

For example, to determine the mapped LibraryUnix name for Verilog `\!Lib!␣`,

➤ In a `sh` shell, type:

```
nmp mapName Verilog LibraryUnix ’\!Lib!␣’
```

– or –

In a `csh` shell, type:

```
nmp mapName Verilog LibraryUnix ’\\!Lib!␣’
```

The `nmp` program returns:

```
#21Lib#21
```

Use the mapped name (`#21Lib#21`) in the `cds.lib` file.

You can avoid name mapping issues by always choosing names that use only lowercase letters and digits.

Related Topics

LibraryUnix name space

Name Mapping

The cdsLibDebug Command

The `cdsLibDebug` command is a testing and debugging tool for `cds.lib` files. It is located in *`your_install_dir`*`/tools/bin`.

The `cdsLibDebug` command is only available with Virtuoso applications.

You can use `cdsLibDebug` to do the following:

- Get a list of `cds.lib` files that are found by the search mechanism.

  `cdsLibDebug` lists the `cds.lib` file found, as well as any other `cds.lib` files that are included (with the `INCLUDE` statement) in that `cds.lib` file.

  These are the `cds.lib` files that your Cadence application will read.

  If `cdsLibDebug` cannot find any `cds.lib` file, it displays the following error:

  ```
  *WARNING* ddUpdateLibList: Did not find any ’cds.lib’ file.
  ```

- Get a list of library definitions that have been set in the `cds.lib` files.

  `cdsLibDebug` lists the logical name and physical path of each library and identifies the `cds.lib` file in which it was found.

  These are the libraries that your Cadence application will access.

- Find syntax errors in `cds.lib` files.

  `cdsLibDebug` reports errors such as an undefined variable or an invalid library path. It also reports a warning if the same library name is mapped to different paths.

Syntax

```
cdsLibDebug [-cdslib cdslib] [-cla] [-help]
```

Argument	Description
`-cdslib` *`cdslib`*	Overrides the default search order for `cds.lib` files and reads the `cds.lib` that you specify.

Argument	Description
-cla	Uses Cadence Library Access (CLA) semantics to read the cds.lib files. CLA is a procedural interface used to parse cds.lib files and access the library structure. It is typically used by non-Virtuoso applications. If you do not specify the -cla option, cdsLibDebug uses Design Data Procedural Interface (DDPI) semantics by default. DDPI is another procedural interface used to access the library structure and is typically used by Virtuoso applications. For more information about DDPI, see Design Management. If you want to see how a non-Virtuoso application will parse your cds.lib files, use the -cla option. Otherwise, use the default DDPI semantics mode. In the default DDPI semantics mode, you will see cdsDefTechLib in the list of libraries and an internal_implicit file in the list of cds.lib files. This is because DDPI adds cdsDefTechLib to your library list by default.
-help	Displays information about the syntax of the cdsLibDebug command. For example: cdsLibDebug cdsLibDebug -cdslib ~/design/cds.lib cdsLibDebug -cdslib ~/design/cds.lib -cla

Examples

In the following example, cdsLibDebug finds the cds.lib file in the current working directory (/mnt3/WorkDir), which includes another cds.lib file (/mnt3/WorkDir/HierEditor/cds.lib), which in turn includes another file (/mnt3/WorkDir/Test/liblist). cdsLibDebug lists all the libraries that are defined in these cds.lib files. It also prints warnings for an undefined environment variable, $VAR, as well as for an invalid library path for the library MonitorTestLib.

- Output of cdsLibDebug with the -cla option

```
% cdsLibDebug -cla
Parsing cdslib file ./cds.lib.
```

```
Warning: Invalid environment variable $VAR on line 5 of /mnt3/WorkDir/cds.lib
Warning: Invalid path /mnt3/WorkDir/MonitorTestLib on line 7 of /mnt3/WorkDir/
cds.lib

cds.lib files:
1:  /mnt3/WorkDir/cds.lib
2:  /mnt3/WorkDir/HierEditor/cds.lib
    included on line 2 of /mnt3/WorkDir/cds.lib
3:  /mnt3/WorkDir/Test/liblist
    included on line 1 of /mnt3/WorkDir/HierEditor/cds.lib

Libraries defined:

Defined in /mnt3/WorkDir/Test/liblist:
Line #  Filesys    Path
------  ----       ----
1    NEW        /mnt3/WorkDir/Test/NEW

Defined in /mnt3/WorkDir/HierEditor/cds.lib:
Line #  Filesys    Path
------  ----       ----
2    mixSigLib  /mnt3/WorkDir/HierEditor/mixSigLib
3    myTempLib  /mnt3/WorkDir/HierEditor/myTempLib
4    myDestLib  /mnt3/WorkDir/HierEditor/myDestLib
5    SourceLib  /mnt3/WorkDir/HierEditor/SourceLib

Defined in /mnt3/WorkDir/cds.lib:
Line #  Filesys    Path
------  ----       ----
3    mylib      /mnt3/WorkDir/mylib
4    myTestLib  /mnt3/WorkDir/myTestLib
6    myLibCopy  /mnt3/WorkDir/myLibCopy
```

- Output of `cdsLibDebug` without the `-cla` option:

```
% cdsLibDebug

*WARNING* The directory: '/mnt3/WorkDir/$VAR/Lib2' does not exist but was
defined in libFile '/mnt3/WorkDir/cds.lib' for Lib 'LIB2'.

*WARNING* The directory: '/mnt3/WorkDir/MonitorTestLib' does not exist but was
defined in libFile '/mnt3/WorkDir/cds.lib' for Lib 'MonitorTestLib'.

Lib Files:
1: /mnt3/WorkDir/cds.lib
2: /mnt3/WorkDir/HierEditor/cds.lib
3: /mnt3/WorkDir/Test/liblist
4: internal_implicit

Libraries defined:
1: mylib from file /mnt3/WorkDir/cds.lib  (refCount 1).
    Path: /mnt3/WorkDir/mylib
2: NEW from file /mnt3/WorkDir/Test/liblist (refCount 1).
    Path: /mnt3/WorkDir/Test/NEW
3: SourceLib from file /mnt3/WorkDir/HierEditor/cds.lib  (refCount 1).
    Path: /mnt3/WorkDir/HierEditor/SourceLib
4: mixSigLib from file /mnt3/WorkDir/HierEditor/cds.lib  (refCount 1).
    Path: /mnt3/WorkDir/HierEditor/mixSigLib
5: myLibCopy from file /mnt3/WorkDir/cds.lib  (refCount 1).
    Path: /mnt3/WorkDir/myLibCopy
6: cdsDefTechLib from file internal_implicit  (refCount 1).
    Path: /net/machine111/usr1/cadence/tools/dfII/etc/cdsDefTechLib
7: myTempLib from file /mnt3/WorkDir/HierEditor/cds.lib  (refCount 1).
    Path: /mnt3/WorkDir/HierEditor/myTempLib
8: myTestLib from file /mnt3/WorkDir/cds.lib  (refCount 1).
```

```
    Path: /mnt3/WorkDir/myTestLib
9: myDestLib from file /mnt3/WorkDir/HierEditor/cds.lib  (refCount 1).
    Path: /mnt3/WorkDir/HierEditor/myDestLib
```

`refCount` is the number of `cds.lib` files that contain that library definition.

Related Topics

cds.lib File Statements

Creating or Editing a cds.lib File

Library Definitions in Virtuoso Applications Using the cds.lib Files

Virtuoso applications on OpenAccess use the `cds.lib` files for library definition.

If you create or edit a `cds.lib` file manually, you must keep it synchronized. You can use the Cadence Library Path Editor to synchronize library definition files.

If a `cds.lib` file is not found, Virtuoso applications issue a warning.

Virtuoso applications support the `cds.lib` files in the following way:

- When you create a library in a Virtuoso application or using a `dd` function that is a part of the DDPI interface used to access the library structure, a `DEFINE` statement is added to the `cds.lib` file:

  If the `cds.lib` found by the Cadence search mechanism is not in the current working directory, a new `cds.lib` file is created in the current working directory, with an `INCLUDE` statement that includes the `cds.lib` that was found and a `DEFINE` statement for the new library.

- When you delete a library with a Virtuoso application or a `dd` function, and the library is defined only once in a `cds.lib` file, it is deleted from the file and removed from disk.

  Multiple definitions of a `cds.lib` can be found because a library can be defined multiple times in a `cds.lib` file as well as in files that are included in the `cds.lib` file.

- If Virtuoso applications do not find a `cds.lib` file, they issue a warning. Specify absolute path from the working directory to avoid any issues. Specifying relative path from the working directory can result in an error.

Related Topics

Creating or Editing a cds.lib File

Library Path Editor Window

Design Management Functions

6

Cadence Data Registry File: data.reg

The Cadence data registry maps design data formats to specific applications, view names, and related files.The Cadence data registry file mechanism supports defining various data types (especially views), associating default editors with data, as well as being a general registry facility.

The data formats and mapping can apply to any design or design project.

Use the data.reg file to register the following for your Virtuoso applications:

- Data format for each available view name
- Default editor for each data format
- Sets of file types, called co-managed sets, that design management operations treat as single entities
- Global properties for libraries and cells
- View aliases and Virtuoso viewtype support
- Icon used to represent a view in various browsers

The file contains data format definitions listing the available types of views and tool definitions the editors for the data formats. The data is not design or project specific. Cadence supplies the data registry for each Cadence application; however, you might want to integrate your own applications. In this case, you need to add that application’s data information to the registry.

The data.reg File Location

By default, registry files are located in two directories:

```
your_install_dir/share/cdssetup/registry/data
your_install_dir/share/cdssetup/registry/tools
```

where *your_install_dir* is the location of the Cadence software installation.

Each directory contains several files with an application name and a `.reg` extension, such as `composer.reg`, where that application's relationships are defined. These directories build the base set of tool and data definitions. If you do not use the default *`your_install_dir`*`/share/local` location for `$CDS_SITE`, you need to set the value of the environment variable `$CDS_SITE` to the path to your new site configuration area.

Data Directory Examples

A typical set of data registry definitions is this one for the Virtuoso® schematic editor located in *`your_install_dir`*`/share/cdssetup/registry/data/composer.reg`:

```
DataFormat ComposerSchematic { // Define Composer Schematics
    Pattern             = sch.oa;
    Preferred_Editor      = schematic;
    dfII_ViewType       = schematic;
    Co_Managed          = sch.oa master.tag data.dm pc.db verilog.v
                    verilog.vams ams_direct.dat amsAPT.apt;
}
/* Map the DataFormat names to the DFII view names*/
ViewAlias {
    schematic = ComposerSchematic;
}
```

Argument	Description
`Pattern`	The name of the master file, which might include wildcards, such as the question mark (?), which matches any one character, and the asterisk (*), which matches any character any number of times.
`Preferred_Editor`	The descriptor of the application used to create or modify views of this data format, which then searches the tools registry.
`dfII_ViewType`	The view type (view name) string is used for backward compatibility for DFII applications.
`Co_Managed`	A space-separated list of files to be checked in or out when the master file is checked in or out using a design management system. The list must include the `Pattern`, the `master.tag` (which contains the name of the master file), and the names of the other files you want to have checked in and out as a set.

Argument	Description
`ViewAlias`	Shows the view type (view name) that are used to name the directory containing the co-managed group. The system uses the `ViewAlias` section to identify the data format name associated with a view name, and then uses the `DataFormat` section to find information about that view.

Tool Directory Example

A tool registry definition for the Virtuoso schematic editor located in *`your_install_dir`*`/share/cdssetup/registry/tool/composer.reg` looks like this:

```
Tool schematic{
}
```

data.reg File Creation

To add extra definitions based on requirements, you can create a custom `data.reg` file in your current, home, or `$CDS_SITE` directory. If you need to create it for a CDS site, then set the value of the environment variable `$CDS_SITE` to the path to the site configuration area.

Related Topics

Data Declarations For data.reg File

Syntax and File Format of data.reg

Path syntax for entries in `data.reg` files are as follows:

- When running on Windows, the backslash (\ ) and the forward slash (/) are treated synonymously.
- When running on Windows, *<letter>*: is a valid path; when running on UNIX, *<letter>*: is not a valid path.
- A comment line consist of a pound sign (#), which has to be the first non whitespace character. Other examples are as follows:

```
/* this is a
   multiline comment */
-- this is a single line comment
# this is a single line comment
// this is a single line comment
```

- Multiline comments only work when a whitespace character, such as space, newline, and tab, is added after a slash-star pair (`/*` or `*/`).
- Whitespace is any blank or tab character. Blank lines are acceptable.
- An entry consists of a keyword followed by zero or more words and ends with a new line character or a pound sign (#).
- Keywords are case insensitive.
- Identifiers and property names are case sensitive.

Related Topics

The data.reg File Location

Data Declarations For data.reg File

Data declarations are need to be done for data.reg files to maintain the database. For each of the relevant types in the registry, you define the set of properties that must exist. You can also provide optional properties.

A property assignment has the following format:

```
propertyAssignment
       : identifier '=' propertyValue ';'
```

And, the format of the data declaration is as follows:

```
declaration

       : Tool identifier '[' toolProperties ']'
       | DataFormat identifier '{' dataFormatProperties  '}'
       | +DataFormat identifier '{' dataFormatProperties '}'
       | Library '{'  libraryProperties  '}'
       | Cell '{'  cellProperties  '}'
       | ViewAlias identifier dataFormatType ';'
       | ViewAlias '{' viewAliasProperties '}'
       | Preferred_Editor '{' preferredEditorProperties '}'
       | Include filename;
       | SoftInclude filename;
```

Tool Identifier

A tool description is denoted by the `Tool` keyword followed by a unique identifier.

The syntax for a `Tool` definition is given below:

```
Tool
       : 'Tool' identifier '[' toolProperties ']'
```

The valid values of tool properties are:

Properties	Description
tool_License	Checks whether a tool is available or not.
toolKnows_LCV	Indicates that the tool understands the library structure.
tool_Icon	Specifies icon that should be used to display a tool.
Start_With	Specifies the command to start the tool.

Examples of Tool Identifiers

```
Tool vi {
        tool_License = NONE;    // Not licensed
        Start_With = Exec -Program "vi" -Args "%FullPath";
        toolKnows_LCV = FALSE;
};
Tool "Daves-AI-Editor" {
        Tool_License = DAVE_AI_EDIT;
        toolKnows_LCV = TRUE;
        tool_Icon = Large share/cdssetup/toolIcons/bigHall
               Small share/cdssetup/toolIcons/smallHall.xpm;
};
```

Data Formats

In addition to tool identifiers, the system needs to know about the available data formats.

A data format is the type or representation for a view. For example, a view inside a library can be a schematic, VHDL text, or any other type. The `master.tag` file in the view directory contains the name of the master tool file in that view.

The `+DataFormat` construct lets you specify additional or overriding properties for an already declared `DataFormat` definition. If the `DataFormat` has not been declared previously, then the construct is ignored.

You can use the `+=` operator append items to the list of values previously assigned to a `DataFormat` property.

`DataFormat` properties can be categorised into:

- Required: `Pattern`, `Preferred_Editor`, and `Co_Managed`
- Not required: `Other_Editors`

Library Properties

A library data definition specifies properties that are generic to all libraries. The syntax for this definition is as follows:

```
library
    : 'Library' '{' libraryProperties '}'
```

The default definition is as follows:

```
Library {
    //cdsinfo.tag is the file that indicates that this directory
    //is a Cadence library if the associated strict lib checking
    //facility is turned on.
    //
    // *.Cat and *.TopCat describe cell categories
    // techfile.cds is the binary tech file representation
    // *.tf is an ASCII tech file representation
    // display.drf is the display resource file
    // prop.xx is the library property file
    // *.att is a translated ‘attach’ file from DFII 4.3.x
    // *.cfg is a DFII 4.3.x config file converted to a DM checkpoint
    Also_Managed = cdsinfo.tag display.drf prop.xx techfile.cds
        *.Cat *.TopCat *.att *.cfg *.rul *.tf;
}
```

Cell Properties

A cell data definition specifies properties that are generic to all cells inside libraries.

The syntax for this definition is as follows:

```
cell
    : 'Cell' '{' cellProperties '}'
```

The default definition is as follows:

```
Cell {
    // prop.xx is the cell property file
    // *.att is an attached file at the cell level
    Also_Managed = prop.xx *.att;
}
```

View Aliases

You can create new view names of any type (`DataFormat`) using a view aliasing capability. When the system needs to perform an operation on a non-existent view, it queries you for the type (`DataFormat`) of the view, or it tries to guess the type. The `ViewAlias` list essentially is a list of default view names.

For example, if you want to edit a view named `schematic` for the cell `alu`, and the view `schematic` did not exist for `alu`, then the system looks at the list of `ViewAliases` for a definition of `schematic`.

The `ViewAlias` list contains view names in the CDBA namespace. Any application using this information maps it, as appropriate, from this name space. For more information, see Name Mapping

Preferred Editor

You can specify the editor to be used to edit or view a specific data format. The following construct lets you specify a preferred editor:

```
preferredEditor
        : 'Preferred_Editor' DataFormatName ToolName ';'
        | 'Preferred_Editor' '{' preferredEditorList '}' ';'
```

```
preferredEditorList
        : DataFormatName '=' ToolName ';'
        | ' ' preferredEditorList
```

Include

```
        | Include filename;
```

The property *`filename`* is interpreted relative to the location of the file it occurs in if it is not rooted in the same way as in your `cds.lib` file.

Specify *`filename`* to either `Include` or `SoftInclude` as an expression using:

- `$(csf_search:somePathlette)`

or

- `$(compute:THIS_TOOL_INST_ROOT)`

Note: `$(csf_search:somePathlette`) specifies that a search for a path fragment is to be performed in each location specified in `setup.loc`. The path being resolved to the first match found.

SoftInclude

```
        | SoftInclude filename;
```

As `Include` above, but no error is displayed if the specified file is not found.

Related Topics

Customizing Predefined Data Formats

Expanding Files Listed in Also_Managed Property

Expanding Files Listed in Also_Managed Property

The `Also_Managed` property specified for the Library properties lists a set of files.

To expand the set of files listed in Also_Managed property, do the following:

1. Add a new or edit an existing `data.reg` file to a site location that contains a Library entry with a `Also_Managed` property specifying the existing filenames along with the new filenames.
2. To determine the existing filenames use the `% dregprint -library` command.

When upgrading to a new release of the Cadence software, it may be necessary to add more filenames to the `Also_Managed` property.

Related Topics

Data Declarations For data.reg File

Adding View Type to the Virtuoso Studio Design Environment

To create a new viewType in the Virtuoso Studio design environment:

1. Create registry files in the Virtuoso installation hierarchy *`your_install_dir/share/cdssetup/registry/`* or use the default *`your_install_dir/share/local`* location for `$CDS_SITE`, you need to set the value of the environment variable `$CDS_SITE` to the path to the site configuration area or put the `data.reg` file in your working directory.

2. Load a SKILL file or files that defines the actions or triggers to be performed when opening up a given view type.

Related Topics

Syntax and File Format of data.reg

Cadence Application Infrastructure Files

Tool and Data Registry Information

The system reads tool and data registry information in the following order:

- All the `.reg` files in *`your_install_dir`*`/share/cdssetup/registry/data` and *`your_install_dir`*`/share/cdssetup/registry/tools` are read. This builds the base set of tool and data definitions. To add data formats, add files into the `data` directory. To add tools, add files into the `tools` directory.

  The `.reg` files in a `tools` or `data` directory are read in a random order. When the same definition is found in more than one `.reg` file, the value from the last file that is read is used. No error is reported. Therefore, do not include the same definition in more than one file.

  Note: If you do any design management tasks (from an application that uses GDM or directly through `gdm` commands), the data registry is reinitialized to look for registry information in all available hierarchies. That is, all the `.reg` files in *`install_dir`*`/share/cdssetup/registry/data` and *`install_dir`*`/share/cdssetup/registry/tools` directories of all Cadence hierarchies that are defined in your environment are read. If you are running applications under the Stream Manager environment, `$CDS_STRM_DIR_LIST` is used to find the hierarchies; otherwise, `$PATH` is used to find the hierarchies. For each definition, the hierarchy of the application that is reading the registry information has the highest priority, followed by the first hierarchy defined by `$CDS_STRM_DIR_LIST` (or `$PATH`), followed by the second hierarchy defined by `$CDS_STRM_DIR_LIST` (or `$PATH`), and so on. If there are duplicate definitions, then the definition from the hierarchy that has the higher priority is used. (You will not get an error for duplicate definitions.)

- The default search mechanism, through the `setup.loc` file, searches for a `data.reg` file. If one is found, it is read on top of the definitions already read in. In other words, after reading the base set of definitions, a `data.reg` file is searched for in the same way as a `cds.lib` file.

  Other `data.reg` files can be readby including them in the first `data.reg` file that was found. To add extra definitions, create a `data.reg` file named `./data.reg` or `$HOME/data.reg`. If you do not use the default *`your_install_dir`*`/share/local` location for `$CDS_SITE`, you need to set the value of the environment variable `$CDS_SITE` to the path to the site configuration area; for example,

  ```
  INCLUDE $CDS_SITE/data.reg;
  ```

Related Topics

Data Declarations For data.reg File

Using the Data Registry for Third-Party Views

If your Cadence libraries contain views that were created in third-party applications and then encapsulated or integrated in Cadence tools, you need to use the data registry to make them available to other applications.

1. Create a site-wide `data.reg` file at `$CDS_SITE` using a text editor, for example,

   ```
   vi your_install_dir/share/local
   ```

2. Create a new data format.

Converting Custom Views

If your existing Cadence libraries contain views that were created in third-party applications and then encapsulated or integrated, you can use the registry to make them available for the future versions.

1. Create a site-wide `data.reg` file using any text editor. For example

   ```
   vi data.reg
   ```

2. Create a new data format.

Related Topics

Syntax and File Format of data.reg

Data Formats

Customizing Predefined Data Formats

You can add new properties or modify existing properties of the current data format definition using the `+DataFormat` construct. This construct adds the new definition to the existing definitions. If the `+DataFormat` construct contains a property that already exists, it gets overridden. You need to register the tool using Cadence SKILL language commands.

This is the format of an original definition:

```
DataFormat ComposerSchematic {
    Pattern = sch.oa;
    Preferred_Editor = schematic;
    dfII_ViewType = schematic;
    Co_Managed = sch.oa master.tag data.dm pc.db verilog.v verilog.vams
               ams_direct.dat amsAPT.apt;
}
```

This is the format of an addition definition in the `data.reg` file:

```
+DataFormat ComposerSchematic {
    # Add "*.vhd" to the Co_Managed set

    #

    Co_Managed += *.vhd;#
    # Add another property
    My_Property = "The user is ME";
}
```

Related Topics

Data Formats

The dregprint Command

The `dregprint` unix command is typically used to examine the registry entries, which are visible to the Cadence tools. The `dregprint` command should run from the same directory where the Cadence tool is run because files in the current working directory can affect the registry entries.

Use the following syntax for the `dregprint` command:

```
dregprint
     [-origin | -o]
     [-help | -h]
     [-i index_file]
     [-tool | -t [identifier]]
     [-dataFormat | -f [identifier]]
     [-library | -l]
     [-cell | -c]
     [-alias | -a [identifier]]
     [-pattern | -p filename]
```

Arguments

`-origin` Displays the origin (filename and line number) of each data registry entry.

`-help` Displays information about the `dregprint` command.

`-i` *`index_file`* The index file to use. An index file is a file with a `.idx` suffix that contains definitions and specifies other `data.reg` files to include.

This option is used only for testing.

`-tool|-t [`*`identifier`*`]` Displays all `Tool` entries in the data registry. Specify *`identifier`* if you want to see the properties for that identifier.

`-dataFormat|-f [`*`identifier`*`]`

Displays all `DataFormat` entries in the data registry. Specify *`identifier`* if you want to see the properties for that identifier.

`-library|-l` Displays all `Library` entries in the data registry.

`-cell|-c` Displays all `Cell` entries in the data registry.

`-alias|-a [`*`identifier`*`]`

Displays all `DataFormat` aliases. Specify *`identifier`* if you want to see aliases for that identifier.

`-pattern|-p `*`filename`*

Displays `DataFormat` entries matching the pattern you specify.

Related Topics

Data Declarations For data.reg File

Syntax and File Format of data.reg

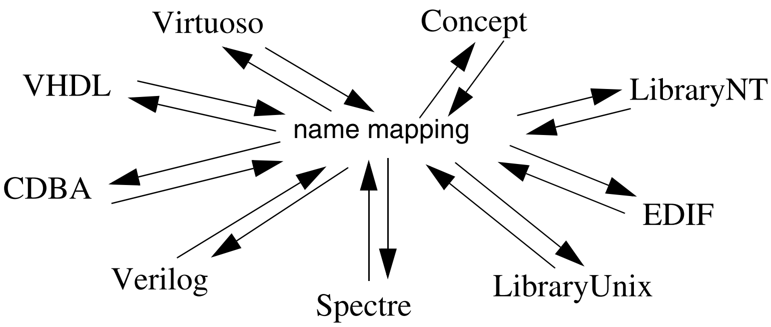
Tool and Data Registry Information

7

Name Mapping

To make data interoperable among Cadence® applications, Cadence developed a common naming convention called name mapping.

The rules for creating legal names within an application define what is called a namespace. Data between Cadence applications is interoperable, as shown below, because Cadence maps names from one namespace to another. Name mapping is consistent and predictable.



The Cadence name mapping system maps names when applications use data from other applications with noncompatible naming conventions.

Sets of Cadence applications can create names in different ways. Therefore, names used for the same design objects (such as nets, ports, instances, libraries, cells, views, and properties) vary across applications.

Importance of Name Mapping

If the Cadence system maps names between products automatically for you, why do you need to know about it?

You need to recognize that some design data appears differently when you use the data in other applications. In other applications, you see the mapped name. The system changes the names to fit each application as a design description moves through a design flow.

When you have to enter names in multiple places of a design flow where the names must match each other, you need to know how to form the corresponding name in multiple namespaces. For example,

- If you are working with a library name that is used in Leapfrog, you use the VHDL namespace.
- If you subsequently use the ncvlog, you use the Verilog namespace
- The library definition in the `cds.lib` file is in the LibraryUnix namespace.

In other words, a library used in Leapfrog with `-work MYLIB` is subsequently used in the ncvlog with `-lib mylib`. This library must be defined in the `cds.lib` file with a statement such as:

```
DEFINE mylib /usr/libs/mylibdir
```

An example of VHDL mapping to Verilog, then to LibraryUnix, and back to VHDL looks as follows:

VHDL →	**Verilog** →	**LibraryUnix** →	**VHDL**
`myName1`	`myname1`	`myname1`	`myname1`

Note: Case-insensitive names mapped into a case-sensitive namespace and back out again do not preserve case.

You also need to be aware that names in applications and names in the file system are in different namespaces. Names in the file system are in the Library namespace. Therefore, a name that you see in the file system might not always be the same as the name that you typed into the application. This usually happens when the original name contains special characters, such as a period, which are changed when the name is mapped to the Library namespace. For example, if you name a view `layout.placed`, the directory name in your file system is `layout#2eplaced`.

When you need to use the name in your application (in user interface forms, command-line options, the Command Interpreter Window, or code), do not use the mapped file system name; specify the original name instead.

You can use the `nmp` command to get the original name for a mapped file system name.

Type the following in a UNIX shell:

```
nmp mapName Library yourAppNameSpace mappedFileSystemName
```

For example, if you type the following command:

```
nmp mapName Library CDBA layout#2eplaced
```

the `nmp` command returns the following:

```
layout.placed
```

You can avoid name mapping issues by always choosing names that use only lowercase letters and digits.

How Name Mapping Works

To properly handle the variety of names in Cadence applications, each software product applies the correct namespace for each identifier (explained below). A port or pin of an instance might be called `i1/addr<3:0>` on a Virtuoso® schematic editor or Concept schematic. In Verilog, this name becomes `i1.addr[3:0]` and in VHDL it becomes `i1:addr(3 downto 0)`. The parts of these names are as follows:

Name Item	Explanation
i1	An identifier (an instance name in this case)
slash (/), period (.), colon (:)	A hierarchy delimiter
addr	An identifier (a port/pin name in this case)
<3:0>, [3:0], (3 down to 0)	An index expression

Related Topics

The nmp Command

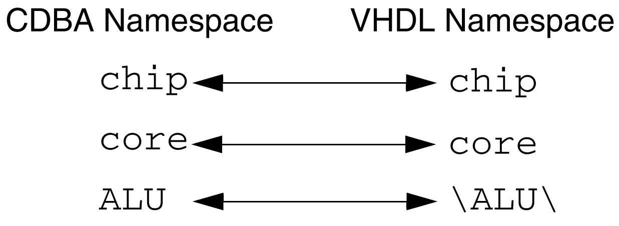
Rules for the Name Mapping Algorithm

Name Mapping SKILL Functions

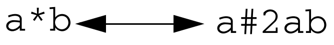
Recognizing Identifiers and Understanding Case Sensitivity

Identifiers are the atomic strings your application chooses to identify a design object uniquely. Identifiers are present in the names of many kinds of design objects. Names of nets, ports, instances, libraries, cells, views, and properties contain identifiers. To handle these names properly as they are referenced in different steps of a design flow, Cadence applications consistently apply the correct namespace for each identifier and follow the name mapping rules wherever names from different namespaces are compared.

The CDBA view specification `/chip/core/ALU` contains three identifiers: `chip`, `core`, and `ALU`. When you subsequently use this information in another application, the algorithm modifies these identifiers into legal identifiers in the target namespace. For example, in the VHDL namespace these identifiers look like `chip`, `core`, and `\ALU\`. Mapping goes both ways.



Illegal Characters in Identifiers



If an escaped namespace is not available for the target namespace, or if the identifier contains characters that are illegal, the algorithm uses character encoding. Characters that are illegal in the target namespace are replaced with a pound sign (#) followed by the character value encoded as two hexadecimal digits (0–9 or a–f); for example, `#2a`. The identifier `a*b` is represented as `a#2ab` in a namespace that does not support the asterisk (*).

Case Sensitivity

Most Cadence applications are case sensitive. However, some namespaces, such as VHDL, Concept, and LibraryNT, are case-insensitive.

To provide a one-to-one map between, for example, Verilog and VHDL, name mapping must provide normal alphabetic names. The Verilog name `StopGap` becomes `StopGap` in VHDL and the Verilog name `stopgap` becomes `stopgap` in VHDL, so, because VHDL is case insensitive, two different Verilog names become a single VHDL name.

To handle this situation, the name mapping rules map all case-insensitive letters as lowercase case-sensitive letters and vice-versa. Case-sensitive uppercase letters map to escaped names where these are available (VHDL and Concept). In the LibraryNT namespace, the algorithm precedes them with a percent sign (%) so that the Verilog name GORP becomes %G%O%R%P.

You can avoid name mapping issues by always choosing names that use only lowercase letters and digits.

Related Topics

The nmp Command

Name Mapping Rules

Rules for the Name Mapping Algorithm

Cadence uses algorithmic name mapping to map identifiers between different namespaces.

- Names can be mapped from one namespace to another using the legal identifier and the source and destination namespaces. The algorithm does not need auxiliary information such as design-specific mapping tables. The algorithm is designed to be context free so that no information about the design is necessary to perform the mapping.
- Every identifier that is legal in one namespace maps to a legal identifier in every other.
- Every pair of identifiers that are different names in one namespace map to different names in every other namespace.
- All mappings are reversible. Mapping an identifier from one namespace to another and then back to the original namespace always results in the original identifier, with the possible exception of a loss of case if the original namespace is case insensitive.

Algorithmic name mapping does not solve certain problems, such as mapping names to another representation where the design has been modified.

Examples of differences between namespaces are as follows:

Differences	**Examples of Namespace Rules**
Keywords	The string `and` is a keyword in Verilog and VHDL, while `process` is a keyword in VHDL but not in Verilog. Many namespaces, such as CDBA and LibraryUnix, have no keywords.
Case Sensitivity	In Concept and VHDL, the identifier `aaa` refers to the same object as `AAA`. In CDBA and LibraryUnix, these names are different.
Syntax and Characters	Many namespaces have an alternative way to include characters in names that would otherwise be illegal. In VHDL, `\a+b*\` is a legal identifier because the backslashes (\ ) escape the characters that are otherwise illegal. A normal Verilog name can contain a dollar sign ($), but a VHDL name cannot unless it is escaped.

Related Topics

Name Mapping Rules

The nmp Command

Cadence provides a command called `nmp` to help you understand name mapping. You can use this command, located in `*your_install_dir*/tools/bin`, to see how a name maps from one namespace to another, to check if a name is legal in a particular namespace, or to get a list of all the namespaces that Cadence applications use.

Use the following syntax for the `nmp` command:

```
nmp { getSpaceNames | isLegalName nameSpace identifier | mapName fromNameSpace
toNameSpace identifier | -v[ersion] | -help }
```

`getSpaceNames` Lists all the namespaces that Cadence applications use.

`isLegalName` *`nameSpace identifier`*

Checks if *`identifier`* is a legal name in the namespace you specify.

If you are specifying an escaped name or a name with special characters that the shell might delete, such as a backslash, enclose the name in single quotes. Also, in some shells, you might need to escape the backslash that is part of an escaped name with another backslash.

`mapName` *`fromNameSpace toNameSpace identifier`*

Maps a name (*`identifier`*) from one namespace to another.

If you are specifying an escaped name or a name with special characters that the shell might delete, such as a backslash, enclose the name in single quotes. Also, in some shells, you might need to escape the backslash that is part of an escaped name with another backslash.

`-v[ersion]` Displays the version of the `nmp` command.

`-help` Displays the syntax and usage of the `nmp` command.

Examples:

```
% nmp isLegalName Verilog buf_addr0
legal
```

```
% nmp mapName Concept VHDL procedure
\procedure\
```

Warning: When you enter names with backslashes, the shell might delete a backslash so that it is not seen by the `nmp` program. To avoid this, include all names in single quotes, such as *`'name'`*. For example:

```
% nmp mapName VHDL Concept \procedure\
**procedure is not a legal VHDL identifier**
```

```
% nmp mapName VHDL Concept ’\procedure\’
procedure
```

Warning: Verilog requires escaped names to end in a space. This space might not be visible in the output. For example:

```
% nmp mapName VHDL Verilog ’\2+2=4\’
\2+2=4
```

```
% nmp mapName Verilog VHDL ’\2+2=4␣’
\2+2=4\
```

Warning: In some shells, you need to escape the backslash that is part of an escaped name with another backslash. For example:

```
sh% nmp mapName Verilog LibraryUnix ’\!Lib!␣’
#21Lib#21
```

```
csh% nmp mapName Verilog LibraryUnix ’\\!Lib!␣’
#21Lib#21
```

Related Topics

Namespaces for Different Data Types

Rules for the Name Mapping Algorithm

Name Mapping Rules

Cadence applications adhere to the following name mapping rules:

- As much as possible, names are not modified. For example, the identifier *`abc`* is left the same in every supported namespace.
- When an identifier contains a character that is illegal in the namespace to which it is being mapped (even using escaping), the algorithm uses the hex value for the character, such as a character with the hex value of `2d` (a minus sign) becomes `#2d`.
- When case-insensitive names are converted to case-sensitive namespaces, the letters are set to lowercase.
- When case-sensitive names are mapped to a case-insensitive namespace, they are left alone if they are lowercase. If they are uppercase, the algorithm creates escaped names if the namespace has a case-sensitive escaped form, such as in VHDL and Concept.
- If an escaped name that is legal and is a different name when the escape characters are left off is mapped to a namespace where there is not a similar escaped form, then the algorithm prepends the characters `ESC_` to the mapped name. For example, in VHDL, `\aaa\` is a different name from `aaa`, so they cannot both map to `aaa` in Verilog. The name mapping rules map a VHDL `\aaa\` to `ESC_aaa` in Verilog.
- Length restrictions are not handled by these name mapping rules. The mapping routines can handle names up to 2,000 characters long.

Examples of Name Mapping

Type	VHDL	Verilog	CDBA	Concept	Library Unix	LibraryNT
lowercaseor case insensi-tive	bigchip BIGCHIP	bigchip	bigchip	bigchip BIGCHIP	bigchip	bigchip BIGCHIP
case-sensitivemixed case	\BigChip\	BigChip	BigChip	\BigChip\	BigChip	%Big%Chip
keyword	\and\	\and ␣	and	and	and	and
unneeded escape	\trash\	ESC_trash	ESC_trash	\trash\	ESC_trash	ESC_trash
embedded space	\foo bar\	\foo#20bar ␣	a#20b	foo bar	foo#20bar	foo#20bar
device names	aux	aux	aux	aux	aux	%%aux
back-slash	\\\a\\b\	\\a\b ␣	#5ca#5cb	‘\a\b’	#5ca#5cb	#5ca#5cb
forward slash	\/a/b\	\/a/b ␣	#2fa#2fb	/a/b	#2fa#2f	#2fa#2fb
special characters*	\a<1:2>\	\a<1:2> ␣	a#3c1:2#3e	“a<1:2>”	a#3c1#3a2#3e	a#3c1#3a2#3e

* Although the special characters look similar to a name with an index expression, this row shows an identifier, with special characters, that is being used as an atomic identifier. The ␣ character indicates a blank space.

Avoid Name Mapping Issues

Use these strategies for name handling:

- Choose names that use only lowercase letters, digits, and underscores.
- Use identifiers that start with an alphabetic character.
- Do not use identifiers that are keywords in any of the namespaces.
- Do not use characters that are not allowed in any of the namespaces.

Related Topics

ALT_NMP Namespace

Rules for the Name Mapping Algorithm

Checking the Names in your cds.lib File

Many Cadence applications include the cdsLibEditor tool.

To check the name (for example, for VHDL) in your `cds.lib` file,

➤ Type the following command in a shell window:

```
cdsLibEditor -namespace VHDL
```

The Library Path Editor opens, showing you library names in the VHDL namespace. In VHDL, the system maps the library name `TEST` as `test` because the library name stored in the `cds.lib` file is case sensitive.

Related Topics

Creating or Editing a cds.lib File

Namespaces for Different Data Types

Cadence applications use the following namespaces:

- VHDL
- VHDLAMS
- VHDL87
- Verilog
- VerilogA
- VerilogAMS
- ALT_NMP
- CDBA
- CDBAFlat
- Concept
- General Constraint Format (GCF)
- Genesis
- Library Exchange Format (LEF)
- Design Exchange Format (DEF)
- Library
- LibraryUnix
- LibraryNT
- Print
- Standard Delay Format (SDF)
- Standard Parasitic Format (SPF)
- Standard Parasitic Exchange Format (SPEF)
- Spectre
- SpectreHDL
- Spice

Related Topics

VHDL, VHDLAMS, and VHDL87 Namespaces

Verilog, VerilogA, and VerilogAMS Namespaces

SystemVerilog Namespace

ALT_NMP Namespace

CDBA Namespace

Concept Namespace

GCF Namespace

Genesis Namespace

LEF and DEF Namespaces

Library Namespace

Print Namespace

SDF Namespace

SPEF Namespace

SPF Namespace

Spectre Namespace

SpectreHDL Namespace

Spice Namespace

VHDL, VHDLAMS, and VHDL87 Namespaces

Each VHDL identifier is either in the normal form, such as `ABC`, or the escaped form, such as `\ABC\`. Most identifiers in a VHDL design are VHDL normal identifiers.

The VHDLAMS namespace is used in the AMS environment. VHDLAMS is identical to the VHDL namespace except that it has additional reserved keywords.

The VHDL87 namespace is used by some VHDL simulators such as NC-VHDL. VHDL87 is identical to the VHDL namespace except that some keywords that are reserved in VHDL are allowed as normal identifiers in VHDL87.

VHDL Normal Names

VHDL normal identifiers are not case insensitive. This means that the identifier `AbC` and the identifier `abc` are equivalent and refer to the same object. VHDL normal identifiers can contain alphabetical characters, digits, or the underscore character. The first letter must be an alphabetical character. The only symbol that is allowed is the underscore. Spaces are not allowed.

VHDL Escaped Names

VHDL escaped identifiers are case sensitive, unlike normal VHDL names. Escaped identifiers always begin and end with a backslash (`\`). This means that the identifier `\AbC\` and the identifier `\aBc\` refer to two different objects.

An identifier in the VHDL normal namespace and the same identifier in the VHDL escaped namespace do not represent the same object. For example, the VHDL identifiers `abc` and `\abc\` refer to two different objects. To embed a backslash in an escaped identifier, use double backslashes.

If the original identifier was in the VHDL escaped form even though it was legal in the VHDL normal form, it needs to be returned to the escaped form, not the normal form. For example, `\abc\` maps to CDBA `ESC_abc`, and it maps back to `\abc\` in VHDL.

All alphanumeric characters and symbols, as well as spaces, are allowed as VHDL escaped identifiers.

Reserved VHDL Keywords

The following keywords are reserved and cannot be used as VHDL identifiers.

abs
access
after
alias
all
allow
and
architecture
array
assert
attribute
begin
block
body
buffer
bus
case
component
configuration
constant
disconnect
downto
element
else
elseif
end
entity
exit
file
for
function
generate
generic
group
guarded
if
impure
in
inertial
inout
is
label
library
linkage
literal
loop
map
mod
nand
new
next
nor
not
null
of
on
open
or
others
out
package
port
postponed
private
procedure
process
pure
range
record
register
reject
rem
report
return
rol
ror
select
severity
signal
shared
sla
sll
sra
srl
subtype
then
to
transport
type
unaffected
units
until
use
variable
wait
when
while
with
xnor
xor

Reserved VHDLAMS Keywords

Reserved VHDL keywords are also reserved in VHDLAMS. In addition, the following keywords are also reserved in VHDLAMS and cannot be used as VHDLAMS identifiers.

across
break
limit
nature
noise
procedural
quantity
reference
spectrum
subnature
terminal
through
tolerance

Reserved VHDL87 Keywords

The following keywords are reserved and cannot be used as VHDL87 identifiers.

abs	access	after	alias	all	allow
and	architecture	array	assert	attribute	begin
block	body	buffer	bus	case	component
configuration	constant	disconnect	downto	element	else
elseif	end	entity	exit	file	for
function	generate	generic	guarded	if	in
inout	is	label	library	linkage	loop
map	mod	nand	new	next	nor
not	null	of	on	open	or
others	out	package	port	private	procedure
process	range	record	register	rem	report
return	select	severity	signal	subtype	then
to	transport	type	units	until	use
variable	wait	when	while	with	xor

Related Topics

Namespaces for Different Data Types

Name Mapping Rules

Verilog, VerilogA, and VerilogAMS Namespaces

Each Verilog identifier is either in the normal form `ABC` or the escaped form `\ABC`␣. Most identifiers in a Verilog design are Verilog normal identifiers. If you need to represent identifiers that are illegal as a Verilog normal identifier, use the Verilog escaped namespace. For example, the identifier `and` in the CDBA namespace maps to the Verilog escaped `\and`␣ because `and` is a Verilog keyword.

The VerilogA analog simulator uses the VerilogA namespace. The namespace is the same as the Verilog namespace, except that it has additional reserved keywords.

The AMS environment uses the VerilogAMS namespace. The VerilogAMS namespace is the same as the Verilog namespace, except that it has additional reserved keywords.

Note: The ␣ character in Verilog names is used to clearly document a blank space.

Verilog Normal Names

Verilog normal identifiers are case sensitive. This means that the identifier *`AbC`* and the identifier *`abc`* refer to two different objects. Verilog normal identifiers might contain letters, digits, the underscore character, and the dollar sign ($). The first character cannot be a digit or a dollar sign ($). All letters and digits are allowed as Verilog normal identifiers. The space is not allowed.

Verilog Escaped Names

Verilog escaped identifiers always begin with a backslash (`\`) and terminate with a space. Verilog escaped identifiers are case sensitive. This means that the identifier `\AbC`␣ and the identifier `\aBc`␣ refer to two different objects.

Although any identifier can be escaped, only those identifiers that might not be represented in the Verilog normal namespace due to character restrictions or because they are keywords need to be escaped. An identifier in the Verilog normal namespace and the identifier in the Verilog escaped namespace represent the same object. For example, the Verilog identifiers `abc` and `\abc`␣ refer to the same object.

All alphanumeric characters and symbols are allowed in Verilog escaped identifiers. Blanks or nonprinting characters are not allowed.

Reserved Verilog Keywords

The following keywords are reserved in Verilog and cannot be used as Verilog identifiers.

always
and
assign
attribute
begin
buf
bufif0
bufif1
case
casex
casez
class
cmos
deassign
default
defparam
disable
edge
else
end
endattribute
endcase
endfunction
endmodule
endpackage
endprimitive
endspecify
endtable
endtask
event
for
force
forever
fork
function
highz0
highz1
if
ifnone
initial
inout
input
integer
join
large
macromodule
medium
module
name
nand
negedge
nmos
nor
not
notif0
notif1
or
output
package
parameter
pmos
posedge
primitive
pull0
pull1
pulldown
pullup
rcmos
real
realtime
reg
release
repeat
rnmos
rpmos
rtran
rtranif0
rtranif1
scalared
signed
small
specify
specparam
strength
string
strong0
strong1
supply0
supply1
table
task
time
tran
tranif0
tranif1
tri
tri0
tri1
triand
trior
trireg
use
vectored
wait
wand
weak0
weak1
when
while
wire
wor
xnor

Reserved VerilogA Keywords

The following keywords are reserved in VerilogA and cannot be used as VerilogA identifiers.

above
abs
absdelay
ddt
ddt_nature
cosh
final_step
flicker_noise
flow
limexp
ln
localparam
pullup
real
realtime
tran
tranif0
tranif1

abstol
access
acos
acosh
ac_stim
aliasparam
always
analog
analysis
and
asin
asinh
assign
atan
atan2
atanh
begin
bound_step
branch
buf
bufif0
bufif1
case
casex
casez
ceil
cmos
connectrules
cross
ddx
deassign
default
defparam
delay
disable
discipline
discontinuity
driver_update
edge
else
end
endcase
endconnectrules
enddiscipline
endfunction
endmodule
endnature
endparamset
endprimitive
endspecify
endtable
endtask
event
exclude
exp
floor
for
force
fork
from
function
generate
genvar
ground
highz0
highz1
hypot
idt
idtmod
idt_nature
if
ifnone
inf
initial
initial_step
inout
input
integer
join
laplace_nd
laplace_np
laplace_zd
laplace_zp
large
log
max
medium
micromodule
min
module
nand
nature
negedge
net_resolution
nmos
noise_table
nor
not
notif0
notif1
or
output
parameter
pmos
posedge
potential
pow
primitive
pull0
pull1
pulldown
pwr
reg
release
repeat
rnmos
rpmos
rtran
rtranif0
rtranif1
scalared
sin
sinh
slew
small
specify
specparam
sqrt
strobe
strong0
strong1
supply0
supply1
table
table_model
tan
tanh
task
temperature
time
transition
tri
tri0
tri1
triand
trior
trireg
units
vectored
vt
wait
wand
weak0
weak1
while
white_noise
wire
wor
wreal
xnor
xor
zi_nd
zi_np
zi_zd

cos	forever	last_crossing	rcmos	timer

Reserved VerilogAMS Keywords

Reserved Verilog keywords are also reserved in VerilogAMS. In addition, the following keywords are also reserved in VerilogAMS and cannot be used as VerilogAMS identifiers.

above	abs	abstol	ac_stim
access	acos	acosh	a2d
analog	aliasparam	analysis	asin
asinh	atan	atan2	atanh
attribute	bound_step	branch	ceil
class	connect	connectmodule	connectrules
cross	continuous	cos	cosh
delay	ddt	ddt	ddt_nature
domain	discipline	discontinuity	discrete
driver_local	driver_active	driver_count	driver_delay
driver_strength	driver_next_state	driver_next_strength	driver_state
endattripute	driver_update	dynamicparam	endconnectrules
endnature	enddiscipline	endparamset	endpackage
flicker_noise	exclude	exp	final_step
generate	floor	flow	from
idt	genvar	ground	hypot
initial_step	idt_nature	idtmod	inf
laplace_zp	laplace_nd	laplace_np	laplace_zd
localparam	last_crossing	log	limexp
ln	max	merged	min
nature	net_resolution	noise_table	noshowcancelled
paramset	pathdelay_controlsignal	pathdelay_max0	pathdelay_max1
pathdelay_min0	pathdelay_min1	pathdelay_sense	potential

pow	pragma	pulsestyle_ondetect	pulsestyle_onevent
real2int	resolveto	showcancelled	signed
sin	sinh	slew	split
strength	sqrt	tan	tanh
timer	to	transition	use
using	units	when	white_noise
with	wreal	zi_nd	zi_np
zi_zd	zi_zp		

Related Topics

Name Mapping SKILL Functions

Namespaces for Different Data Types

Name Mapping Rules

SystemVerilog Namespace

SystemVerilog has eight namespaces for identifiers, which includes:

1. Definitions namespace: Merges all the non-nested module, macromodule, primitive, program, and interface identifiers defined outside of all other declarations.
2. Package namespace: Merges all the package identifiers defined among all compilation units. Once a name is used to define a package within one compilation unit the name shall not be used again to declare another package within any compilation unit.
3. Compilation-unit scope namespace: Exists outside the module, macromodule, interface, package, program, and primitive constructs. It merges the definitions of the functions, tasks, parameters, named events, net declarations, variable declarations and user defined types within the compilation-unit scope.
4. Text macro namespace: Is global within the compilation unit. Since text macro names are introduced and used with a leading ‘ character, they remain unambiguous with any other namespace. The text macronames are defined in the linear order of appearance in the set of input files that make up the compilation unit.
5. Module namespace: Merges the definition of modules, macromodules, interfaces, programs, functions, tasks, named blocks, instance names, parameters, named events, net declarations, variable declarations and user defined types within the enclosing construct.
6. Block namespace: Merges the definitions of the named blocks, functions, tasks, parameters, named events, variable type of declaration and user defined types within the enclosing construct.
7. Port namespace: Provides a means of structurally defining connections between two objects that are in two different namespaces.
8. Attribute namespace: Is enclosed by the (* and *) constructs attached to a language element. An attribute name can be defined and used only in the attribute namespace. Any other type of name cannot be defined in this namespace.

Listing the SystemVerilog supported and non-supported keywords:

- Keywords Supported in SystemVerilog

  implements

  interconnect

  nettype

  soft

- Keywords Not Allowed as SystemVerilog Identifiers

  accept_on
  alias
  always
  always_comb
  always_ff
  always_latch
  and
  assert
  assign
  assume
  automatic
  before
  begin
  bind
  bins
  binsof
  bit
  break
  buf
  bufif0
  bufif1
  byte
  case
  casex
  casez
  cell
  chandle
  class
  clocking
  cmos
  config
  const
  constraint
  context
  continue
  cover
  covergroup
  coverpoint
  cross*
  deassign
  default
  defparam
  design
  disable
  dist
  do
  edge
  else
  end
  endcase
  endchecker
  endclass
  endclocking
  endconfig
  endfunction
  endgenerate
  endgroup
  endinterface
  endmodule
  endpackage
  endprimitive
  endprogram
  endproperty
  endspecify
  endsequence
  endtable
  endtask
  enum
  event
  expect
  export
  extends
  extern
  implies
  final
  first_match
  for
  force
  foreach
  forever
  fork
  forkjoin
  function
  generate

genvar
highz0
highz1
if
iff
ifnone
ignore_bins
illegal_bins
import
incdir
include
initial
inout
input
inside
instance
int
integer
interface
intersect
join
join_any
join_none
large
liblist
library
local
localparam
logic
longint
macromodule
matches
medium
modport
module
nand
negedge
new
nmos
nor
noshowcancelled
not
notif0
notif1
null
or
output
package
packed
parameter
pmos
posedge
primitive
priority
program
property
protected
pull0
pull1
pulldown
pullup
pulsestyle_onevent
pulsestyle_ondetect
pure
rand
randc
randcase
randsequence
rcmos
real
realtime
ref
reg
release
repeat
return
rnmos
rpmos
rtran
rtranif0
rtranif1
scalared
sequence
shortint
shortreal
showcancelled
signed

small
solve
specify
specparam
static
string
strong0
strong1
struct
super
supply0
supply1
table
tagged
task
this
throughout
time
timeprecision
timeunit
tran
tranif0
tranif1
tri
tri0
tri1
triand
trior
trireg
type
typedef
union
unique
unsigned
use
var
vectored
virtual
void
wait
wait_order
wand
weak0
weak1
while
wildcard
wire
with
within
wor
xnor
xor

Related Topics

Name Mapping SKILL Functions

Namespaces for Different Data Types

Name Mapping Rules

ALT_NMP Namespace

The ALT_NMP namespace is an alternative method for name-space mapping in a mixed VHDL and Verilog environment. It deals with inconsistencies between the VHDL and Verilog namespaces.

The ALT_NMP namespace is set with the following environment variable:

```
CDS_ALT_NMP = MATCH
```

In the ALT_NMP namespace, you cannot have two names that vary only in case. For example, do not name a library `MyLib` when you already have a library `mylib`.

The ALT_NMP namespace has the following name mapping rules:

- When case-sensitive and case-insensitive namespaces interact, the case-sensitive namespace is treated as case-insensitive. However, escaped names from the case-insensitive namespace are not changed. The mapping of escaped names in this case is irreversible. For example, `\AbC\` in the VHDL namespace is irreversibly mapped to `AbC` in the Verilog namespace.
- When names in case-insensitive namespaces are mapped to the LibraryUnix namespace, they are treated as lower case. When names in case-sensitive namespaces are mapped to the LibraryUnix namespace, they are not changed.

You might face case-sensitive issues while compiling the VHDL-AMS entities, to resolve these issues, see Advice and Solutions for VHDL-AMS Compiler Issues.

Related Topics

Name Mapping SKILL Functions

Namespaces for Different Data Types

Name Mapping Rules

CDBA Namespace

CDBA normal identifiers are case-sensitive. This means that the identifier *`AbC`* and the identifier *`abc`* refer to two different objects.

All alphanumeric characters and symbols are allowed in CDBA normal identifiers except the comma (,), backslash (\ ), forward slash (/), and the opening (<) and closing angle brackets (>). Spaces are not allowed. Opening and closing parentheses are allowed with an identifier only if they enclose digits.

Note: CDBA does not use escaped names.

CDBAFlat Namespace

The CDBAFlat namespace is used by applications. CDBAFlat is identical to the CDBA namespace except for the following:

- The pipe character (|) is used as the hierarchy delimiter character instead of the forward slash character (`/`)
- Bus bit characters are enclosed in parentheses (`()`) instead of angle brackets (`< >`)

Related Topics

Name Mapping SKILL Functions

Namespaces for Different Data Types

Name Mapping Rules

Concept Namespace

The Concept namespace has many similarities to the VHDL namespace. It has both a normal and an escaped form. The normal form is case insensitive; the escaped form is used only to provide case sensitivity. The escaped form starts and ends with a backslash (\ ), and double backslashes (\\ ) can be used to include a backslash in the identifier. Escaped identifiers and normal identifiers that are otherwise identical denote different objects.

The normal form also allows quoting. You can use either a quotation mark (") or an apostrophe ('), and you can start the quoting anywhere in the identifier. You must match an opening quotation mark or apostrophe with a closing one of the same type (" or '). You can use a quotation mark to include an apostrophe, and use an apostrophe as part of the identifier. Hence `"ab'cd"` and `'ab''cd'` are equivalent identifiers that contain an apostrophe.

The Concept namespace has no keywords. Spaces are allowed except at the beginning and ends of a normal identifier.

All printable ASCII characters might be used in a normal identifier except for the apostrophe ('), the quotation mark ("), the angle bracket (<) and the colon (:). These four characters can be used in quoted names where the quotation mark is different from the included quotation mark or else where the included apostrophe is doubled. Spaces can also be used if they are not the first or last character. If quoted, this means the first or last character other than the quotation mark.

Escaped identifiers can contain any of the above characters (all printable graphic ASCII characters). Only the backslash (\ ) needs to be doubled to be included. Blank spaces are legal anywhere in an escaped name.

Related Topics

Name Mapping SKILL Functions

Namespaces for Different Data Types

Name Mapping Rules

GCF Namespace

The General Constraint Format (GCF) namespace is case sensitive. Names cannot be longer than 1024 characters.

GCF Normal Names

GCF normal names can contain only alphanumeric characters (`a-z`, `A-Z`, `0-9`) and the underscore character (_). In addition, the following special characters are allowed if they are specified as delimiters in the header section of the GCF file:

- The hierarchy divider character

  If the hierarchy divider character is not defined in the header section of the GCF file, the default hierarchy divider is the period (`.`).

- The bus bit index characters—usually the left and right square brackets, parentheses, or angle brackets (`[] () <>`). The index characters can enclose one positive integer to represent a single object, a pair of positive integers separated by a colon to represent a range, or an asterisk to represent all objects in the array. For example: `pipe[4]`, `pipe[1:3]`, or `pipe[*]`

  If the bus bit index characters are not specified in the header section of the GCF file, the default index characters are the left and right angle brackets (`<>`).

Normal names cannot include any other characters unless they are escaped. White space characters, such as tabs, spaces, or newlines are not allowed.

GCF Escaped Names

You can use characters that are not allowed in GCF normal names by escaping them. A character is escaped when it is preceded by a backslash character (`\`).

The following characters can be used in GCF names if they are escaped:

```
! “ # $ % & ‘ ( ) * + , - . / : ; < = > ? @ [ \ ] ^ ` { | } ~
```

The following examples show how escaped characters can be used in GCF names:

```
AMUX\+BMUX
pipe4\-done\&enb[3]
```

Related Topics

Name Mapping SKILL Functions

Namespaces for Different Data Types

Name Mapping Rules

Genesis Namespace

The Genesis namespace is case sensitive. This means that the identifier `AbC` and the identifier `abc` refer to two different objects.

Genesis Normal Names

Genesis normal names can contain any printable ASCII characters—which include alphanumeric characters (`a-z`, `A-Z`, `0-9`), the underscore character (_), and control characters— except the following special characters:

`|` Hierarchy delimiter character

`[ ]` Bus bit characters

Genesis Escaped Names

You can use characters that are not allowed in Genesis normal names by escaping them. A character is escaped when it is preceded by a backslash character (`\`).

Related Topics

Name Mapping SKILL Functions

Namespaces for Different Data Types

Name Mapping Rules

LEF and DEF Namespaces

The Library Exchange Format (LEF) and Design Exchange Format (DEF) namespaces are case sensitive.

LEF and DEF Normal Names

LEF and DEF names can contain any ASCII characters except the semicolon (`;`), pound sign (`#`), newline (`\n`), or space characters.

In addition, the following special characters can be used in LEF and DEF names:

- Characters that denote regular expressions:

  `*` Matches any sequence of characters

  `-` Matches any sequence of characters up to the next period (`.`)

  `%` Matches a single character

- Bus bit index characters, as defined in the `BUSBITCHARS` statement

  The default index characters are the left and right square brackets (`[]`).

- Hierarchy divider character, as defined in the `DIVIDERCHAR` statement

  The default hierarchy divider character is the forward slash (`/`).

LEF and DEF Escaped Names

You can use characters that are not allowed in LEF and DEF normal names by escaping them. A character is escaped when it is preceded by a backslash character (`\`).

Note: You cannot escape the pound sign (`#`).

Reserved LEF Keywords

The following keywords are reserved in the LEF namespace and cannot be used as LEF identifiers.

abut	abutment	accurrentdensity
active	analog	and
antennaareafactor	antennalengthfactor	antennametalarea

antennametallength	antennasize	anyedge
array	average	beginext
block	bottomleft	bottomright
buffer	busbitchars	by
cannotoccupy	canplace	capacitance
capmultiplier	class	clock
clocktype	columnmajor	componentpin
components	core	corner
correctionfactor	correctiontable	cover
cpersqdist current	currentden	currentsource
cut	cutarea	data
database	dccurrentdensity	default
defaultcap	delay	dielectric
direction	dividerchar	do
edgecapacitance	edgerate	edgeratescalefactor
edgeratethreshold1	edgeratethreshold2	eeq
else	end	endcap
endext	extension	fall
fallcs	fallrs	fallsatcur
fallsatt1	fallslewlimit	fallt0
fallthresh	fallvoltagethreshold	false
fe	feedthru	fixed
floorplan	fn	foreign
frequency	frompin	fs
function	fw	gcellgrid
generate	generator	ground
height	history	hold
horizontal	if	inout
inoutpinantennasize	input	inputnoisemargin

inputpinantennasize	integer	intrinsic
invert	inverter	irdrop
iterate	iv_tables	layer
leakage	lengththreshold	leq
library	macro	masterslice
match	maxdelay	maxload
megahertz	metaloverhang	milliamps
milliwatts	minfeature	minpins
mpwh	mpwl	mustjoin
mx	mxr90	my
myr90	namemapstring	namescasesensitive
nanoseconds	negedge	nets
new	noisetable	nondefaultrule
noninvert	nonunate	nowireextensionatpin
obs	off	offset
ohms	on	or
orientation	origin	output
outputnoisemargin	outputpinantennasize	outputresistance
outputresistance	overhang	overlap
overlaps	pad	path
pattern	peak	period
picofarads	pin	pitch
placed	polygon	port
posedge	post	power
pre	property	propertydefinitions
pulldownres	pwl	r0
r180	r270	r90
range	real	recovery
rect	resistance	resistive

ring	rise	risecs
risers	risesatcur	risesatt1
riseslewlimit	riset0	risethresh
risevoltagethreshold	rms	routing
rowmajor	rpersq	samenet
scanuse	sdfcond	sdfcondend
sdfcondstart	setup	shape
shrinkage	signal	site
size	skew	source
spacer	spacing	specialnets
stable	stack	start
step	stop	string
structure	symmetry	table
tableaxis	tabledimension	tabledimension
tableentries	taperrule	then
thickness	tiehigh	tielow
tieoffr	time	timing
to	topin	topleft
topofstackonly	topright	tracks
transitiontime	tristate	true
type	unateness	units
universalnoisemargin	use	uselengththreshold
user	variable	version
vertical	vhi	via
viarule	victimlength	victimnoise
virtual	vlo	voltage
volts	w	width
wirecap	wireextension	x
y		

Reserved DEF Keywords

The following keywords are reserved in the DEF namespace and cannot be used as DEF identifiers.

align	array	assertions
beginext	bottomleft	busbitchars
by	cannotoccupy	canplace
capacitance	commonscanpins	component
componentpin	components	constraints
cover	defaultcap	design
diearea	diff	direction
distance	dividerchar	do
drivecell	e	eeqmaster
end	endext	equal
estcap	fall	fallmax
fallmin	fe	fixed
floating	floorplan	floorplanconstraints
fn	foreign	fromclockpin
fromcomppin	fromiopin	frompin
fs	fw	gcellgrid
generate	group	groups
history	holdfall	holdrise
horizontal	in	integer
iotimings	layer	max
maxdist	maxhalfperimeter	maxx
maxy	microns	min
minpins	mustjoin	n
namemapstring	namescasesensitive	net
nets	new	nondefaultrule
noshield	ordered	original

out	parallel	partitions
path	pattern	patternname
pin	pinproperties	pins
placed	property	propertydefinitions
range	real	rect
reentrantpaths	region	regions
rise	risemax	risemin
routed	row	rows
s	scanchains	setupfall
setuprise	shape	shield
shieldnet	site	slewrate
soft	source	spacing
special	specialnet	specialnets
start	step	stop
string	style	subnet
sum	synthesized	taper
taperrule	technology	timingdisables
toclockpin	tocomppin	toiopin
topin	topright	tracks
turnoff	units	unplaced
use	variable	version
vertical	vias	voltage
vpin	w	weight
width	wirecap	wiredlogic
xtalk		

Related Topics

Name Mapping SKILL Functions

Library Namespace

The name mapping software supports a Library namespace that maps to either the LibraryUnix or the LibraryNT namespace, depending on which operating system the software is running. For programs running on UNIX, it is the same as the LibraryUnix namespace. For programs running on Windows, it is the same as the LibraryNT namespace.

LibraryUnix Namespace

LibraryUnix is used for library names in `cds.lib` files. It is also used for cell and view directory names in libraries that are stored on UNIX systems. LibraryUnix is designed to follow the rules for creating directory names on UNIX systems, except that it is more restrictive. Many characters, such as the period (.) and the comma (,), which are legal in the file system, are not allowed in LibraryUnix.

LibraryUnix is a case-sensitive namespace; for example, the filenames `abc` and `AbC` refer to different directories.

All alphanumeric characters plus the underscore (_), at sign (@), and pound sign (#) are allowed in LibraryUnix identifiers.

LibraryNT Namespace

The LibraryNT namespace is used for cell and view subdirectories in Windows libraries. LibraryNT is a case-insensitive, but case-preserving, namespace. This means the filenames `abc` and `AbC` are the same file, and only one of them can exist in any given directory. Digits, the underscore character (_), the at sign (@), the percent sign (%), and the pound sign (#) are allowed. Several names are reserved by Windows. These might be thought of as keywords that are illegal names. The keywords are `aux`, `con`, `com0` through `com9`, `lpt0` through `lpt9`, `nul`, and `prn`.

LibraryNT uses the pound sign (#) followed by two hex digits to map characters that are otherwise illegal.

LibraryNT uses the percent sign (%) to map case-sensitive uppercase letters. An `X` in Verilog becomes `%X` in LibraryNT.

LibraryNT also uses the percent sign (%) to map names that would otherwise look like reserved names. For example, a cell named `lpt4` in a UNIX library is called `%%lpt4` in a Windows library.

Related Topics

Name Mapping SKILL Functions

Namespaces for Different Data Types

Name Mapping Rules

Print Namespace

The Print namespace is case sensitive. This means that the identifier `AbC` and the identifier `abc` refer to two different objects.

Names in the Print namespace can contain any printable ASCII characters, which include alphanumeric characters (`a-z`, `A-Z`, `0-9`) and control characters.

There is no escaping mechanism in the Print namespace.

Related Topics

Name Mapping SKILL Functions

Namespaces for Different Data Types

Name Mapping Rules

SDF Namespace

The Standard Delay Format (SDF) namespace is case sensitive. This means that the identifier `AbC` and the identifier `abc` refer to two different objects.

SDF names cannot be longer than 1024 characters.

SDF Normal Names

SDF normal names can contain only alphanumeric characters (`a-z`, `A-Z`, `0-9`) and the underscore character (_). In addition, the following special characters are allowed:

- The hierarchy divider character

  If the hierarchy divider character is not specified in the header section of the SDF file, the default hierarchy divider is the forward slash (`/`).

- Bus bit index characters—square brackets that enclose a positive integer to represent a single object or a pair of positive integers separated by a colon to represent a range. For example: `pipe4[3]` or `pipe4[0:2]`. The default index characters are the left and right square brackets (`[]`).

SDF normal names cannot include any other characters unless they are escaped. White space characters, such as tabs, spaces, or newlines, are not allowed.

SDF Escaped Names

You can use characters that are not allowed in SDF normal names by escaping them. A character is escaped when it is preceded by a backslash character (\).

The following characters can be used in SDF names if they are escaped:

```
! “ # $ % & ‘ ( ) * + , - . / : ; < = > ? @ [ \ ] ^ ‘ { | } ~
```

The following examples show how escaped characters can be used in SDF names:

```
AMUX\+BMUX
pipe4\-done\&enb[3]
```

Related Topics

Name Mapping SKILL Functions

SPF Namespace

The Standard Parasitic Format (SPF) namespace is case sensitive. This means that the identifier `AbC` and the identifier `abc` refer to two different objects.

SPF Normal Names

SPF normal names can contain any ASCII character except white space characters such as space or tab. The following special characters are allowed:

- The hierarchy divider character, as defined in the header section of the SPF file. If the hierarchy divider character is not defined, the default hierarchy divider is the forward slash (`/`).
- Bus bit index characters, as defined in the header section of the SPF file. The index characters can enclose only one positive integer; a range is not allowed. If the bus bit index characters are not defined in the header section of the SPF file, the default index characters are the left and right square brackets (`[]`).

SPF Escaped Names

You can use characters that are not allowed in SDF normal names by escaping them. A character is escaped when it is preceded by a backslash character (\).

Related Topics

Name Mapping SKILL Functions

Namespaces for Different Data Types

Name Mapping Rules

SPEF Namespace

The Standard Parasitic Exchange Format (SPEF) namespace is case sensitive. This means that the identifier `AbC` and the identifier `abc` refer to two different objects.

SPEF Normal Names

SPEF normal names can contain only alphanumeric characters (`a-z`, `A-Z`, `0-9`) and the underscore character (_). In addition, the hierarchy divider character and the bus bit index characters are allowed. The default hierarchy divider character is the forward slash (/); the default index characters are the left and right square brackets (`[]`).

White space characters, such as space or tab, and control characters, such as newline, are not allowed.

SPEF Escaped Names

You can use characters that are not allowed in SPEF normal names by escaping them. A character is escaped when it is preceded by a backslash character (`\`).

The following characters can be used in SDF names if they are escaped:

```
! “ # $ % & ‘ ( ) * + , - . / : ; < = > ? @ [ \ ] ^ ‵ { | } ~
```

Related Topics

Name Mapping SKILL Functions

Namespaces for Different Data Types

Name Mapping Rules

Spectre Namespace

The Spectre namespace is case sensitive. This means that the identifier `AbC` and the identifier `abc` refer to two different objects.

Spectre Normal Names

Spectre normal identifiers must be in one of the following forms:

- The identifier must begin with an alphabetical character (`a-z`, `A-Z`) or an underscore (_) and can only be followed by zero or more of the following characters: alphanumeric characters (`a-z`, `A-Z`, `0-9`), the underscore character (_), and the exclamation mark (`!`).

  For example:

  ```
  abc8
  Abc!
  _ABC
  ```

- The identifier must begin with one or more digits (`0-9`), followed by one or more alphabetical or underscore characters (`a-z`, `A-Z`, _), followed by a digit, and end with zero or more alphanumeric, underscore, or exclamation characters (`a-z`, `A-Z`, `0-9`, _, `!`).

  For example:

  ```
  1a2abc
  11a45
  1asdfasdf4
  6adfsa324!
  ```

- The identifier must consist of one or more digits (0-9).

  For example:

  ```
  1
  135
  ```

Spaces are not allowed in Spectre identifiers.

Spectre Escaped Names

Characters are escaped in the Spectre namespace if they are preceded by a backslash (`\`). For example, the escaped form of `a>=` is `a\>\=`

The following characters are allowed in Spectre identifiers if they are escaped:

```
- ! " # $ % & ' () * + , . / \ 0-9 : ; < > = ? @ A-Z a-z [ ] ^ _ ` { } | ~
```

The `TAB` character is also allowed if it is escaped.

The following examples show how escaped characters can be used in Spectre identifiers:

```
ab\@c
\@aBc
1\$2abc8
1\23
1\\2!
```

Reserved Spectre Keywords

The following keywords are reserved in the Spectre namespace and cannot be used as Spectre identifiers.

abs	acos	acosh	altergroup
asin	asinh	atan	atan2
atanh	ceil	correlate	cos
cosh	else	end	ends
exp	export	floor	for
function	global	hypot	ic
if	inline	int	library
local	log	log10	M_1_PI
M_2_PI	M_2_SQRTPI	M_DEGPERRAD	M_E
M_LN10	M_LN2	M_LOG10E	M_LOG2E
M_PI	M_PI_2	M_PI_4	M_SQRT1_2
M_SQRT2	M_TWO_PI	march	max
min	model	nodeset	P_C
P_CELSIUS0	P_EPS0	P_H	P_K
P_Q	P_U0	parameters	paramset
plot	pow	print	real
return	save	sens	sin
sinh	sqrt	statistics	subckt
tan	tanh	to	truncate
vary			

Related Topics

Name Mapping SKILL Functions

Namespaces for Different Data Types

Name Mapping Rules

SpectreHDL Namespace

The SpectreHDL namespace is case sensitive. This means that the identifier `AbC` and the identifier `abc` refer to two different objects.

SpectreHDL Normal Names

SpectreHDL normal identifiers can contain only alphabetical characters (`a-z`, `A-Z`), digits (`0-9`), and underscores (_). The identifiers must begin with an alphabetical character or an underscore.

SpectreHDL Escaped Names

Escaped identifiers in the SpectreHDL namespace must begin with a backslash character (`\`) and end with a space (space, tab, or newline character). Escaped names can contain any printable ASCII character.

Reserved SpectreHDL Keywords

The following keywords are reserved in the SpectreHDL namespace and cannot be used as SpectreHDL identifiers.

abs	abstol	ac_stim	acos
acosh	always	analog	analysis
ascii	asin	asinh	assert
assign	atan	atan2	atanh
bound_step	branch	break	break_point
build_string	build_table	bus	case
ceil	complex	const	continue
cos	cosh	cross	debug
default	do	domain	dot
else	enum	error	event
exclude	exp	export	fatal
fclose	fdebug	fflush	final

finish	flicker_noise	floor	flow
fopen	for	fread	fread_table
freq	from	fstrobe	fwrite
fwrite_table	generate	global	halt
hypot	idtmod	if	inf
initial	inout	input	integ
integer	interpolate	laplace_nd	laplace_np
laplace_zd	laplace_zp	last_crossing	limexp
ln	log	mag	max
min	model	module	negedge
net	netlist_node_alias	node	noise_table
notice	output	param_given	parameter
pclose	phase	popen	posedge
pow	pwl	pwr	quantity
random	read	real	reg
reject_step	reltol	return	sin
sinh	slew	sqrt	state
step	stop	str	strcat
strchr	strcmp	strcpy	strcspn
stream	string	strlen	strobe
strrchr	strspn	strstr	strtoint
strtoreal	substr	switch	system
table	tan	tanh	tdelay
temp	terminal	threshold	time
transition	unit	val	void
vt	wait	warning	when
while	white_noise	wire	write
zdelay	zi_nd	zi_np	zi_zd
zi_zp			

Related Topics

Name Mapping SKILL Functions

Namespaces for Different Data Types

Name Mapping Rules

Spice Namespace

The Spice namespace is case insensitive. This means that the identifier `AbC` and the identifier `abc` refer to the same object.

Spice Normal Names

Spice identifiers must begin with an alphabetical character. Identifiers can contain alphabetical characters (`a-z`, `A-Z`), digits (`0-9`), and the following characters: `# % * - < > [ ] _ $ ! + /`

Spice Escaped Names

Any character that is preceded by a backslash (`\`) is escaped in the Spice namespace.

Reserved Spice Keywords

The following keywords are reserved in the Spice namespace and cannot be used as Spice identifiers.

ac	aci	agauss	all
am	at	aunif	avg
bart	beta	bisection	black
brief	cross	current	data
dc	debug	dec	deriv
derivative	dtemp	enddata	err1
exp	fall	fil	find
freelib	freq	from	fs
gauss	goal	hamm	hann
harris	ic	ignor	integ
integral	kaiser	lam	last
limit	lin	max	maxfld
mer	min	minval	model
monte	nodeset	none	norm

numf	oct	opt	optimize
par	passfail	pe	pl
plot	poi	power	pp
pu	pulse	pwl	r
rd	rect	results	rin
rise	rms	rout	sffm
sin	start	sweep	targ
td	temp	to	tol
top	trig	uic	unif
unorm	voltage	weight	when
yin	ymax	ymin	yout
zin	zout		

Related Topics

Name Mapping SKILL Functions

Namespaces for Different Data Types

Name Mapping Rules

8

Generic Design Management (GDM) Commands

Cadence provides a facility known as Generic Design Management (GDM). This facility implements an interface that CAD applications, which make direct DM system calls, can use so that they can work with many different design management systems without having any special knowledge of those systems.

Part of the GDM facility is a set of shell commands that can be used to perform design management operations. These commands are also used by some Cadence® applications that perform these operations from shell scripts.

GDM is designed to support data stored in Cadence Library Structure libraries, including properly handling co-managed sets and other data that must be accessed in a synchronized fashion. By using GDM, you avoid the need to understand the Cadence library structure when using any design management system.

GDM recognizes data formats defined in multiple installation hierarchies.

GDM Concepts

GDM is based on a simple DM system model, which defines 12 operations or commands based on the following concepts:

Concept	Description
`workarea`	A directory hierarchy in the file system, where a related set of designs and other data including Cadence design libraries are stored.
`version`	A design management system–specific identifier of an instance of a file, which GDM handles in an opaque manner, expecting the users and underlying design management systems to understand the meaning of the identifier.

Concept	Description
`repository`	Associated with each design management system installation. The repository knows what files are managed, where they are stored, and what versions of those files exist and are accessible by users.
`wrapping`	Integrates your chosen design management system into GDM. It consists of a shared library that is dynamically loaded and a set of design management–specific commands that GDM executes. The Cadence Services organization offers a service for implementing a wrapping of a specific design management system.

Using Third-Party GDM Integrations

If you are using a third-party GDM integration (or wrapping), such as Synchronicity, Inc. DesignSync, and you cannot copy the shared libraries into the Cadence installation hierarchy, do the following:

1. Set the following environment variable that enables GDM to look for third-party shared libraries outside the Cadence installation hierarchy:

   ```
   setenv GDM_USE_SHLIB_ENVVAR yes
   ```

   You can also use the `CDS_GDM_SHLIB_LOCATION` environment variable to set the path to where the DM library can be found. Multiple paths can be set and GDM will investigate each of them until the DM shared library is found. For example:

   ```
   CDS_GDM_SHLIB_LOCATION=/tools/icmanage/gdm/lib64:/tools/
   icmanage/gdm/lib
   ```

2. Add the location of the shared libraries to the platform-specific library path environment variable. The library path environment variable is

`LD_LIBRARY_PATH`	on Sun
`LIBPATH`	on IBM RS
`LD_LIBRARY_PATH`	on Linux

   If both 32-bit and 64-bit program versions are used, both library locations should be specified. The 32-bit directories are located in `<path>/lib`, while the 64-bit libraries can be found in `<path>/lib/64bit`.

   GDM then looks for shared libraries in the locations specified by the platform-specific library path environment variable.

GDM Environment Variables

The following environment variables are used by GDM:

Environment Variables	Description
`GDMNOTLOADLIB`	Specifies that no design management system be used. When this variable is set, no DM operation will be executed and no shared library will be loaded.
`GDM_USE_SHLIB_ENVVAR`	Enables GDM to look for shared libraries outside the Cadence installation hierarchy. If you set this variable, you must also add the location of the shared libraries to the platform-specific library path environment variable (such as `LD_LIBRARY_PATH).`

Related Topics

GDM Commands

Generic Design Management (GDM) SKILL Functions

Common Arguments for GDM Commands

All `gdm` commands, except `gdmimport`, support the following arguments:

- `-lib` and `-file`
- `-recurse`
- `-cdslib`
- `-xtra`
- `-help`

If you use a `gdm`*`xxx`* command that the specific design management wrapping does not support, the system generates an error message. All `gdm`*`xxx`* commands have a nonzero exit status if the command fails or if any file operated on generates an error.

The -lib and -file Arguments

Specify items in libraries by using the argument `-lib` followed by a library name, using the following syntax:

```
lib.cell:view/file
```

Library, cell, and view names in this syntax are always in the library namespace (LibraryNT or LibraryUnix), which means that cell and view names appear just as an `ls` command shows them. The `-lib` argument need appear only once to determine how the system parses multiple specifications.

For example, everything in the cell `MUX2` in library `gates` is identified as:

```
-lib gates.MUX2
```

The Verilog source file in the `hdl` view of cell `cpu` in the `system` library is identified as:

```
-lib system.cpu:hdl/verilog.v
```

When you use the library syntax, all items appropriate to the level being used must be present. To specify a file in a view, you must specify all four items `lib.cell:view/file`. If a file is named, the forward slash (/) must be present. The remaining punctuation is optional if there is no name following these items. The following example specifies a cell-level file, `delay.tla` in cell `aoi3`, and a library level-file, `template.usr`.

```
-lib asic.aoi3/delay.tla asic/template.usr
```

You can give file specifications, which are not library references, with no arguments. If there is a need to specify both files and library references to a single command, use `-file` to change back to file specifications from library specifications. The following example refers to the entire contents of two cells, `mpeg` and `frame`, in library `video`, and the nonlibrary file `synth.cmd`, where `-lib` applies to all the arguments after the `-lib` argument until any following the `-file` argument. If `-file` is present, it applies to all the arguments until any following the `-lib` argument. No other arguments are positional. They have the same effect no matter where they appear in the command line.

```
% gdmcmd -lib 'video.mp*g'  video.frame  'video/index.htm*' -file ../synth.*
```

For GDM (Unix shell) commands, you can use wildcard characters, `*` and `?`, and letter range characters, `[ ]`, while specifying library, cell, view, and file names.

- The `*` character matches zero or more characters of the library, cell, or cellview name. For example, if you specify `-lib 'S7E100A_*.*:s*'`, it extracts all the elements where the library name starts with `S7E100A_`, such as `S7E100A_XU`, and the view name starts with `s`, such as `schematic` or `symbol`.

- The `?` character matches one character in the library, cell, or cellview name. For example, if you specify `-lib 'S7E100?.Cell??:layout'`, it extracts all the elements where the library name starts with `S7E100` and matches one character following it, such as `S7E100X` or `S7E100Y`; and extracts all the elements where the cell name starts with `Cell` and matches two characters following it, such as `Cell01` or `Cell02`, and the view name is `layout`.

- The letter range `[a-z]` matches one character in the given range in the library, cell, or cellview name. For example, if you specify `-lib 'S7E100[A-C].*:symbol'`, it extracts all the elements where the library names are `S7E100A`, `S7E100B`, and `S7E100C`, and the view name is `symbol`.

Important Points to Note

- The wildcard characters do not work when specified for the filenames starting with a dot (`.`). For example, if you specify `-lib 'lib/*'`, it will not extract the elements, such as `lib/.oalib`. In such cases, you need to specify the pattern as `-lib 'lib/.*'`.

- When you run the `gdmDelete` command, a prompt is displayed for confirmation before deleting the library elements only if wildcard characters are specified in the `-lib` argument.

- When using wildcard characters for the values specified for the `-lib` argument, ensure that those values are enclosed within single quotes. However, do not enclose values with single quotes when using the `-file` argument because the operating system shell needs to process those wildcard characters.

The -recurse Argument

If a nonlibrary specification is a directory name, by default it refers only to the files immediately below that directory. If the argument `-recurse` is present anywhere on the command line, it refers to the entire tree of files under that directory.

The -cdslib Argument

This argument specifies the library definition file to be used for mapping library names to library directories. If this argument is not specified, `gdm` looks for `cds.lib` files.

The -xtra Argument

GDM commands that run design management–specific commands take an argument of `-xtra`. This enables the GDM command to take additional arguments that are specific to the design management command so that it can pass them to the design management command.

The -help Argument

This argument prints out a brief message that describes the available arguments.

Related Topics

GDM Commands

Common Arguments for DM Commands

GDM commands call the design management–specific commands that constitute the wrapping of a design management system. For example, `gdmexport` calls *`xxxexport`*, where *`xxxexport`* is the wrap around the design management system export command. These design management-specific wraps must accept the following arguments, which are passed to them by the corresponding GDM commands:

Argument	Description
`-gdm`	Forces the command to follow GDM requirements.
`-u` *`file`*	Precedes files in the file list that are not managed but exist in the file system.
`-setstart` *`file1 file2`* `... -setend`	
	`-setstart` marks the beginning of a list of files that are to be treated as a set. `-setend` marks the end of the set.
`-xtra` *`arguments`*	Precedes arguments specific to the design management system.

Related Topics

Common Arguments for GDM Commands

GDM Commands

Designers can use Generic Design Management (GDM) shell commands to perform operations without using a graphical user interface. These commands are as follows:

- gdmcancel
- gdmci
- gdmco
- gdmdelete
- gdmexport
- gdmhistory
- gdmimport
- gdmsetdefver
- gdmsetname
- gdmstatus
- gdmsubmit
- gdmupdate

If you use a `gdm` command that the specific design management wrapping does not support, the system generates an error message. All `gdm` commands have a nonzero exit status if the command fails or if any file operated on generates an error.

Case Sensitivity

All arguments are case insensitive. Therefore, `-lib`, `-Lib`, and `-LIB` are equal. The variables that follow the arguments are case sensitive; for example, `gdmci lib ABCLibrary` is not the same as `gdmci -lib abcLibrary`.

Related Topics

Generic Design Management (GDM) SKILL Functions

gdmcancel

```
gdmcancel [-cdslib file] [-recurse] [-xtra str] [-optimize] [-help]
     [-lib lib.cell:view/file] [-file file] ...
```

Description

Cancels the checked-out status of files in the workarea. Co-managed files are always canceled as a group. Co-managed set behavior applies only to views when the view is specified as a library entry, such as `-lib lib.cell:view.`

Arguments

`-cdslib` *`file`*

Specifies the library definition file to be used for mapping library names to library directories.

`-recurse`

If a nonlibrary specification is a directory name, by default it refers only to the files immediately below that directory. If the argument `-recurse` is present anywhere on the command line, it refers to the entire tree of files under that directory.

`-xtra` *`str`*

Allows additional arguments, specific to the design management cancel command, to be passed through `gdmcancel` to that command.

`-optimize`

Skips some calls to the design management system to improve performance. Specifically, when you cancel a checkout, GDM first obtains the status of the files from the design managment system and removes any files that are not checked out from the file list. With the `-optimize` argument, GDM will skip the extra call to get status and just provide the list of files to the DM system. The DM system will then issue an error if a file has not been checked out.

`-help`

Displays information about this command and its arguments.

`-lib` *`lib.cell`*`:`*`view`*`/`*`file`*

Library elements for which the cancel checkout task needs to be performed. Names listed without `-lib` or `-file` are treated as `-file` arguments.

`-file` *`file`*

Files and directories (nonlibrary elements) for which to cancel checkout. Names listed without `-lib` or `-file` are treated as `-file` arguments.

Related Topics

GDM Commands

Generic Design Management (GDM) SKILL Functions

gdmci

```
gdmci [-cdslib file] [-recurse] [-xtra str] [-initial] [-description des_str]
     [-dfile file] [-optimize] [-help] [-lib lib.cell:view/file] [-file file]
     ...
```

Description

Checks in specified files and registers files that were previously unmanaged. Gives checked-out and previously unmanaged files to the repository so the files can be shared. Co-managed files in a view are always checked in as a group. Co-managed set behavior applies only when directories or files are specified as library elements; that is, with the `-lib` argument.

When you specify a library or cell, all the managed files in the library or cell are checked in, but only the co-managed files in the views are checked in.

Arguments

`-cdslib` *`file`*

Specifies the library definition file to be used for mapping library names to library directories.

`-recurse`

If a nonlibrary specification is a directory name, by default it refers only to the files immediately below that directory. If the argument `-recurse` is present anywhere on the command line, it refers to the entire tree of files under that directory.

`-xtra` *`str`*

Allows additional arguments, specific to the design management checkin command, to be passed through `gdmci` to that command.

`-initial`

Checks in all specified files and registers files that were previously unmanaged. If it is not used, new or unmanaged files that are members of a registered co-managed set will be checked in. All arguments are case insensitive and can be shortened to any abbreviation that is unique across all `gdm` command arguments. Hence, `-initial` can be specified as `-Ini`.

`-description` *`des_str`*

Cannot be used with `-dfile.` The string `"str"` is used as the description for the `checkin` operation.

`-dfile` *`file`*

Cannot be used with `-description`. Use `-dfile` to enter a multiline description. The contents of the file named are used as the description for the `checkin` operation.

`-optimize`

Skips some calls to the design management system to improve performance. Specifically, when you check in library elements or files, GDM first obtains the status of the files from the design managment system and removes any files that have already been checked in from the file list. With the `-optimize` argument, GDM will skip the extra call to get status and just provide the list of files to the DM system. The DM system will then issue an error if a file has already been checked in.

`-help`

Displays information about this command and its arguments.

`-lib` *`lib.cell:view/file`*

Library elements to check in. Names listed without `-lib` or `-file` are treated as `-file` arguments.

`-file` *`file`*

Files and directories (nonlibrary elements) to check in. Names listed without `-lib` or `-file` are treated as `-file` arguments.

Related Topics

GDM Commands

Generic Design Management (GDM) SKILL Functions

gdmco

```
gdmco [-cdslib file] [-recurse] [-xtra str] [-version version] [-name <tagname>]
    [-optimize] [-help] [-lib lib.cell:view/file] [-file file] ...
```

Description

Checks out specified files. Makes files in the workarea available for editing. Co-managed files are checked out in the same grouping in which they were checked in. Co-managed set behavior applies only when directories or files are specified as library elements; that is, with the `-lib` argument.

Arguments

`-cdslib` *`file`*
Specifies the library definition file to be used for mapping library names to library directories.

`-recurse`
If a nonlibrary specification is a directory name, by default it refers only to the files immediately below that directory. If the argument `-recurse` is present anywhere on the command line, it refers to the entire tree of files under that directory.

`-xtra` *`str`*
Allows additional arguments, specific to the design management checkout command, to be passed through `gdmco` to that command.

`-version` *`version`*
Value is a specific design management string representing the version from which this file is derived.

`-name` *`<tagname>`*
Allows you to check out the file from the *`version`* with the given tagname.

`-optimize`
Skips some calls to the design management system to improve performance. Specifically, when you check out library elements or files, GDM first asks the DM system to expand the library element or directory and return the status of the files to the server. With the `-optimize` argument, GDM expands the library element or directory on the client side and sends the list to the server, thus skipping the extra call to the DM system.

`-help`
Displays information about this command and its arguments.

`-lib` *`lib.cell:view/file`*

Library elements to check out. Names listed without `-lib` or `-file` are treated as `-file` arguments.

`-file` *`file`*

Files and directories (nonlibrary elements) to check out. Names listed without `-lib` or `-file` are treated as `-file` arguments.

Related Topics

GDM Commands

Generic Design Management (GDM) SKILL Functions

gdmdelete

```
gdmdelete [-cdslib file] [-xtra str] [-local] [-help][-recurse]
     [-lib lib.cell:view/file] [-file file] ...
```

Description

Deletes files both from the work area and from the default configuration. If deleted from the default configuration, `gdmupdate` does not place the file in your workarea. The file is hidden from future use.

Arguments

`-cdslib` *`file`*
Specifies the library definition file to be used for mapping library names to library directories.

`-xtra` *`str`*
Allows additional arguments, specific to the design management delete command, to be passed through `gdmdelete` to that command.

`-local`
Deletes files from your work area. Files in the repository are not deleted.

`-help`
Displays information about this command and its arguments.

`-recurse`
If a nonlibrary specification is a directory name, by default it refers only to the files immediately below that directory. If the argument `-recurse` is present anywhere on the command line, it refers to the entire tree of files under that directory.

`-lib` *`lib.cell:view/file`*
Library elements to delete. Names listed without `-lib` or `-file` are treated as `-file` arguments.

`-file` *`file`*
Files and directories (nonlibrary elements) to delete. Names listed without `-lib` or `-file` are treated as `-file` arguments.

Related Topics

GDM Commands

Generic Design Management (GDM) SKILL Functions

gdmexport

```
gdmexport [-cdslib file] [-recurse] [-xtra str]
     [-lib lib.cell:view/file | -file file] ... -destination path
     [-version identifier] [-complete] [-exportpaths] [-help]
```

Description

Exports files from the design management system data repository to the destination you specify. The destination can be a directory or a file; it can be a filename only if the file already exists and if you are exporting only one file.

When you export a library or cell directory, all the managed files in the library or cell are exported. When you export a view directory, only the co-managed files in the view are exported. Co-managed files are exported in the same grouping in which they were checked in. To export all the managed files in a view directory, use the `-complete` argument.

Note: When you a export a directory that is not a Cadence library element, only the files in the directory are exported. Files in subdirectories are not exported unless you use the `-recurse` argument.

The `gdmexport` command does not affect the data in the design management system.

The `gdmexport` command calls the `*xxx*export` command, which implements your design management system's export of data. The `*xxx*export` command should take the following arguments:

```
xxxexport -destination path [-version identifier] [-exportpaths]
```

Arguments

`-cdslib *file*`

The library definition file to be used for mapping library names to library directories.

`-recurse`

Exports the entire tree of files under the source directory. The files are copied to the destination directory. The source directory tree structure is not created in the destination directory unless you also specify the `-exportpaths` argument. If you do not use the `-recurse` argument, only the files in the top directory are exported; files in subdirectories are not exported.

When you export a Cadence library element (such as a library or cell), the entire tree of files under the directory is exported, even if you do not use the `-recurse` argument

`-xtra` *`str`*

Allows additional arguments, specific to the *`xxx`*`export` command, to be passed through `gdmexport` to that command.

`-lib` *`lib.cell:view/file`* `|` `-file` *`file`*

`-lib:` Library elements to export. All names that follow `-lib` and are before the next `-`*`argument`* are assumed to be library elements.

`-file`: Files and directories (nonlibrary elements) to export. All names that follow `-file` and are before the next `-`*`argument`* are assumed to be files.

All the files in the directory you specify are exported. Files in subdirectories are not exported unless you specify the `-recurse` option. The source directory tree structure is not created in the destination directory unless you also specify the `-exportpaths` argument.

`-destination` *`path`*

The location to which you want to export the files. You must specify this argument. If the path you specify does not exist, GDM creates the missing directories. You can specify a directory or a file as the destination. The destination can be a file only if it already exists and if you are exporting only one file. If the destination is an existing file, `gdmexport` overwrites its contents.

`-version` *`identifier`*

A string representing the version of the files to be exported. The value of `version` depends on the design management system you are using. For example, if you use Team Design Manager, the value of `version` would be one of the following: a version number, the string `current`, or the string `local`.

If a file does not have the version you specify, it is not exported.

`-complete`

Specifies that all managed files in view directories should be exported, instead of only the co-managed set of files.

`-exportpaths`

Creates the directory tree structure of the source directory in the destination directory. If you do not use `-exportpaths`, all the files are copied in the destination directory but subdirectories are not created. See also the descriptions for `-recurse` and

`-help`

Displays information about this command and its arguments.

Examples

```
gdmexport -lib LIB1.cell1:schematic -destination /tmp/tmplib/cell1/schematic
```

Exports the current version of the schematic to the directory `/tmp/tmplib/cell1/schematic`.

```
gdmexport -file myfile -version 3 -destination /tmp/
```

Exports version 3 of `myfile` to `/tmp/`.

```
gdmexport -lib LIB1.cell1 -version 7 -destination /tmp/tmplib/
           -exportpaths -recurse
```

Exports all the files in `cell1`, including the files in subdirectories under `cell1`, that have a version 7.

Related Topics

Common Arguments for DM Commands.

The -lib and -file Arguments

GDM Commands

Generic Design Management (GDM) SKILL Functions

gdmhistory

```
gdmhistory [-cdslib <filename>] [-lib lib.cell:view/file] [-file <file>] [-xtra
     <str>] [-full] [-author] [-size] [-last nn|-all] [-header] [-status]
     [-parent] [-long] [-labels]
```

Description

Returns information about the version history of a file. The library and file arguments must specify only one file.

Arguments

`-cdslib` *`<filename>`* Specifies the library definition file to be used for mapping library names to library directories.

`-lib` *`lib.cell:view/file`*

Library elements for which to get their histories. Names listed without `-lib` or `-file` are treated as `-file` arguments.

`-file` *`<file>`* Files and directories (nonlibrary elements) for which to get their histories. Names listed without `-lib` or `-file` are treated as `-file` arguments.

`-xtra` *`<str>`* Allows additional arguments, specific to the design management history command, to be passed through `gdmhistory` to that command.

`-full` Full description is printed on separate lines from the other information.

`-author` Name of the author of the file.

`-size` Size of the file.

`-last` *`nn|-all`* Takes a numeric option. Prints each version number and date and the first several characters of the history up to the given number of versions.

`-header` Header of the history output.

`-status` Status of the file.

`-parent` Returns the version from where the current version is derived from. For example, if the current version is 1.5 and it is derived from 1.4, then `-parent` will display the version as 1.4 in the `gdmhistory` command line.

The `-parent` argument is always *on* by default, so that the returned data of *version*, *date*, and *description* will be displayed.

`-long` Ignores any given options, but will return all the options on the list.

`-labels` Will return any labels/names attached to the version by calling the `gdmsetname` API.

Examples

```
gdmhistory -file my_lib/my_cell/schematic/sch.oa -author -header
=> ${CWD}/my_lib/my_cell/schematic/sch.oa
Derived       Version    From Date                        Author      Desc
1.2           1.1        Mon Nov 11 14:30:32 GMT 2019 userA  [NO_COMMENT]
```

The output shows information about version `1.2` that was derived from version `1.1` (for example, the checked out version) created by the user `userA` on the listed date. No check in comment is included.

This information is according to the expected output of the example. However, the availability of the information is dependent on a DM system's support for GDM features.

When the `-file` *`filePath`* option is used with `gdmhistory`, the data is examined as a relatively located file, and there is no name mapping. This distinction is important because when you specify a file inside a library, the mapped names for the library, cell, or view required by the `-lib` option might differ from the non-mapped names required by the `-file` option.

Related Topics

gdmsetname

GDM Commands

Generic Design Management (GDM) SKILL Functions

gdmimport

```
gdmimport srcLib targetPath [-help]
```

Description

Copies a Cadence library (*srcLib*) into the current workarea (*targetPath*).

Arguments

srcLib	The source library.
targetPath	The workarea path to which you want to copy the library.
-help	Displays information about this command and its arguments.

Related Topics

GDM Commands

Generic Design Management (GDM) SKILL Functions

gdmregprint

```
gdmregprint [-help] [-version] [-indexName index_name]
       [-dataFormat] [-library] [-cell]
```

Description

Examines the registry entries that are design management related and visible to the Cadence tool.

Note: The `gdmregprint` command must run from the same directory where the Cadence tool is executed because files in the current working directory can affect the registry entries.

When no arguments are specified, the help information is displayed.

Arguments

`-help` Displays information about this command and its arguments.

`-version` Displays version information for the `gdmregprint` command.

`-indexName` *`index_name`* Contains statements to use instead of the `*.reg` files content, which is available in the same directory. If the index file is not found, content from the `*.reg` files is read. To create an index file, add the `.idx` suffix to the specified name.

`-dataFormat` Displays the pattern property and property information for the `dataFormat` entries that contain one or both of the following properties:

- `Co_Managed`
- `Also_Managed`

`-library` Displays the `Also_Managed` property information for the library entry. If the property is not defined, library information is not displayed.

`-cell` Displays the `Also_Managed` property information for the cell entry. If the property is not defined, cell information is not displayed.

Related Topics

GDM Commands

Generic Design Management (GDM) SKILL Functions

gdmsetdefver

```
gdmsetdefver -version version [-cdslib file] [-xtra str] [-name tag] [-help]
     [-lib lib.cell:view/file] [-file file] ...
```

Description

Associates the given version to the specified set of files currently in the workarea. Action is DM system specific. For TDM, this command sets the specified version as the default version.

Arguments

`-version` *`version`* — Gives the version of the file. Value is a specific design management version identifier. The file list must be a single file or a single co-managed set.

`-cdslib` *`file`* — Specifies the library definition file to be used for mapping library names to library directories.

`-xtra` *`str`* — Allows additional arguments, specific to the design management command, to be passed through `gdmsetdefver` to that command.

`-name` *`tag`* — Corresponds to a tag specification in RCS wraps, or a release name in TDM.

`-help` — Displays information about this command and its arguments.

`-lib` *`lib.cell:view/file`*

Library elements for which to set the default version. Names listed without `-lib` or `-file` are treated as `-file` arguments.

`-file` *`file`* — Files and directories (nonlibrary elements) for which to set the default version. Names listed without `-lib` or `-file` are treated as `-file` arguments.

Related Topics

GDM Commands

Generic Design Management (GDM) SKILL Functions

gdmsetname

```
gdmsetname -name tag [-cdslib file] [-xtra str] [-recurse] [-version version]
     [-help] [-lib lib.cell:view/file] [-file file] ...
```

Description

Associates the given name to the specified set of files currently in the workarea.

Arguments

`-name` *`tag`* — Corresponds to a tag specification in RCS wraps. DM-specific symbolic name string.

`-cdslib` *`file`* — Specifies the library definition file to be used for mapping library names to library directories.

`-xtra` *`str`* — Allows additional arguments, specific to the design management command, to be passed through `gdmsetname` to that command.

`-recurse` — If a nonlibrary specification is a directory name, by default it refers only to the files immediately below that directory. If the argument `-recurse` is present anywhere on the command line, it refers to the entire tree of files under that directory.

`-version` *`version`* — Value is a specific design management version identifier. For some design management systems, a missing file list is legal if `-name` is used. The file list must be a single file or a single co-managed set.

`-help` — Displays information about this command and its arguments.

`-lib` *`lib.cell:view/file`*

Library elements for which to set name. Names listed without `-lib` or `-file` are treated as `-file` arguments.

`-file` *`file`* — Files and directories (nonlibrary elements) for which to set `name`. Names listed without `-lib` or `-file` are treated as `-file` arguments.

Related Topics

GDM Commands

Generic Design Management (GDM) SKILL Functions

gdmstatus

```
gdmstatus [-cdslib file] [-lib lib.cell:view/file] [-file file] [-xtra str]
     [-workarea] [-repository] [-civersion] [-coversion] [-updateversion]
     [-needupdate] [-cachedok] [-status] [-header] [-absolute] [-modified]
     [-where] [-recurse] [-help]
```

Description

Returns the design management status of files. If no file is present, the command operates on the current directory with `-recurse`.

Arguments

`-cdslib` *`file`* Specifies the library definition file to be used for mapping library names to library directories.

`-lib` *`lib.cell:view/file`*

Specifies the library elements. Names listed without `-lib` or `-file` are treated as `-file` arguments.

`-file` *`file`* Specifies the files and directories (non-library elements) that need have their check out canceled. Names listed without `-lib` or `-file` are treated as `-file` arguments.

`-xtra` *`str`* Allows additional arguments, specific to the design management status command, to be passed through `gdmstatus` to that command.

`-workarea` Shows the specifications of the files present in the specified location.

`-repository` Shows information, such as the files to be managed, where they are stored, and which views are accessible.

`-civersion` Shows the `checkin` version.

`-coversion` Shows the `checkout` version.

`-updateversion` Shows the update version.

`-needupdate` Indicates if the file needs to be updated. Returns one of the following:

- `outOfDate`: The file is out-of-date and an update is required in the cellview.
- `upToDate`: The file is up-to-date and no update is required in the cellview.
- <blank space>: DM does not support this feature.

`-cachedok` Indicates to the DM to accept the cached status value information when querying the files. This option is recommended only when the speed for querying the files has the highest priority.

`-status` Shows the status.

`-header` Shows the header.

`-absolute` Shows the absolute name from the slash (/). Otherwise, the output is relative to the current directory or as a library specification.

`-modified` Displays the filename followed by an asterisk (*) if it is modified, or the filename followed by a question mark (?) if its modification status is unknown.

`-where` Asks GDM for both the `COLOC` and `COWHO` status. If `COLOC` is returned, it is displayed; otherwise, the `COWHO` string is displayed.

`-recurse` Refers only to the files immediately below the directory, if the non-library specification is a directory name. If the argument `-recurse` is present anywhere on the command line, it refers to the entire tree of files under that directory.

`-help` Displays information about this command and its arguments.

Example

Returns `outOfDate`. The `master.tag` file needs to be updated in the <`rodTrLib | Design | layout`> cellview.

```
gdmstatus -lib rodTrLib.Design:layout/master.tag –needupdate
=> outOfDate
```

Related Topics

GDM Commands

Generic Design Management (GDM) SKILL Functions

gdmsubmit

```
gdmsubmit [-cdslib file] [-recurse] [-xtra str] [-description des_str]
     [-dfile file] [-name name] [-includeco] [-initial] [-help]
     [-lib lib.cell:view/file] [-file file] ...
```

Description

Submits files for a release; DM specific.

Arguments

`-cdslib` *`file`*
Specifies the library definition file to be used for mapping library names to library directories.

`-recurse`
If a nonlibrary specification is a directory name, by default it refers only to the files immediately below that directory. If the argument `-recurse` is present anywhere on the command line, it refers to the entire tree of files under that directory.

`-xtra` *`str`*
Allows additional arguments, specific to the design management submit command, to be passed through `gdmsubmit` to that command.

`-description` *`des_str`*

Cannot be used with `-dfile`. The string `"str"` is used as the description for the `submit` operation.

`-dfile` *`file`*
Cannot be used with `-description`. Use `-dfile` to enter a multiline description. The contents of the file named are used as the description for the `submit` operation. Co-managed files in a view are always submitted as a group. Co-managed set behavior applies only to views when the view is specified as a library entry, such as `-lib lib.cell:view.`

`-name` *`name`*
Integration request name for TDM.

`-includeco`
Includes checked out files in the submission.

`-help`
Displays information about the command and its arguments.

`-lib`
Library elements for which to submit. Names listed without `-lib` or `-file` are treated as `-file` arguments.

`-file` Files and directories (nonlibrary elements) for which to submit. Names listed without `-lib` or `-file` are treated as `-file` arguments.

Related Topics

GDM Commands

Generic Design Management (GDM) SKILL Functions

gdmupdate

```
gdmupdate [-cdslib file] [-recurse] [-xtra str] -name relName [-byname]
     [-version fileversion] [-force] [-help] [-lib lib.cell:view/file]
     [-file file] ...
```

Description

Makes files in the workarea available for reading. Updates co-managed files in the same grouping they were checked in with. Cancels the checked-out status of files in the workarea only if `-force` is used.

Arguments

`-cdslib` *`file`*
: Specifies the library definition file to be used for mapping library names to library directories.

`-recurse`
: If no file list is present, the command operates on the current directory with `-recurse` unless `-byname` is used.

`-xtra` *`str`*
: Allows additional arguments, specific to the design management update command, to be passed through `gdmupdate` to that command.

`-name` *`relName`*
: Corresponds to a tag specification in RCS wraps or a release name in TDM. This is a required field. Do not use `-name` if the name request is non-NULL but an empty string.

`-byname`
: Can only be used with `-name`, and only with some design management systems. It specifies use of the complete set of files indicated by `-name`.

`-version` *`fileversion`*
: Value is a specific design management version identifier. For some design management identifier systems, a missing file list is legal if `-name` is used. File list must only be a single file or view. A single file can be specified with a path, or as an explicit file in a library, cell, or view.

`-force`

Performs the update even if it overwrites modified files. GDM deletes modified files and makes sure that none are checked out. Receives the `-force` argument because TDM uses it to affect version selection for the update. Most design management wraps ignore it. Tells GDM to update files even if modifications are overwritten by doing so. If `-force` is not present and any modified files are present, the command gives an error.

`-help`

Displays information about this command and its arguments.

`-lib` *`lib.cell:view/file`*

Library elements for which to update. Names listed without `-lib` or `-file` are treated as `-file` arguments.

`-file` *`file`*

Files and directories (nonlibrary elements) for which to update. Names listed without `-lib` or `-file` are treated as `-file` arguments.

Related Topics

GDM Commands

Generic Design Management (GDM) SKILL Functions

9

cdsCopy

You can use the cdsCopy system to perform the following:

- Merge libraries to make a new release library
- Back up a library by copying it to a new location

  All the co-managed cellview files, except derived cellview files, are copied.
- Rename a library, updating references to it from a number of “client” libraries
- Move a library by copying and deleting the library without updating cross-references

The cdsCopy system can be used in either batch or interactive mode.

In batch mode, using SKILL access or from a user interface, you can run `cdsCopy` on an object for the system to perform the following tasks:

- Performs the expansion, depending on the options provided
- Copy the data
- Display all cross-reference updaters that are registered

You can run `cdsCopy` in batch mode from a `cdsCopyShell`, Virtuoso Studio design environment Command Interpreter Window (CIW).

In interactive mode, you can do one of the following:

- Set up the copy and then perform a copy operation without further interaction.
- Perform the expansion first, making any desired changes to the expanded list, and then do a copy and update.

You can run `cdsCopy` functions in interactive mode from the Command Interpreter Window or `cdsCopyShell`.

Guidelines for Using cdsCopy

Before you copy or rename a library, you should stabilize and verify your design. Similarly, you need to perform certain tasks after you copy or rename the library. The following process shows you how to use cdsCopy with a design management system:

1. Stabilize and verify your design.
2. Check in all the files in the design so that the current state of the design files is saved.
3. Copy or rename the design with cdsCopy. cdsCopy checks out managed files if it needs to but does not check them back in.
4. Verify the copied or renamed design.
5. Check in the files that cdsCopy checked out. The cdsCopy system does not check in the files; you have the option of checking in the files if you want to keep the changes made by cdsCopy cross-reference updaters or canceling the checkout if you do not want the changes.

Related Topics

cdsCopy SKILL Functions

Starting the cdsCopyShell

cdsCopy Functions

The copy process is divided into the following small steps for quick customization:

- Expansion
- Filtering
- Copy
- Cross-reference updating

There are three types of expansion—directory expansion, design expansion, and configuration expansion. Directory expansion expands a directory such as a library directory into all the files in that directory and its subdirectories. Design expansion follows the hierarchical structure of the design (even into other libraries) and usually does not include all the files in any library. Configuration expansion is similar to design expansion except that a configuration file is used to select the items to expand (such as views).

The next step in the process is filtering. There is some built-in filtering in the expansion step that is controlled by options to the expansion functions. However, it is also possible to do customized additional filtering after expansion. For example, you could add documentation files that your design methodology requires.

Once a list of files to copy has been created, copy can proceed. There is only one option for copy—whether to overwrite existing files. After the files are copied, the update process is initiated.

Cross-reference updating is needed whenever design files contain references to other design files that were moved or copied. The cross-reference update process splits the list of files that need to be updated into lists that contain files of the same data registry type, such as CDBA or Verilog. Each list is then handed to the updater program that is registered for that type in the data registry. Custom updaters are registered in the data registry just like Cadence updaters.

Related Topics

cdsCopy SKILL Functions

Starting the cdsCopyShell

Customizing cdsCopy

The cdsCopy system provides two levels of customization:

- Simple customization is controlled by a set of options and covers most needs.
- Full customization provides complete control over what gets copied and how cross-references are updated.

You implement copy customization by writing SKILL functions that run in Virtuoso applications or a cdsCopyShell and by providing custom cross-reference updaters for non-Cadence data types.

Simple Customization

The options for simple customization are:

Option	Description
Overwrite	Overwrites existing files.
Expansion Mode	Copies the files of a view. ■ Don’t Expand: No files are copied. ■ Expand Comanaged*:* Copies only the files that are part of the co-managed set of the view. ■ Expand All*:* Copies all the files in a view.
View Type List	Copies only views of this type. Ignored if Don’t Expand is selected.
View Name List	Copies only views with these names. Ignored if Don’t Expand is selected.
What to Cross-Reference Update	Updates the copied Data, destination Library, or a provided list of libraries.

The options used in simple customization are also used in full customization as arguments in cdsCopy SKILL functions.

Full Customization

Full customization includes copy customization, which lets you add or remove files from the list to copy, and cross-reference customization, which lets you update file types that Cadence does not support.

Copy Customization

To implement copy customization, you write functions that run in Virtuoso applications (in a Command Interpreter Window or from a customized user interface) or a cdsCopyShell. `cdsCopyShel`l is a standalone application with a SKILL interpreter and cdsCopy SKILL functions.

Your custom SKILL function does the following:

1. (Optional) Displays a SKILL graphical user interface form to gather information.
2. Calls one of the `cdsCopy` expansion functions to get the starting list of items to copy.
3. Adds or deletes items from this list.
4. Calls the `ccpCopy` function with the *Don’t Expand* option to copy the data.

Cross-Reference Update

Many design files contain references to other design files. In many cases, when you copy design files, you would like the internal references to point to the new copies of the files rather than to the old files. The cdsCopy cross-reference updating system, which consists of cross-reference updater programs and data registry entries, does this.

Related Topics

View Files and a Co-Managed Set

cdsCopy SKILL Functions

Cross-Reference Updater System

Starting the cdsCopyShell

`cdsCopyShell` is a standalone application with a SKILL interpreter and `cdsCopy` SKILL functions. You can run your copy programs in a `cdsCopyShell`.

To start a `cdsCopyShell`,

➤ In a UNIX shell, type

```
cdsCopyShell
```

This gives you a SKILL prompt, where you can enter SKILL input.

To exit the SKILL prompt, type `exit`.

To start a copy program directly,

➤ In a UNIX shell, type

```
cdsCopyShell myprogram.il
```

where *`myprogram`*`.il` is your SKILL program.

Related Topics

Customizing cdsCopy

cdsCopy SKILL Functions

cdsCopy Functions

Cross-Reference Updater System

Design files often contain references to other design files. Typically, when you copy design files, you would like internal references to point to the new copies of the files rather than to the old files. The cdsCopy cross-reference updating system, which consists of cross-reference updater programs and entries in data registry files, does this.

Cadence provides cross-reference updaters for many types of design files. Currently, these include CDBA, category (both *`Cat` and `.ctf` files), `prop.xx`, and configuration (`expand.cfg`) files. You can add custom cross-reference updaters for other types of files. Entries in the data registry determine which updater is run.

To add a custom cross-reference updater:

1. Write a program to do cross-reference updating.
2. Associate the cross-reference updater with the data type in the data registry.

Guidelines for Creating a Cross-Reference Updater

Follow the guidelines described in this section while creating cross-reference updaters.

All cdsCopy cross-reference updater programs provided by Cadence accept the command line arguments described below. Any custom cross-reference updater program you write must also accept these arguments.

`-fromToList` *`file`*

File, generated by the cdsCopy system, containing a list of all files copied in the copy or rename operation. This file has copied/renamed `from` and `to` file pairs, one per line, with library, cell, and view names written in the LibraryNT namespace. Each word in these pairs can be one of the following:

`Lib/File`

`Lib.Cell/File`

`Lib.Cell:View/File`

`Lib.Cell:View/Dir/File`

For example:

```
mylib/myfile newlib/myfile
lib1.mycell:myview/mydir/myfile lib2.mycell:myview/
mydir/myfile
```

`-updateFileList` *`file`*

List of files in one of the above forms, one per line. If the file is the master for a cellview, `master` is the second word on the line. The cdsCopy system ensures that this list contains only files that the updater can handle, based on information in the data registry.

For example:

```
mylib.mycell/prop.xx
```

or

```
lsttl.ls04:logic/vhdl.vhd master
```

`-cdslib` *`file`*

The `cds.lib` file that the updater should use to interpret the library, cell, and view names.

`-reason` *`operation`* One of the following strings, which indicates the reason for running the cross-reference updater:

`"copy"`

`"rename"`

`"renameRef"`

`[-renameRefOldToNew` *`fromLib toLib`*`]`

The old and new library names, in the LibraryNT namespace, which are used by the cross-reference updater for a rename reference operation. This argument is optional and is used only during a rename reference operation.

`[-renameRefAllow` *`option`*`]`

One of the following strings, which specifies the references to change during a rename reference operation:

- `"all"` Changes all occurrences.
- `"to"` Changes references only if the `"to"` library exists.
- `"from"` Changes references only if the `"from"` library exists.
- `"both"` Changes references only if both the `"from"` and `"to"` libraries exist.

This argument is optional and is used only during rename reference operation.

Basic Updater Program Algorithm

To update the program algorithm,

1. Read in a file from `updateFileList`.
2. Find all external references.
3. For each external reference, see if it is one of the files in the `from` (word one) list in the `fromToList` file. If so, replace it with the `to` file (word two).
4. Save the file.

Properties of fromToList

The entries in the `fromToList` file can be applied in any sequence to the files listed in the `updateFileList`. This is because the `fromToList` does not have any order dependencies.

The `fromToList` has the following properties, which ensure that it is free of the ambiguities of an unordered list:

- A `to` entry never appears in any of the `from` entries. This means that you are never asked to update a file when the copy system has copied a file to one location and then copied it to another location.
- A `from` entry never appears more than once in the `from` entries. This means that you are never asked to update a file when the copy system has copied one file to two different places.
- A `to` entry never appears more than once in the `to` list. This means that two objects are never copied to the same place.

Error Messages

Your cross-reference updater should return error messages to cdsCopy by writing to `stderr` in a special format. Use the following format for error messages:

```
{#nn L msg}
```

nn The number of the file in the list of files to update that is handed to the cross-reference updater. The first file in the list is `1`. Use `0` for errors that are not related to a specific file to be updated.

L The message level. One of the following:

`I`	Information
`W`	Warning
`E`	Error
`F`	Fatal

msg

The message to be returned. If you need to include a `}` character in the message, enclose the message in double quotes (`"  "`). If you need to include a `"` character in the message, use two quotation marks (`""`) instead, and enclose the message in quotation marks.

Messages can be of any length and can have multiple lines.

For example:

```
{#1 I Updating file}
{#17 E File system full}
{#0 W "Cannot update ""mylogfile.log"""}
```

Related Topics

ccpRenameReferenceLib

Properties of fromToList

Adding a Cross-Reference Updater

After you create a cross-reference updater, you need to associate it with a data type in the data registry. The `updaterProg` property determines which cross-reference updater is run for a data type.

The following sections describe how to add cross-reference updaters for new data types as well as for data types that are already defined in the data registry.

Adding a Cross-Reference Updater for an Existing Data Type

To add a cross-reference updater for a data type that is already defined in the data registry, you either modify the original `DataFormat` definition for the data type or add a `+DataFormat` entry to your site or personal `data.reg` file.

While you are developing and testing your cross-reference updater, you might find it convenient to add it just to your site or personal `data.reg` file. When the cross-reference updater is ready, you can modify the original `DataFormat` definition.

To add a cross-reference updater to your site or personal `data.reg` file,

➤ Add the following to the `data.reg` file:

```
+DataFormat datatype{
    updaterProg = updater;
}
```

For example, if you have the following `DataFormat` entry for `myDataType` in a `.reg` file in the `/share/cdssetup/registry/data` directory:

```
DataFormat myDataType{
    Pattern = mydata.kanth;
    Preferred_Editor = mydata-Editor;
    dfII_ViewType = mydataType;
    Co_Managed = master.tag prop.xx pc.db;
}
```

and you want to add the cross-reference updater `mydataUpdater`, add the following to your site or personal `data.reg` file:

```
+DataFormat myDataType{
    updaterProg = mydataUpdater;
}
```

To modify the original data format definition in the data registry,

1. Add the following property to the `DataFormat` entry (which is in a `.reg` file in the `/share/cdssetup/registry/data` directory):

   ```
   updaterProg = updater;
   ```

   The following example shows a `DataFormat` definition that includes the `updaterProg` property.

   ```
   DataFormat myDataType{
       Pattern = mydata.kanth;
       updaterProg = mydataUpdater;
       Preferred_Editor = mydata-Editor;
       dfII_ViewType = mydataType;
       Co_Managed = master.tag prop.xx pc.db;
   }
   ```

2. Remove any `+DataFormat` entries for the cross-reference updater that you might have added to your site or personal `data.reg` files.

Adding a Cross-Reference Updater for a New Data Type

To add a cross-reference updater for a data type that is not defined in the data registry,

1. Add a new `DataFormat` entry to the `.reg` file in `share/cdssetup/registry/data` that contains your data formats. Or create a new *`datatype`*`.reg` file containing a `DataFormat` entry in the `share/cdssetup/registry/data` directory.

2. Include the following property in the `DataFormat` entry:

   ```
   updaterProg = updater;
   ```

   The `updaterProg` property determines which cross-reference updater is run for that data type.

For example, to add a cross-reference updater for the new data type `myDataType`, create a `myDataType.reg` file in the `share/cdssetup/registry/data/` directory with the following `DataFormat` entry:

```
DataFormat myDataType{ //Define new data type
    Pattern = mydata.kanth;
    updaterProg = mydataUpdater;
    Preferred_Editor = mydata-Editor;
    dfII_ViewType = mydataType;
    Co_Managed = master.tag prop.xx pc.db;
}
```

Related Topics

data.reg File Creation

Tool and Data Registry Information

Cadence Data Registry File: data.reg

cdsCopy SKILL Functions

10

Cadence Locking System

The Cadence® locking system (CLS) is the mechanism used by many Cadence applications to lock files.

CLS creates a lock file, called Lock-Stake file, when it locks a file for editing. The Lock-Stake file contains information about the application that has locked the file.

CLS allows applications to steal locks from active processes; however, few applications support this feature. CLS automatically recovers locks that are stranded if an application exits without releasing the locks.

In addition, CLS provides a user interface for administrative tasks such as releasing all locks in a directory hierarchy.

Applications on OpenAccess

Cadence applications on the OpenAccess 2.2 database use OpenAccess locking instead of CLS. OpenAccess locking is compatible with CLS—the lock file has the same format and file name extension as the file created by CLS, and the same rules are followed while getting locks.

Virtuoso applications on OpenAccess use both CLS and OpenAccess locking; they use OpenAccess locking to lock database files and CLS to lock other files such as `CDS.log`. Since the two systems are compatible and they lock different files, conflicts should not arise.

OpenAccess locking uses a different daemon, `oaFSLockD`, instead of the CLS boolean daemon `clsbd` to recover locks. `oaFSLockD` is run automatically whenever a library is opened. Virtuoso applications on OpenAccess start both daemons.

You can use the `clsAdminTool` to search for and release locks created by either system.

Note: A lock set by a host that runs only non-OpenAccess applications (which start only `clsbd`) cannot be recovered by a native OpenAccess application and vice versa. This is because each locking system needs to talk to its own boolean daemon to recover locks. In such a case, remove stranded locks manually or with `clsAdminTool`.

Installation

CLS is automatically installed when you install Cadence applications; you do not need to install it separately.

However, on Windows NT, you should manually set up the Boolean daemon (a component of CLS) as a service after you install Cadence applications.

You can also customize how the Boolean daemon is started. The daemon must run on every system that runs Cadence applications. You can set up your system to run the Boolean daemon every time the system is started. This procedure is optional, applications that use CLS start the Boolean daemon if it is not already running on the system.

Related Topics

Setting Up the Boolean Daemon on Windows NT

Adding the Boolean Daemon to System Startup Files

Boolean Daemon (clsbd)

The Boolean daemon, called `clsbd`, is a daemon that runs on all systems on which any Cadence application using Cadence® locking system (CLS) is running. The daemon responds to requests from remote systems to determine the status of processes that own Edit locks.

Note: There is a default timeout of 20 seconds in respect of `clsbd` connections. This is also the same time period used for Open Access connections.

When an application wants to lock a file and finds that it is already locked by a process on another system, CLS queries the Boolean daemon running on that system about the status of the process that locked the file. Whether the application can obtain the lock depends on the information CLS receives from the Boolean daemon.

- If the Boolean daemon indicates that the process that locked the file has exited, the application gets the lock. The Lock-Stake file (*`filename`*`.cdslck`) is overwritten with information about the new process.
- If the Boolean daemon indicates that the process that locked the file is still active, the application cannot get a lock on the file.
- If the application cannot connect to the Boolean daemon, the process that owns the lock is assumed to be active and the application cannot get a lock on the file.

If you do not want CLS to query Boolean daemons on remote hosts (you may not want to query remote hosts if you have firewall access issues, for example), set the following environment variable:

```
CLS_DONT_QUERY_REMOTE_HOSTS [listOfRemoteHosts]
```

where *`listOfRemoteHosts`* is an optional list of patterns (typically host or domain names), separated by blank spaces, that you do not want CLS to query. For example, if you specify

```
setenv CLS_DONT_QUERY_REMOTE_HOSTS cds00 cadence.com ns.com
```

CLS will not query hosts that match `cds00` or any hosts from the domains `cadence.com` or `ns.com`.

If you do not specify *`listOfRemoteHosts`*, CLS will not query any remote hosts.

When the `CLS_DONT_QUERY_REMOTE_HOSTS` environment variable is set, if CLS comes across a file locked by a remote host, it always assumes that the process that locked the file is still active. Therefore, CLS cannot get a new lock on the file. In such a case, if you want to lock the file, you need to remove the old lock with the CLS administrative tool.

Note: When an application wants to lock a file that is locked by a process on its own host, CLS determines the status of the process directly without sending a query to a Boolean daemon.

Setting Up the Boolean Daemon on Windows NT

After you install Cadence applications on Windows NT, you should manually set up the Boolean daemon as a service so that Cadence applications can start it.

Note: You must have administrator privileges for the following procedure.

To list the Boolean daemon as a service,

1. Log on to the Windows NT system as a user with administrator privileges.
2. From the *Start* menu, choose *Programs – Command Prompt*.

   The MS DOS Command Prompt window appears.
3. In the MS DOS Command Prompt window, type

   ```
   clsbd.exe -install
   ```

   The following message appears:

   ```
   CDS Boolean Daemon Installed.
   ```
4. Close the MS DOS Command Prompt window by typing `exit`.
5. From the *Start* menu, choose *Settings – Control Panel*.
6. In the Control Panel window, double-click *Services*.

   The Services dialog box appears.
7. Confirm that *CDS Boolean Daemon* is listed as a service.
8. In the Services dialog box, click *Close*.

Location of the Boolean Daemon

The Boolean daemon is automatically installed when you install Cadence applications.

The Boolean daemon program, `clsbd`, is located in the following directory:

```
application_install_dir/tools/bin/
```

where *`application_install_dir`* is the directory in which your Cadence application is installed.

Related Topics

CLS Administrative Tool

Starting the Boolean Daemon

The Boolean daemon must run on all systems that run Cadence applications. It is started in several ways.

- You can add the Boolean daemon to a system startup files so that the daemon is run automatically when your system is started.
- Cadence applications that use Cadence® locking system (CLS) start the daemon if it is not already running.
- You can start the Boolean daemon directly from the application installation hierarchy, if it is not already running.

Adding the Boolean Daemon to System Startup Files

To add the Boolean daemon to a system startup files, do one of the following:

- For SunOS 5.x, create a `/etc/rc2.d/S98clsbd` file that contains the following:

  ```
  #!/bin/sh
  # Start-up Cadence locking system boolean daemon (clsbd).
  #
  INSTALL_DIR=’hierarchyPath’
  if [ -x ${INSTALL_DIR}/tools/bin/clsbd ] ; then
      ${INSTALL_DIR}/tools/bin/clsbd
      echo “Starting Cadence locking system boolean daemon (clsbd).”
  else
      echo “Cadence locking system boolean daemon not started.”
  fi
  ```

  where *`hierarchyPath`* is the path to your Cadence installation directory.

- For IBM AIX, add the following to the `/etc/inittab` file:

  ```
  INSTALL_DIR=’hierarchyPath’
  if [ -x ${INSTALL_DIR}/tools/bin/clsbd ] ; then
      ${INSTALL_DIR}/tools/bin/clsbd
      echo “Starting Cadence locking system boolean daemon (clsbd).”
  else
      echo “Cadence locking system boolean daemon not started.”
  fi
  ```

  where *`hierarchyPath`* is the path to your Cadence installation directory.

- For HPUX, add the following to the `localrc` function definition to the `/etc/rc` file:

  ```
  localrc()
  {
      ...
      INSTALL_DIR=’hierarchyPath’
      if [ -x ${INSTALL_DIR}/tools/bin/clsbd ] ; then
          ${INSTALL_DIR}/tools/bin/clsbd
          echo “Starting Cadence locking system boolean daemon (clsbd).”
  ```

```
    else
        echo “Cadence locking system boolean daemon not started.”
    fi
}
```

where *hierarchyPath* is the path to your Cadence installation directory.

- For Linux, create a `/etc/rc.d/rc3.d/S98clsbd` file that contains the following:

```
#!/bin/sh
# Start-up Cadence locking system boolean daemon (clsbd).
#
INSTALL_DIR='hierarchyPath'
if [ -x ${INSTALL_DIR}/tools/bin/clsbd ] ; then
    ${INSTALL_DIR}/tools/bin/clsbd
    echo "Starting Cadence locking system boolean daemon (clsbd).
else
    echo "Cadence locking system boolean daemon not started."
fi
```

where *hierarchyPath* is the path to your Cadence installation directory.

- For Windows NT:

  a. Log in to your system as a user with administrator privileges.

  b. From the *Start* menu, choose *Settings – Control Panel*.

  c. In the *Control Panel* window, double-click *Services*.

  The Services dialog box appears.

  d. In the *Service* list, select *CDS Boolean Daemon*.

  e. Click *Startup*.

  The Service dialog box appears.

  f. In the *Startup Type* field, select *Automatic*.

  g. Click *OK*.

  h. Click *Close* to close the Services window.

Starting the CLS Boolean Daemon from the Installation Directory

To start the CLS Boolean daemon from the application installation directory,

➤ In a UNIX or MS DOS window, type

```
application_install_dir/tools/bin/clsbd
```

where *application_install_dir* is the directory in which your Cadence application is installed.

You can use the following command-line options when you start clsbd:

`-fg` Runs clsbd in the foreground, instead of in the background.

`-debug` Turns on clsbd debug messages that are written to the log file.

`-logfile` *fileName* Writes debug messages to the file you specify. The default log file is `/tmp/clsbd.`*userId.processID*.

`-portnum` *portNumber*

Specifies the port number on which to run clsbd. By default, clsbd uses port 16723 on every system on which it runs. You can use this option if another application is using port 16723 or if a clsbd is already running on that port and you want to run another clsbd.

If the `CLS_PORTNUM` environment variable is set, clsbd will use the port number that the variable is set to, regardless of whether you specify the `-portnum` option.

`-setuid` *login* Changes the user ID of the clsbd process to the login you specify, after clsbd is started. You may need to start clsbd with super-user privileges so that it can change the user ID. This option uses the UNIX `setuid` command.

You can also use this option with the `-portnum` option to run a non-root clsbd process on a privileged port.

The `-setuid` option is not available on Windows NT.

`-install` Installs the Boolean daemon as a service on Windows NT. This option is only available on Windows NT.

`-remove` Removes the Boolean daemon as a service from Windows NT. This option is only available on Windows NT.

`-version` Displays the version number of the Boolean daemon.

Related Topics

Setting Up the Boolean Daemon on Windows NT

Changing the CLS Boolean Daemon Port Number

Changing the CLS Boolean Daemon Port Number

If the default port number used by the Boolean daemon is consistently used by another application, you might have to change the Boolean daemon port number. This situation is uncommon.

`clsbd` uses port number 16723 on every system on which it runs. When `clsbd` is started, if it finds that this port number is already being used, it checks if another `clsbd` is running on the system. If another `clsbd` is running, the new `clsbd` exits. If another `clsbd` is not running, the new `clsbd` exits and displays the following error:

```
"Fatal (clsbd): port number xxx is in use by a program that is not clsbd. Exiting."
```

If you get this error, close and restart the Cadence application. However, if you get this error repeatedly, you have to set a different port number for the Boolean daemon or for the application that is using the Boolean daemon port number.

Important

> Because the Boolean daemon is run on all systems that run Cadence applications, if you change the Boolean daemon's port number you must change it on every system, regardless of the platform.

To change the Boolean daemon port number,

1. Exit all Cadence applications.
2. Stop all `clsbd` processes on all systems.
3. Set the following environment variable on every system that runs Cadence applications:

   ```
   CLS_PORTNUM portNumber
   ```

   where *`portNumber`* is a port that is not being used.
4. Restart Cadence applications.

Note: The value of `CLS_PORTNUM` must be the same on every system that runs Cadence applications on any platform.

Related Topics

Troubleshoot Cadence Locking System Problems

Edit Locks

Cadence$^{®}$ locking system (CLS) locks files with Edit locks. An Edit lock on a file signifies that a process is editing the file and prevents other processes from getting an Edit lock simultaneously on the file. However, a process can steal an Edit lock from another process.

Files remain locked even if the network, or the client or server computer, fails. If the process that holds the lock exits without releasing the lock, the lock is considered stranded. for information about how CLS handles stranded locks, see Recover Stranded Locks.

A file can be locked only if its parent directory has `write` permissions.

Lock-Stake File

Edit locks are implemented using Lock-Stake files. The existence of a corresponding Lock-Stake file in the same directory as a file indicates that the file is locked.

When an application gets an Edit lock on a file, CLS creates the following Lock-Stake file in the same directory as the file:

```
filename.cdslck
```

where *`filename`* is the name of the file that is locked.

The Lock-Stake file, an ASCII file, contains information about the lock and the process that holds the lock. The Lock-Stake file has the following information:

- Lock-Stake version
- Login name of the user who locked the file
- Host name (including domain name) of the process that locked the file
- Process ID of the process that locked the file
- Time and date when the process was created

  There are two entries for the time:

  - `ProcessCreationTime_UTC`, the time in Coordinated Universal Time format
  - `ProcessCreationTime_Readable`, the time in a readable format
- Application that holds the Edit lock
- Type of operating system
- Reason for the Edit lock on the file

- File path used to lock the file
- Time and date when the file was locked

The following is a sample `.cdslck` file:

```
# Edit Lock-Stake file. CAUTION: Do not change.
# Information about current Edit Lock Owner.
#
LockStakeVersion               1.0
LoginName                      ns
HostName                       cds111.cadence.com
ProcessIdentifier              11503
ProcessCreationTime_UTC        928949249
ProcessCreationTime_Readable   Wed Jun 09 10:27:29 1999 PDT
OSType                         sun4v
ReasonForPlacingEditLock       XXopen upgrade lock
TimeEditLocked                 Wed Jun 09 10:29:21 1999 PDT
```

A file cannot have more than one Edit lock, so it can have only one corresponding Lock-Stake file.

The Lock-Stake file is automatically deleted by the process that owns it when it releases the Edit lock on the file. You can also delete the Lock-Stake file using the CLS administrative interface.

Steal Edit Locks

Applications can steal Edit locks from another process. When a lock is stolen, information about the new owner is added to the Lock-Stake file (*`filename`*`.cdslock`). Information about previous owners is retained at the end of the file.

A few Cadence applications provide this option of stealing locks. See your application’s documentation for information.

Applications cannot steal locks in compatibility mode.

Related Topics

CLS Administrative Tool

Lock Links

In this topic, we are discussing locking the symbolic and hard links.

- Lock Symbolic Links

  Cadence® locking system (CLS) handles symbolic links automatically. When an application tries to lock a symbolic link, CLS locks the file to which it is linked. The Lock-Stake file is created in the directory that contains the file. If the file does not exist, CLS displays an error.

- Lock Hard Links

  CLS does not lock hard links by default. However, you can set your environment to lock hard links.

  When an application tries to lock a hard link, CLS creates a Lock-Stake file in the directory that contains the hard link, not in the directory that contains the file.

  Locking hard links is risky because you can get multiple Edit locks simultaneously on a file.

Setting Your Environment to Lock Hard Links

To force CLS to lock hard links,

➤ Set the following environment variable:

```
CLS_LOCK_HARD_LINKS YES
```

Important

If both the hard link and a file point to the same chunk of memory, then the hard link will be locked but the original file will remain intact. However, when an application tries to lock the hard link, CLS creates the Lock-Stake file in the directory associated with one of these file handles (hard link). All the other file handles (other hard links) to the same file space are not protected. Therefore, multiple Cadence locks via the other file handles are not prevented. So, resolving the links is always the better option.

Related Topics

Edit Locks

Recover Stranded Locks

Recover Stranded Locks

Stranded locks are locks that remain after the process that owns them has exited. Locks can be stranded if the application or application host exits abnormally, or if the file server is inaccessible.

Stranded Edit locks are recovered in the following ways:

- The Cadence locking system recovers a stranded Edit lock when a process tries to lock the file that has the stranded Edit lock.

  When a process wants to lock a file that is already locked, Cadence® locking system (CLS) determines the status of the process that owns the Edit lock on the file (if the process is running on another system, CLS sends a query to a Boolean daemon running on that system). If the process that owns the lock has exited, the new process takes over the stranded lock—the Lock-Stake file (`*filename*.cdslck)` is overwritten with information about the new process. If the process is active or if its status cannot be determined, the old lock is not modified.

- You can search for and remove both stranded and good Edit locks using the CLS administrative tool (`clsAdminTool`).

  The `clsAdminTool` does not differentiate between stranded locks and non-stranded locks.

Related Topics

CLS Administrative Tool

Edit Locks

CLS Administrative Tool

The Cadence® locking system (CLS) administrative tool lets you view and release locks in any directory hierarchy.

The `clsAdminTool` does not differentiate between stranded locks and non-stranded locks.

With the CLS administrative tool, you can:

- Remove a specific Edit lock
- View all Edit locks in a directory hierarchy
- Remove all Edit locks in a directory hierarchy
- View all Edit locks belonging to a process in a directory hierarchy
- Remove all Edit locks belonging to a process in a directory hierarchy
- View and remove all locks that are in the CLS format and that have the `.cdslck` extension, regardless of whether they were set by CLS or OpenAccess locking

You can use the adminstrative tool interactively (as a shell interface) or in batch mode.

Note: You need to manually clean up lock stakes as the `clsAdminTool` automatically recovers any stranded locks. You can however use the `clsAdminTool` to clean up lock stakes from those libraries that have been, or will be, tarred for transfer (as the hosts that obtained these locks would not exist in the new environment).

clsAdminTool Commands

The CLS administrative tool consists of the following commands, which can be used in both interactive mode and batch mode:

- `ale`

  `ale` *`directoryHierarchy`*

  Lists all Edit locks in the directory hierarchy you specify. For each edit lock, the following information is provided: the file path, host name, user name of the user who locked the file, process ID, and the time the process was created.

- `aple`

  `aple` *`directoryHierarchy hostName`* `[`*`processID`* `[`*`processCreationTime`*`]]`

  Lists all Edit locks in the directory hierarchy that match the host name, process ID, and process-creation time that you specify. The *`processID`* and *processCreationTime* arguments are optional. However, if you need to specify *processCreationTime*, you must also specify *processID*.

- `are`

  `are` *`directoryHierarchy`*

  Releases all Edit locks in the directory hierarchy you specify and displays the list of released locks.

- `apre`

  `apre` *`directoryHierarchy hostName processID processCreationTime`*

  Releases all Edit locks in a directory hierarchy that match the host name, process ID and process-creation time you specify and displays the list of released locks.

- `asre`

  `asre` *`filePath`*

  Removes the Edit lock on the file you specify.

Command Arguments

`filePath` Full path to the file.

`directoryHierarchy`

The path to the directory hierarchy you want to search for CLS Edit locks.

`hostName` The host of the process that owns the locks. For example:

`cds111.cadence.com.`

If you do not provide the domain name, `clsAdminTool` takes the domain name from the system you are using. If you do not want `clsAdminTool` to add the domain name, use the `-force` command or the `-force` argument to `clsAdminTool`.

`processID` The ID of the process that owns the locks.

`processCreationTime`

The time when the process was created.

You can get this information from the Lock-Stake file, *`filename`*`.cdslck`, which is in the same directory as the locked file.

Related Topics

Changing the CLS Boolean Daemon Port Number

Running the CLS Administrative Tool in Interactive Mode

In interactive mode, the Cadence® locking system (CLS) administrative tool is a shell interface that lets you run one command at a time.

To start the CLS administrative tool in interactive mode,

➤ In a UNIX shell, type

```
clsAdminTool [-noprompt] [-echo] [-force]
```

`-noprompt` The > prompt is not displayed.

`-echo` The input is echoed.

`-force` Sets the force mode, which forces `clsAdminTool` to use a simple host name, that is, a host name without a domain. If `clsAdminTool` is not in force mode and you do not provide a domain name with the *`hostName`* argument of the `aple` and `apre` commands, `clsAdminTool` automatically adds the domain name of the system you are using.

You can also set the force mode with the `force` command.

The shell displays the > prompt. You can type `clsAdminTool` commands at this prompt.

To display the version number of the CLS administrative tool,

➤ In a UNIX shell, type

```
clsAdminTool -version
```

Using the CLS Administrative Tool

After you start the administrative tool, at the > prompt,

➤ Type one of the following clsAdminTool commands: `ale`, `aple`, `are`, `apre`, `asre`.

In addition, you can use the following commands:

- `force`

  Sets the force mode, which forces `clsAdminTool` to use a simple host name, that is, a host name without a domain. For example, `cds111` is a simple host name, while `cds111.cadence.com` is not.

If `clsAdminTool` is not in force mode and you do not provide a domain name with the *`hostName`* argument of the `aple` and `apre` commands, `clsAdminTool` automatically adds the domain name of the system you are using.

You can also set the force mode with the `-force` argument to `clsAdminTool`.

- `help`

  Displays a description of `clsAdminTool` commands.

- `system`

  `system` *`commandName`*

  Executes the shell command you specify.

- `quit`

  Exits the `clsAdminTool` interface.

- `exit`

  Exits the `clsAdminTool` interface with the return status of the last command used in `clsAdminTool`. This command is mainly used when `clsAdminTool` is run from scripts.

  For example, if you have a file `myfile`, which contains the following:

  ```
  ale mixSigLib/OpAmp
  exit
  ```

  and you use the following command:

  ```
  clsAdminTool < myfile
  ```

  `clsAdminTool` lists all the locks in the `mixSigLib/OpAmp` directory hierarchy and then exits.

Exiting the CLS Administrative Tool

To exit the CLS administrative tool

➤ At the > prompt, type

```
quit
```

The prompt changes from > to the default shell prompt.

Displaying the CLS Administrative Tool Options

To display the arguments you can use with the `clsAdminTool` command,

➤ At the UNIX command-line, type:

```
clsAdminTool -help
```

Related Topics

clsAdminTool Commands

Using the CLS Administrative Tool in Batch Mode

In addition to using the Cadence® locking system (CLS) administrative tool interactively in a shell interface, you can run it in batch mode on a UNIX command-line.

In batch mode, you can use any of the clsAdminTool commands—`ale`, `aple`, `are`, `apre`, `asre`—as arguments to `clsAdminTool`.

To run the administrative tool in batch mode,

➤ At the command-line, type the following command:

```
clsAdminTool [-force] -Command commandOptions
```

where `-Command commandOptions` is one of the following:

```
-ale aleOptions
-aple apleOptions
-are areOptions
-apre apreOptions
-asre asreOptions
```

For a description of these commands and their options, see clsAdminTool Commands.

`-force` sets the force mode, which forces `clsAdminTool` to use a simple host name, that is, a host name without a domain. For example, `cds111` is a simple host name, while `cds111.cadence.com` is not. If you do not use this argument and you do not provide a domain name with the *`hostName`* argument of the `aple` and `apre` commands, `clsAdminTool` automatically adds the domain name of the system you are using.

To display the arguments you can use with the `clsAdminTool` command,

➤ At the command-line, type:

```
clsAdminTool -help
```

You can also create aliases for clsAdminTool commands and use the alias to call the command. For example:

```
alias findLocks “clsAdminTool -ale .”
```

or

```
alias cleanLocks “clsAdminTool -are .”
```

To display the version number of clsAdminTool,

➤ At the command-line, type:

```
clsAdminTool -version
```

Related Topics

CLS Administrative Tool

Troubleshoot Cadence Locking System Problems

Problem: Application hangs on lock request to a remote system

For NFS prior to version 4, check whether the network lock daemons `lockd` and `statd` are running on the remote system. `lockd` and `statd` must always be running on every UNIX system that runs Cadence applications or stores data. Some applications that use read and write locks use the operating system's `fcntl` function, which relies on `lockd` and `statd` daemons. (CLS Edit locks no longer use `fcntl`.) If the daemons are running, but calls to them still hang, verify that you have the latest operating system patches.

Problem: Locking requests to a Linux machine fail with the following error:

```
"No locks available"
```

While CLS Edit locks no longer use `fcntl`, some applications that use read and write locks do use the operating system's `fcntl` function. Locking requests to a Linux machine will fail if the Linux NFS server does not support locking.

To solve this problem, upgrade your NFS software to a version that supports locking.

Problem: Failed to obtain lock information on a cellview:

```
"no data on lock available"
```

This message appears when the SKILL function `geGetLockInfo()` fails to obtain lock information from the `.cdslck` file or there is no lock available on a cellview. For more information, see `geGetLockInfo()`.

Problem: Lock requests fail for files locked by an application running on remote systems

Check whether the Boolean daemon (`clsbd`) is running on the remote system on which the application is running. The Boolean daemon must always be running on every system that runs Cadence applications. For more information about how to start it, see Starting the Boolean Daemon.

Problem: You cannot start the Boolean daemon or you get the following error:

```
"Fatal (clsbd): port number xxx is in use by a program that is not clsbd. Exiting."
```

Check if another application is using the Boolean daemon's port number (16723). If another application is using the port number, you might need to change the Boolean daemon's default port number.

Problem: Your Cadence application does not run and you get the following error:

```
*Warning* file /usr/xyz/CDS.log Malformed Lock-Stake file.
```

```
Failed to lock log file: /usr/xyz/CDS.log
```

This error message indicates that the Lock-Stake file is either empty or corrupted. A Lock-Stake file can be empty if it was created when the disk was full. A Lock-Stake file can be corrupted if it was manually edited or if the application or system exited abnormally.

Resolve the cause of the problem. If it is safe to remove the lock on the file, remove the Lock-Stake file (*filename*.cdslck) manually or with clsAdminTool.

Problem: If the machine that a cellview lock is registered to is no longer available, DFII will only inform that the cellview is still locked after a lengthy period of time.

This situation can arise after several minutes of DFII trying to access the cellview, via an exchange with oaFSLockD, only to inform that the cellview cannot be accessed, and remains locked.

You can however reduce the time that will be allowed for such an exchange, using the CLS_CLSBD_CONNECT_TIMEOUT environment variable. For example:

```
setenv CLS_CLSBD_CONNECT_TIMEOUT 5
```

where, 5 = 5 seconds.

Related Topics

Changing the CLS Boolean Daemon Port Number

CLS Administrative Tool

11

Miscellaneous Infrastructure Technologies

This topic describes miscellaneous infrastructure technologies. It covers the following technologies:

- The cds_root Utility
- Temporary Directory Standard
- Log File Environment Variables

The cds_root Utility

`cds_root` is a utility that identifies the location of Cadence installation hierarchies. Startup scripts typically use `cds_root` to find the location of Cadence installation hierarchies before starting applications. `cds_root` requires one argument—an executable name.

Use the following syntax for `cds_root`:

```
cds_root executableName
```

where *`executableName`* is either an executable found in `$PATH` or is a full path to an executable.

`cds_root` uses the *`executableName`* argument to identify the installation hierarchy of the application and to check if it is a legal hierarchy. If you specify the full path to the executable, then `cds_root` checks only that location to see if it is a legal hierarchy.

If the executable is not found in `$PATH` or is not located in the hierarchy, `cds_root` displays the following error message:

- `Error! Can’t determine installation root from PATH`

Temporary Directory Standard

Cadence applications have a new standard for determining the location of temporary files. Note, however, that not all applications use this standard yet.

Unix and Linux Platforms

According to the standard, applications running on UNIX and Linux platforms use the following order of precedence to determine the temporary directory:

- `$CDS_TMP_DIR`, if the variable is set and the directory has write permissions
- `$TMPDIR`, if the variable is set and the directory has write permissions
- `$TEMPDIR`, if the variable is set and the directory has write permissions
- `/tmp`

Microsoft Windows

According to the new standard, applications running on Microsoft Windows use the following order of precedence to determine the temporary directory:

- `$CDS_TMP_DIR`, if the variable is set and the directory has write permissions
- `$TMP`, if the variable is set and the directory has write permissions
- `$TEMP`, if the variable is set and the directory has write permissions
- `.` (current directory)

Compatibility with Old Behavior

If you want your application to continue to use the old way of determining the temporary directory, do the following:

➤ Set the following environment variable:

```
CDS_OLD_TMP_LOC
```

Your application will then use the old behavior. For some applications, the old behavior was the same as the new standard, while, for others, the old behavior was the following:

- For UNIX platforms, `$TEMP` was used as the temporary directory if it was set, otherwise `/tmp` was used.

- For NT platforms, the order of precedence was as follows:
  `$TMP`
  `$TEMP`
  the Windows directory

Log File Environment Variables

You can set environment variables to control log files that are created by Cadence applications. You can specify the location, the type of version, and the maximum number of log files.

Note: Not all Cadence applications use these variables; refer to the documentation of the application you are using for more information.

Specifying the Location of Log Files

To specify a location for log files,

➤ Set the following environment variable:

```
setenv CDS_LOG_PATH directoryList
```

where *`directoryList`* is a colon-separated list of directories.

Panic log files are created in the first directory in *`directoryList`* that has write permissions.

You can also specify the panic.log file path(s) using the `CDS_LOG_PATH` environment variable. For example, `setenv CDS_LOG_PATH my_directory_path_1:my_directory_path_2......"`

Note: The first writable directory in the list will be used.

The panic.log file name stored at the specified location will have a unique name. For example, `panic.log.sjfsb015.3081`. The panic.log file name has the extension of `hostName` and `processID` to make the name unique.

The output message on your shell window will also display the unique name i.e. `panic.log.sjfsb015.3081`.

The `panic.log.sjfsb015.3081` file will be in the user's home directory if the CDS_LOG_PATH environment variable is not specified.

Specifying the Type of Version

To specify the the type of version you want to use for log files,

➤ Set the following environment variable:

```
setenv CDS_LOG_VERSION pid | sequential
```

If you set `CDS_LOG_VERSION` to `pid`, the application creates (and locks) *`logFile.processId`*, where *`processId`* is the process ID of the application.

If you set `CDS_LOG_VERSION` to `sequential`, the application looks for the highest version of *`logFile.version`* in the directory and increments the version by 1. For example, if the directory has the log file `CDS.log.3`, the application creates (and locks) `CDS.log.4`. If, for some reason, the application cannot create the next version, it tries up to *`version`* +10.

If `CDS_LOG_VERSION` is not set or is set to an illegal value, the application tries to use (and lock) *`logFile`*. If that fails, the default behavior is `sequential`, that is, the application tries to create *`logFile`*`.1` through *`logFile`*`.10`.

(The name of the log file is determined by the application.)

Specifying the Maximum Number of Log Files

Note: The Virtuoso Studio design environment does not use this variable.

To specify the maximum number of log files that can be created in a directory,

➤ Set the following environment variable:

```
setenv CDS_LOG_LIMIT maxNumber
```

where *`maxNumber`* is the maximum number of log files that can be present in the directory.

If *`maxNumber`* is reached, when the next log file is created, the oldest log file in the directory will be deleted. For example, if `CDS_LOG_LIMIT` is 4, and the directory has the following log files:

```
CDS.log.1
CDS.log.2
CDS.log.3
CDS.log.4
```

the next log file will `CDS.log.5` and the oldest log file, `CDS.log.1`, will be deleted. The directory will have the following files:

```
CDS.log.2
CDS.log.3
CDS.log.4
CDS.log.5
```

However, if the oldest log file is locked, it will not be deleted. Instead, the next unlocked file will be deleted. In the above example, if `CDS.log.1` is locked, then `CDS.log.2` will be deleted instead (if it is unlocked) and the directory will have the following files:

```
CDS.log.1
CDS.log.3
CDS.log.4
CDS.log.5
```

If all the existing log files are locked and *`maxNumber`* is reached, no new log files will be created and an error will be displayed.

If `CDS_LOG_LIMIT` is set to a lower number than the number of log files already present in the directory, the oldest log files will be deleted to bring the number of files down to the limit. For example, if a directory has `CDS.log.1...CDS.log.15` and you set `CDS_LOG_LIMIT` to 10, when `CDS.log.16` is created, `CDS.log.1...CDS.log.6` will be deleted.

Note: `CDS_LOG_LIMIT` only applies when `CDS_LOG_VERSION` is set to `sequential`. It is ignored when `CDS_LOG_VERSION` is `pid`.

Related Topics

cdsDaemonStarter Configuration

Support for IBM® Rational® ClearCase® Dynamic Views through the MultiVersion File System

For most databases, Virtuoso reads into memory only the data that is used by an application. This partial-read functionality is intended to minimize the virtual memory requirements of an application. For partially read databases opened in read mode, Virtuoso creates an additional hard link to the database file when the file is accessed across an NFS mount; this is done to ensure that the process has a handle to the original file in case the file is modified on the remote host.

There are known issues with ClearCase dynamic views and programs like Virtuoso that hold files open for an extended period of time —that is, multiple days— without accessing the file. In particular, the ClearCase VOB (Versioned Object Base) scrubber removes files from the VOB cleartext cache that have not been accessed for a certain number of days (configurable by the ClearCase administrator) to reduce disk space usage on the VOB servers.

Note: Some Linux NFS client implementations do not update the last-access-time every time a file is accessed, increasing the odds of encountering a problem on Linux.

If Virtuoso holds a database open for a prolonged period of time without accessing it, it may become inaccessible due to the ClearCase VOB scrubber's configured behavior and this prevents reading any more data from the file. To address this issue, Virtuoso introduces and recognizes a new UNIX environment variable, `OA_USING_MVFS`.

ClearCase users who keep databases open for long periods of time and experience “stale NFS file handle” or ‘file not found” errors can set the `OA_USING_MVFS` environment variable. Virtuoso will perform additional checking and if a database is in a library that is on an MVFS mount point, the database will be fully read into memory, therefore, avoiding the use of a hard-link and keeping the database open for an extended period of time.

12

Occurrence Property Dictionary

An occurrence property dictionary is a central repository for simulation-control property definitions. Applications such as UltraSim register properties in the property dictionary via application-specific property dictionary files. Other applications, such as the Cadence Hierarchy Editor, which do not necessarily know the semantics of the properties, can then be used to set the properties.

The use of a property dictionary enables the Hierarchy Editor to provide the right error checks when users set properties. It also prevents multiple applications from using the same property name for different purposes or with different property characteristics, such as a different default value.

The property dictionary only contains property definitions, which include information such as the default value of a property, the applications it applies to, and any restrictions on its use. The property dictionary does not store the values of these properties; property values are stored in the property file `prop.cfg`.

You can also customize the property dictionary by providing property definitions in a user property dictionary file (`propdict.def` file). You typically do this only for properties that you use frequently with an application but that have not been added to the dictionary by the application. The `propdict.def` file is typically placed in a site-level directory; however, you can also have user property dictionary files at the personal or group levels. The Cadence Setup Search File mechanism (CSF) is used to find the file.

Important

> The occurrence property dictionary is only for simulation-control properties.

Property Dictionary File

The property dictionary comprises two types of files:

- Application property dictionary files

Each application that uses the property dictionary provides an application-specific property dictionary file named *appName*.def.

Do not modify *appName*.def files. If you want to customize property definitions, create a user property dictionary file instead.

- User property dictionary files

  User property dictionary files are named propdict.def and can be located in any directory that is found by the CSF search mechanism. You can have more than one propdict.def file.

  User property dictionary files are read after the application files are read.

Both application and user property dictionary files have the same format.

The property dictionary file has the following format:

```
/* Comment */

// Comment

occPropDef propName
{
    attribute;
    attribute;
    ...
}

occPropDef propName addAffectedApp “appName”;

include path;
```

Related Topics

Customizing the Property Dictionary

Property Definition Statements

Property definition include the following statements:

- Comments
- Property definitions (occPropDef statements)
- addAffectedApp statements
- include statements

Comments

Comments have the following syntax:

- Multiple-line comment:

  ```
  /* comment */
  ```

- One-line comment that continues till the end of the line:

  ```
  // comment
  ```

Property definitions (occPropDef statements)

Property definitions specify the default value of a property, the value type and value range that is allowed, a description, a tool tip, the applications the property applies to, and any restrictions on its use.

Property definitions have the following syntax:

```
occPropDef propName
{
    attribute;
    attribute;
    ...
}
```

where *propName* is the name of the property and *attribute* is one of the following:

`valueType = int | double | enum | string;`

Specifies the type of the property value. The following types are supported: integer, double, enumerated, and string.

Example:

```
valueType = int;
```

`valueDefault = `*`default`*`;`

Specifies the default value of the property. The type of the default value must correspond to the type specified by the `valueType` attribute.

Example:

```
valueDefault = 6;
```

`valueRange = `*`range`*`;`

Specifies the range of legal values. The syntax of *`range`* varies depending on the type of the value.

If `valueType = int`, then the syntax is:

```
valueRange = “integer <|<= value <|<= integer”;
```

If `valueType = double`, then the syntax is:

```
valueRange = “double <|<= value <|<= double”;
```

If `valueType = enum`, then the syntax is:

```
valueRange = “word word ...”;
```

Examples:

```
valueRange = -6 <= value < 14;
valueRange = 1.54 < value <= 4.8;
valueRange = “standard accelerated fast”;
```

`affectedApp: “`*`appName`*`”;`

Specifies the application the property applies to. A property definition can have more than one `affectedApp` expression.

Example:

```
affectedApp: “AMSUltra”;
```

`useRestrictions = `*`restriction`*`;`

Specifies any restrictions on the use of the property.

Syntax for one restriction:

```
useRestrictions = restriction;
```

Syntax for more than one restriction:

```
useRestrictions = "restriction restriction ...";
```

where *`restriction`* is one of the following:

```
cantBeOnOcc   (cannot be placed on an occurrence)
cantBeOnInst  (cannot be placed on an instance)
cantBeOnCell  (cannot be placed on a cell)
cantBeGlobal  (cannot be a global property)
cantInherit   (cannot be inherited)
```

Example:

```
useRestrictions = "cantBeGlobal cantInherit";
```

`description = "`*`description`*`";`

Specifies a description of the property.

`description` is either a set of non-special characters that do not contain whitespace or a string of printable characters that is enclosed in double quotes. If *`description`* is in quotes and you want to include a double quote or a backslash in the string, escape it with a preceding backslash: `\"` or `\\`. To specify a newline, end the line with a backslash.

Example:

```
description = "Defines the simulation speed and accuracy\
within the chosen simulation mode.";
```

`toolTip = "`*`tip`*`";`

Specifies a tool tip for the property, that is, the descriptive text that will be displayed when the cursor is placed over the property.

`tip` is either a set of non-special characters that do not contain whitespace or a string of printable characters that is enclosed in double quotes. If *`tip`* is in quotes and you want to include a double quote or a backslash in the string, escape it with a preceding backslash: `\"` or `\\`.

If `valueType` is `enum`, `toolTip` has the following syntax:

```
toolTip = "General description; description for enum 1;
description for enum 2; ...";
```

The following general rules apply to attributes:

- Each attribute expression ends with a semi-colon.
- Attributes that use a colon instead of an equals sign can be used multiple times in an `occPropDef` statement. Attributes that use an equals sign can only be used once.
- Whitespace is optional between *`attributeName`* and = and between = and *`attributeValue`*. Similarly, whitespace is optional between *`attributeName`* and `:` and `:` and *`attributeValue`*.
- A legal *`attributeValue`*, unless otherwise specified above, is either a set of non-special, non-whitespace characters or a string of printable characters that is enclosed in double quotes.
- If *`attributeValue`* is in quotes and you want to include a double quote or a backslash in the string, escape it with a preceding backslash: `\"` or `\\`. To specify a newline, end the line with a backslash.

addAffectedApp statements

The `addAffectedApp` statement adds an application to an existing property definition. It does not have to be in the same file as the property definition.

The `addAffectedApp` statement has the following syntax:

```
occPropDef propName addAffectedApp "affectedApp";
```

For example, the statement

```
occPropDef mos_method addAffectedApp "Hspice";
```

adds the application Hspice to the previously-defined property `mos_method`.

include statements

A property dictionary file can include other property dictionary files with the `include` statement.

The `include` statement has the following syntax:

```
include path;
```

where *`path`* is the path to the property dictionary file to be included. If the path is relative, it is interpreted as being relative to the file that contains the `include` statement. *`path`* can include environment variables such as `$HOME` and installation root expressions such as

`$(inst_root_with: `*`path`*`)`. Included files do not have to be named `propdict.def` or *`appName`*`.def`.

Related Topics

Installation Root Expressions

Customizing the Property Dictionary

If you want to add properties to the property dictionary, you can do so by customizing the dictionary.

In general, you do not need to customize the dictionary. However, if there are properties that you use frequently with an application, and you want the ability to set them through the Hierarchy Editor, you can add them to the dictionary. Any properties you add will be displayed in, and can be set through, the Hierarchy Editor. Also, since the Hierarchy Editor uses the property definitions to do appropriate error checks such as checking whether a specified value is of the right type or is within the legal range, adding frequently-used properties to the dictionary provides a way of minimizing errors when the properties are set.

To customize the property dictionary, you create a user property dictionary file called `propdict.def`.

A `propdict.def` file can include other `propdict.def` files. Therefore, you can have multiple `propdict.def` files. For example, if you have customized the property dictionary at different levels such as the personal, group and site levels, your personal `propdict.def` file can include the group and site `propdict.def` files.

The Cadence Setup Search mechanism (CSF) is used to find the `propdict.def` file to use.

To create a `propdict.def` file:

1. Create an ASCII file called `propdict.def` in any directory that is listed in your `setup.loc` file, for example, `$HOME`.
2. In the `propdict.def` file, specify the changes you want to make to the property dictionary. You can do the following:
   - ❑ Add property definitions

     Specify property definitions with the `occPropDef` statement.
   - ❑ Include other user property dictionary files

     Include files with the `include` statement. Included files do not have to be named `propdict.def`.

You should only add simulation-control properties that are understood by a Cadence application.

Sample Property Dictionary File

```
//----------------------
```

```
// Property definitions
//----------------------


occPropDef speed
{
    valueType = enum;
    valueRange = "1 2 3 4 5 6 7 8";
    valueDefault = "5";
    description = "Sets the speed/accuracy tradeoff for simulation.";
    tooltip = "Sets the speed/accuracy tradeoff for simulation.";
    affectedApp: "UltraSim";
}


occPropDef sim_mode
{
    valueType = enum;
    valueRange = "s a amr ms da df";
    valueDefault = "ms";
    toolTip = "Sets the simulation mode; SPICE mode; Analog mode; Analog \
                Multi-rate mode; Mixed signal mode; Digital accurate mode; \
                Digital fast mode";
    affectedApp: "UltraSim";
}
```

Related Topics

Cadence Setup Search File: setup.loc

Property Dictionary File

13

cdsDaemonStarter Configuration

cdsDaemonStarter configuration enables you to launch a program without inheriting the control group of its parent. The following programs are started in Virtuoso using an optional cdsDaemonStarter script:

- clsbd
- oaFSLockD
- cdsXvnc
- cdsNameServer

  The cdsNameServer program is used to establish communications between Cadence programs. It relies upon the TCP port number, 7325. In addition, whenever needed, the cdsNameServer program starts automatically and exits itself after being idle.

Control Groups

When the Linux control group functionality is enabled, a job is considered to be running until all the processes in the control group exit. This causes a problem because some processes are intended to daemonize themselves and continue to run after the parent job has finished.

To solve this problem, a support for an optional wrapper script has been introduced, which is automatically invoked by the Cadence software to start daemonizing processes. The arguments to the wrapper script indicates which daemon program (and its arguments) should start.

This is illustrated by the following pass-through example:

```
#!/bin/ksh
exec "$@"
```

The korn shell uses "$@" to denote the arguments provided to the wrapper script.

You can use the following wrapper script to launch the program in a specific control group:

```
#!/bin/ksh
cgexec -g someControlGroupInfo --sticky "$@"
```

If Load Sharing Facility (LSF, version 9) control group functionality is enabled, you can use the following wrapper script to launch the program:

```
#!/bin/ksh
# If it looks like an LSF environment, look for job-specific control group
if [ -n "${LSF_CGROUP_TOPDIR_KEY}" -a -n "${LSB_JOBID}" -a \
    -n "${LSF_BINDIR}" -a -r /proc/self/cgroup ]; then
    if grep ":/lsf/${LSF_CGROUP_TOPDIR_KEY}/job\.${LSB_JOBID}\." \
            /proc/self/cgroup >/dev/null; then
        exec ${LSF_BINDIR}/lsgrun -m $(uname -n) "$@" < /dev/null
        # only reached if above exec failed
    fi
fi
exec "$@"
```

As illustrated above, the `lsgrun` command starts the daemon in a different control group than the parent.

Note: You can also copy the LSF daemon starter sample script from the following location:

`<cdsInstallDir>/cdssetup/daemon/starter.lsf.sample`

If clsbd or other daemons are not exiting and preventing LSF jobs to get completed then use this script to start daemons in the LSF environment.

Note: You can copy the OpenLava (OLV) daemon starter sample script from the following location:

`<cdsInstallDir>/cdssetup/daemon/starter.olv.sample`

Note: The `$HOME` variable must be set to ensure that these daemon processes are started.

Wrapper Installation

Normally, the wrapper is installed as `${CDS_SITE}/cdssetup/daemon/starter`, where the default value for `${CDS_SITE}` is `<cdsInstallDir>/share/local/`. You can also install the wrapper as `cdssetup/daemon/starter` into a directory that is searched by the Cadence Setup File (`csf`) package.

Important

> It is recommended that you test the wrapper before wide-spread deployment by installing it into a local location, such as `~/cdssetup/daemon/starter`. You must also ensure that you have appropriate execute permissions.

After installing the wrapper, you can use the following command to confirm that it is found by `csf`:

```
% cdswhich cdssetup/daemon/starter
```

Caution

As some programs change their working directory on startup, you should avoid installing the wrapper in the current working directory.

Related Topics

Search Mechanism in Cadence Setup Search File

A

Common Design Files

Using the Cadence library structure, you can create files using a variety of different applications. The following table lists some of the more common filenames used by Cadence applications.

Filenames	Application	Information Stored in File
data.dm	OpenAccess	Properties (OpenAccess libraries only)
layout.oa	Custom Layout editors	IC layout
netlist.ahd	AHDL (Analog HDL)	ASCII source
netlist.oa	OpenAccess	Netlist
netlist.src	Verilog	ASCII source in HDL Composer
pc.db	Various applications	Parent-child database information
sch.oa	Virtuoso schematic editor	Schematic
symbol.oa	Virtuoso schematic editor	Symbol
symbol.cnc	Concept	Symbol
text.ahd	AHDL (Analog HDL)	ASCII source
verilog.src	Verilog	ASCII source in HDL Composer
verilog.v	Verilog	ASCII source
vhdl.vhd	VHDL	ASCII source
vhdl.ast.arch.rev	VHDL	Intermediate file produced by Leapfrog
vhdl.cod.arch.rev	VHDL	Intermediate file produced by Leapfrog

Filenames	Application	Information Stored in File
*.cdl	Various applications (for example, netlist, schematic, symbol).	Various applications (for example, netlist, schematic, symbol).
thumbnail_*.png	Various applications (for example, netlist, schematic, symbol).	Visual snapshot of the enclosing views.

Related Topics

Cadence Application Infrastructure