Cisco Embedded Automation Systems (EASy) Installer Guide

Introduction

About EASy Installer

The EASy Installer tool is a Cisco® Tool Command Language (Tcl) script designed to run within the Cisco IOS® **tclsh** environment. EASy Installer provides a framework to install EASy packages using a menu-driven, data-driven interface.

Limitations of EASy Installer

EASy Installer requires **tclsh**. Tcl scripting in Cisco IOS Software was first introduced in release **12.3(2)T** and then merged into **12.2(25)S** and **12.2(33)SXH**. However, it is known to be present and functional in other Cisco IOS releases, such as **12.2(40)SE** and **12.2(18)SXF5**. EASy Installer endeavors to support **tclsh** on as many platforms as possible. However, it should be noted that EASy packages primarily use the Cisco IOS Embedded Event Manager (EEM). EEM was not introduced until **12.3(4)T**.

EASy Installer also requires that the device have at least one flash file system that supports the **mkdir** command. The device must also support the **archive tar /xtract** command. This command is widely available on many platforms. If the device lacks a file system capable of **mkdir**, or does not support **archive tar /xtract**, EASy packages must be installed manually.

About EASy Packages

EASy packages are nothing more than tar archives. Within the archive are multiple files. Some of the files are metadata files that tell EASy Installer how they should work. Others are the actual EEM policies that will be installed and configured on the device.

EASy package names should end with a ".tar" extension. If you receive a package which has been further compressed (e.g. with Zip, GNU zip, etc.), make sure you uncompress it before trying to install it onto your device. Do **NOT** untar the file yourself. The EASy Installer will handle extracting the tar file.

About This Guide

This guide explains how to create EASy packages that can be used by the EASy Installer. It also discusses how to use the EASy Installer to install these packages. This guide is meant for users and developers alike.

Installing and Using the EASy Installer

Installing the Easy Installer

Put simply, the EASy Installer does not really need to be installed. Because of the flexibility of Cisco IOS Software, the EASy Installer can reside on a network file server (such as Trivial File Transfer Protocol [TFTP] or Secure Copy [SCP]) and be loaded as needed on each device. However, if you wish, you can copy the **easy-installer.tcl** script to a device's local flash. The script can reside on any flash file system on the device. For example:

```
Router#copy tftp://10.1.1.1/easy-installer.tcl flash:
```

It might be convenient to configure a command alias for EASy Installer so that you need not always invoke EASy Installer by first typing **tclsh**. To do this, configure an EXEC alias such as the following:

```
Router(config)#alias exec easy-installer tclsh flash:/easy-installer.tcl
```

Then you can invoke EASy Installer simply by typing **easy-installer** followed by the appropriate arguments.

Using EASy Installer

To get a short usage summary, run EASy Installer without any arguments. For example, assuming that you have defined an EXEC alias for it, you would issue the following command:

```
Router#easy-installer
Distribution URL and install prefix must be specified.
Usage: easy-installer [--debug] <package URL> <destination>


Usage: easy-installer [--debug] --uninstall --prefix <directory>
       --pkgname <installed package name>


Usage: easy-installer --version


Usage: easy-installer --list [--prefix <directory>]


Usage: easy-installer --help


Usage: easy-installer --usage
```

To see more details about each argument, run EASy Installer with the --**help** argument. For example:

```
Router#easy-installer --help
Usage: easy-installer [OPTIONS...] <package URL> <destination>
  --uninstall                     Perform a package
                                  uninstallation.
  --prefix                        Local directory in which the
                                  package to be uninstalled is
                                  installed.
  --pkgname                       Name of the package to be
                                  uninstalled.
  --debug                         Enable debugging output.
  --version                       Print version and exit.
  --list                          Print a list of installed
                                  packages.
  <package URL>                   URL of the package tar file
                                  to install.
  <destination>                   Local directory into which
                                  the package will be installed.
Help options:
```

```
--help                          Show this help message.
--usage                         Display a brief command
                                summary.
```

To install a new EASy package, specify a URL pointing to the package you wish to install, followed by a local path to the location where EASy packages will live on the device. This second argument can be omitted on future executions, as the local prefix is cached as an EEM environment variable.

For example, if you want to install the EASy package *green-ports.tar* on the TFTP server *10.1.1.1*, and you want EASy packages to be installed into *flash:/easy*, run the following command:

```
Router#easy-installer tftp://10.1.1.1/green-ports.tar flash:/easy
```

This will invoke the EASy Installer menu which will guide you through configuring and installing the package. Before you get the EASy Installer menu, however, you must first agree to the EASy package's license. The first screen you see after launching the EASy Installer for a new package will be similar to the following:

```
Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.
3. Neither the name of Cisco, the name of the copyright holder nor the
   names of their respective contributors may be used to endorse or
   promote products derived from this software without specific prior
   written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS'' AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY

TECHNICAL ASSISTANCE CENTER (TAC) SUPPORT IS NOT AVAILABLE FOR THIS SCRIPT.
For questions or help, send email to ask-easy@cisco.com.
```

```
Do you agree to this license? (y/n) [n]
```

You must answer "y" or "yes" to continue. Once the license has been accepted, you will be presented with the EASy Installer main menu.

```
A typical EASy Installer menu looks like the following:
--------------------------------------------------------------------------------

Configure and Install EASy Package ‘green-ports-1.0’

--------------------------------------------------------------------------------

1. Display Package Description
2. Configure Package Parameters
3. Deploy Package Policies
4. Verify Installed Package
5. Exit

Enter option:
```

To reconfigure a package that is already installed, run EASy Installer with the name of the installed package as the argument. To find the list of packages that are already installed, use the `--list` argument to EASy Installer. For example:

```
Router#easy-installer --list
EASy packages installed:

green-ports-1.0        Automatically shutdown ports to conserve energy
```

In this example, package *green-ports* is installed at version *1.0*. To reconfigure this package, run EASy Installer with *green-ports* as the argument. For example:

```
Router#easy-installer green-ports
```

This will launch the EASy Installer Configure and Uninstall menu. From here, you can change attributes of the installed package or remove the package from the device.

To uninstall an EASy package, either launch the Configure and Uninstall menu as just described or run EASy Installer with the --uninstall argument. For example, to uninstall a package named *green-ports*, use following command:

```
Router#easy-installer --uninstall --pkgname green-ports
```

Creating EASy Packages

Package Layout

An EASy package is a tar archive that contains any number of files. All of the files **must** be in the root directory of the archive. For example, given a package named *custom-mib.tar*, its contents look like the following:

```
$ tar -tvf custom-mib.tar
drwxr-xr-x  0 marcus staff       0 Mar 17 16:27 ./
-rw-r--r--  0 marcus staff    2869 Mar 17 16:27 ./pkgconfig
-rw-r--r--  0 marcus staff     433 Mar 17 15:00 ./envvars
-rw-r--r--  0 marcus staff     164 Mar 17 01:39 ./pkgdescr
-rw-r--r--  0 marcus staff    8378 Mar 17 01:46 ./ExpressionMIB_CLI.tcl
-rw-r--r--  0 marcus staff    9039 Mar 17 15:41 ./ExpressionMIB_SNMP.tcl
```

Package Files

Each EASy package must contain a set of metadata files that drive the EASy Installer. Some of the files can be omitted but may require additional code to facilitate their function. Table 1 describes the metadata files.

Table 1. Metadata Files

Filename	Mandatory	Description
pkgconfig	Yes	**pkgconfig** contains the required variables and Tcl procs necessary to drive EASy Installer to configure and install your package. This file is essentially a Tcl script that is sourced into EASy Installer.
envvars	No	**envvars** contains a list of environment variables used by the EEM policies in your EASy package. The format of this file looks like a list of Tcl arrays, with each variable on its own line: condition **CONDITION** name **NAME** prompt **PROMPT** default **DEFAULT** pattern **PATTERN** range **RANGE** **CONDITION** is an optional global variable name. If the value of the variable is $EASY::FALSE, that environment variable will be skipped during the do_configure phase. However, the variable will still be removed during the do_uninstall phase. The condition element can be omitted entirely. **NAME** is the name of the variable, **PROMPT** is the string to use to prompt the user for the variable value, **DEFAULT** is the default value for the variable if the user simply presses Enter at the prompt, and **PATTERN** is the regular expression to use to check the entered value for correctness. Alternatively, if the value is to be a number within a numeric range, specify the range key instead of pattern. The value of **RANGE** should then be in the form of **MIN-MAX**. The pattern and range keys are optional. If omitted, any value entered by the user is allowed.
manifest	No	**manifest** contains a list of EEM policies to install and register. Each policy file should be on a line by itself.
manifest.optional	No	**manifest.optional** contains a list of EEM policies to install but not register. Each policy file should be on a line by itself.
pkgdescr	Yes	**pkgdescr** contains a description of your package in as many words as necessary. This description is designed to be read by users before installing your package. It will be displayed from the EASy Installer menu.
pkgmessage	No	**pkgmessage** contains a message to display to users after installing your package. This message should contain any additional instructions, caveats, or other information the users may require. If this file is omitted, a default message will be displayed.
pkguninstall	No	**pkguninstall** contains Tcl code that will be run before uninstalling your package. EASy Installer will handle removing the files listed in the **manifest** and **manifest.optional** files and unregistering the package from the package database. If additional cleanup is required, use the **pkguninstall** file.

Other files in the package are not treated specially by EASy Installer. If your package is installing EEM Tcl policies, make sure they obey the EASy scripting conventions.

Variables

EASy Installer uses and provides a number of variables. These variables come in one of three categories: package-provided variables, global variables, and EASY namespace variables. Each of these types is described in the sections that follow.

Package-Provided Variables

Table 2 lists the variables that can be set in a package's **pkgconfig** file.

Table 2. Variables for the **pkgconfig** File

Variable Name	Mandatory	Description
PKGNAME	Yes	**PKGNAME** is the name of the package. The value should contain letters, numbers, and dashes **only**. For example, **custom-mib, green-ports**, and **trap2email** are valid EASy package names. The name of the package file should be the same as **PKGNAME** with .tar appended.
PKGVERSION	Yes	**PKGVERSION** is the version of the package. Valid versions include, **1.0, 2.1.a**, and **0.71**.
PKGCOMMENT	Yes	**PKGCOMMENT** is a short, one-sentence description of your package. It should begin with a capital letter, and **not** end with a period. The length should not exceed 70 characters. The **PKGCOMMENT** is displayed when one views the list of installed packages.
MINRAM	No	The **MINRAM** value is used when EASy Installer checks the requirements for the package. If a value is specified, the device must have at least **MINRAM** kilobytes of memory in order for the package to install.
MINFLASH	No	The **MINFLASH** value is used when EASy Installer checks the requirements for the package. If a value is specified, the device must have at least one flash partition that is at least **MINFLASH** kilobytes in size in order for the package to install.
VERSION_CHECK	No	The **VERSION_CHECK** variable, if specified, must point to a Tcl function name in **pkgconfig** that is called when EASy Installer checks the requirements for the package. It should perform any additional requirements checking beyond RAM and flash checks. For example, this function can call additional functions to check for a minimum Cisco IOS Software release or a minimum version of EEM.
OPTIONS	No	The **OPTIONS** variable, if specified, is a list of extra items to add to the EASy Installer menu. The extra items are added just before the Exit item. The format of each item in **OPTIONS** is a list in itself with the following elements: **NAME DEPENDENCIES TARGETS** **NAME** is the name of the menu item that will be displayed to the user. **DEPENDENCIES** is a list in the format of **CHECK_VAR DEP_TARGETS**, where **CHECK_VAR** is a variable to check to see whether or not the targets specified in **DEP_TARGETS** need to be executed. **TARGETS** is a list of target functions to run to complete the desired tasks for this item.

Global Variables

Global variables are those variables provided by EASy Installer that are available to your code in **pkgconfig** and can be overridden as needed. Table 3 contains a complete list of global variables and their default values.

Table 3. Global Variables

Variable Name	Default Value	Description
PREFIX	N/A	The **PREFIX** is the path into which EASy packages are installed. The value of this variable will be different for every device, depending on where the user wants to put installed packages. The value of this variable will always be a fully qualified path.
DIST_URL	N/A	The **DIST_URL** is available only when performing a package installation. It points to the URL of the package tar archive.
WRKDIR	N/A	The **WRKDIR** points to the location where the package contents can be found prior to installation or when performing a reconfiguration. A package is extracted into **WRKDIR**, and EASy Installer then changes the directory to **WRKDIR**. Therefore, you can reference files relative to **WRKDIR** in your pkgconfig code.
EXTRACT_DONE	*0*	The **EXTRACT_DONE** variable is a boolean that indicates whether or not package extraction has been performed. Although the default value of this variable is *0*, it will always be set to *1* when EASy Installer sources **pkgconfig**.
CONFIGURE_DONE	*0*	The **CONFIGURE_DONE** variable is a boolean that indicates whether or not package configuration has been performed. If your **pkgconfig** code overrides the do_configure target, you should set this variable to *1* after completing the configuration phase.
INSTALL_DONE	*0*	The **INSTALL_DONE** variable is a boolean that indicates whether or not package installation has been performed. If your **pkgconfig** code overrides the do_install target, you should set this variable to *1* after completing the installation phase.
PACKAGE_DONE	*0*	The **PACKAGE_DONE** variable is a boolean that indicates whether or not the installed package creation has been performed. After the package is installed, the necessary metadata is created to facilitate reconfiguration and uninstallation of the package. This phase is called installed package creation. If your **pkgconfig** code overrides the do_package target, you should set this variable to *1* after completing the installed package creation phase.
VERIFY_DONE	*0*	The **VERIFY_DONE** variable is a boolean that indicates whether or not the installed package was verified. After the package is installed, it can be verified to make sure it is properly installed and working. If you override the do_verify target, you should set this variable to 1 after completing the package verification phase.
CLEANUP_DONE	*0*	The **CLEANUP_DONE** variable is a boolean that indicates whether or not the temporary package data has been cleaned up. If your **pkgconfig** code overrides the do_cleanup target, you should set this variable to *1* after completing the cleanup phase.
UNINSTALL_DONE	*0*	The **UNINSTALL_DONE** variable is a boolean that indicates whether or not the uninstallation phase has been performed. Packages cannot override the do_uninstall target, so your code should never set this variable.
MANIFEST	*manifest*	The **MANIFEST** variable is the name of the manifest file containing the list of EEM policies to install. Although this variable can be overridden by your **pkgconfig** code, it is usually not necessary to do so.
MANIFEST_OPT	*manifest.optional*	The **MANIFEST_OPT** variable is the name of the optional manifest file containing a list of EEM policies to be installed but not registered. Policies listed in this file will be unregistered and removed during uninstall time. Adding policies to this file is useful if you wish to conditionally register them during a configure or reconfigure operation.
PKGDESCR	*pkgdescr*	The **PKGDESCR** variable is the name of the package description file containing the full description of your EASy package. Although this variable can be overridden by your **pkgconfig** code, it is usually not necessary to do so.
PKGDB	*${PREFIX}/pkgdb*	The **PKGDB** variable holds the name of the package database that lists what EASy packages have been installed on a device. The value is always a fully qualified path. This variable should not be overridden.
PKGMESSAGE	*pkgmessage*	The **PKGMESSAGE** variable is the name of the package message file. The contents of this file are displayed to the user after the package has been installed. Although this variable can be overridden by your **pkgconfig** code, it is usually not necessary to do so.
PKGUNINSTALL	*pkguninstall*	The **PKGUNINSTALL** variable is the name of the package uninstall file. The contents of this file are Tcl code that will be run prior to uninstalling the package. Although this variable can be overridden by your **pkgconfig** code, it is usually not necessary to do so.
ENVVARS	*envvars*	The **ENVVARS** variable is the name of the file containing the list of environment variables used by your package. Although this variable can be overridden by your **pkgconfig** code, it is usually not necessary to do so.
ENVVAR_VALS	*N/A*	The **ENVVAR_VALS** variable holds a list of environment variables and their values used by your policy. This variable is set in the do_configure target. Therefore, if you override this target, make sure you set **ENVVAR_VALS** appropriately if you require such data.

Variable Name	Default Value	Description
MENU_DONE	*0*	The **MENU_DONE** variable is a control variable that determines when to exit the main EASy Installer menu. If this variable is set to *1*, the EASy Installer will exit. This variable is set to 1 by the do_exit target. If your **pkgconfig** code overrides this target, be sure to set **MENU_DONE** to 1 to exit cleanly from EASy Installer.
SAVEDIR	N/A	The **SAVEDIR** variable holds the name of the current working directory in which the user was prior to running EASy Installer. EASy Installer will take care of changing the directory back to this directory after it finishes its work. You should not override this variable.
CONFIG_CHANGED	*$EASY::FALSE*	The **CONFIG_CHANGED** variable is a boolean that indicates whether the config was changed. If your package provides a function or target that changes the running config, this variable should be set to *$EASY::TRUE*, which instructs EASy Installer to prompt the user to save the running config to startup.

EASY Namespace Variables

The EASY namespace inside the EASy Installer provides a set of variables that can be accessed by your **pkgconfig** and **pkguninstall** code using the **EASY::** calling convention. Table 4 lists these variables.

Table 4. EASY Namespace Variables

Variable Name	Value	Settable?	Allowed Values	Description
EASY::UNTAR	*archive tar /xtract*	No	N/A	The **EASY::UNTAR** variable holds the command used to untar an EASy package. This command must be supported on the given platform in order for EASy Installer to work.
EASY::INFO	N/A	No	N/A	The **EASY::INFO** variable is used as the first argument to the write_error function to specify that the message to print is an informational message.
EASY::WARN	N/A	No	N/A	The **EASY::WARN** variable is used as the first argument to the write_error function to specify that the message to print is a warning message.
EASY::ERROR	N/A	No	N/A	The **EASY::ERROR** variable is used as the first argument to the write_error function to specify that the message to print is an error message. Error messages will cause write_error to return a Tcl error code. These codes must be caught, or they will eventually terminate the EASy Installer.
EASY::FALSE	*0*	No	N/A	The **EASY::FALSE** variable is the boolean value false (or 0). If your code calls for a boolean value of false, you should use this variable.
EASY::TRUE	*1*	No	N/A	The **EASY::TRUE** variable is the boolean value true (or 1). If your code calls for a boolean value of true, you should use this variable.
EASY::DEBUG	*$EASY::FALSE*	Yes	*$EASY::FALSE* *$EASY::TRUE*	The **EASY::DEBUG** variable controls whether or not debugging is enabled for EASy Installer. If set to *$EASY::TRUE*, debugging messages will be printed when EASy installer runs. You can use this boolean in your own code if you need to print debugging messages. This variable is set to *$EASY::TRUE* if EASy Installer is called with the --debug argument.
EASY::VERSION	N/A	No	N/A	The **EASY::VERSION** variable holds the current version of EASy Installer.

EASy Installer Targets

EASy Installer goes about performing its operations through a series of target functions. A **target function** is nothing more than a Tcl proc. The code in **pkgconfig** can override these targets to customize the steps EASy Installer takes to configure and install an EASy package. In the simplest cases, however, it is usually not necessary to override any of the targets, as the default target code performs the steps that are required.

Target functions take no arguments, and they must return an *ok* Tcl code on success and an *error* code on failure. Upon seeing a failure, EASy Installer will notify the user and abort the current session. If this is not desired, you should override the target and catch any potential errors. Overriding a target function is as easy as redeclaring it inside **pkgconfig**. For example, to override the **do_install** target, add the following to **pkgconfig**:

```
proc do_install { } {
    # Custom code goes here.
    if { $error } {
        return -code error "An error occurred"
    }
}
```

Below is a list of the available EASy Installer target functions. All of these can be overridden within **pkgconfig**.

- **post_extract**

  The post_extract target is run immediately following the extraction of the EASy package into a temporary directory. The current working directory is *${PREFIX}/${WRKDIR}.* The **post_extract** target has no default implementation.

- **pre_configure**

  The pre_configure target is run prior to performing any environment variable configuration for the package. It is assumed that the current working directory will be *${PREFIX}/${WRKDIR}* unless changed in **post_extract**. There is no default implementation for **pre_configure**.

- **do_configure**

  The **do_configure** target does the work of configuring environment variables required for the EASy package. It will also record all of these values into the global **ENVVAR_VALS** list. The default implementation of the **do_configure** target expects the current working directory to be *${PREFIX}/${WRKDIR}*, and will fail if that is not the case. If you do not want to use the default code to configure environment variables, you should override this target. However, it is usually sufficient to override **pre_configure** and/or **post_configure** instead.

- **post_configure**

  The **post_configure** target is run just after **do_configure**. The working directory should be *${PREFIX}/${WRKDIR}.* There is no default implementation for **post_configure**.

- **pre_reconfigure**

  The **pre_reconfigure** target is run prior to reconfiguration. The default implementation is simply a wrapper around **pre_configure**. The working directory should be the installed package directory.

- **do_reconfigure**

  The **do_reconfigure** target performs the reconfiguration steps. This target is separate from **do_configure**, in case additional work needs to be done to reconfigure a package. However, the default implementation simply calls **do_configure**. The default working directory should be the installed package directory.

- **post_reconfigure**

  The **post_reconfigure** target is run just after **do_reconfigure**. The default implementation is simply a wrapper around **post_configure**. The working directory should be in the installed package directory.

- **pre_install**

  The **pre_install** target is run just prior to performing the package installation. At this point, no files have been copied to active locations on the device. The current working directory should be *${PREFIX}/${WRKDIR}*. There is no default implementation for **pre_install**.

- **do_install**

  The **do_install** target handles copying the EEM policies included in an EASy package into the EEM user policy directory. If no policy directory has been configured, **do_install** will prompt the user to specify a user policy directory. The **do_install** target is also responsible for registering the package policies with the EEM policy director. The default implementation expects the current working directory to be *${PREFIX}/${WRKDIR}* and will fail if this is not the case. If you do not wish to install EEM Tcl policies in a typical fashion, you should override this function. If you do so, it may also be necessary to override **do_package**.

- **post_install**

  The **post_install** target is run just after **do_install**. The current working directory should be *${PREFIX}/${WRKDIR}*. The **post_install** target has no default implementation.

- **pre_package**

  The **pre_package** target is run just after **post_package** and just prior to installing the various metadata files into their final location. The current working directory should be *${PREFIX}/${WRKDIR}*. The **pre_package** target has no default implementation.

- **do_package**

  The **do_package** target is responsible for building and installing the metadata files needed to manage and uninstall the EASy package. This target will copy these files to the desired installation location. You may need to override this target if you override **do_install**, but this is not recommended. Instead, you should try to create the metadata files as described above, so that **do_package** can do the heavy lifting for you. In fact, in a future release, this target may be made private. The default implementation requires the current working directory to be *${PREFIX}/${WRKDIR}* and will fail if it is not.

- **post_package**

  The **post_package** target is run just after **do_package**. The current working directory should be *${PREFIX}/${WRKDIR}.* There is no default implementation for **post_package**.

- **pre_verify**

  The **pre_verify** target is run prior to package verification. Package verification attempts to confirm that a package has been properly installed. The current working directory should be *${PREFIX}/${WRKDIR}* unless called from the Reconfigure menu, in which case the current working directory should be the installed package directory. There is no default implementation for **pre_verify**.

- **do_verify**

  The **do_verify** target verifies that an installed package is properly installed. The default implementation simply checks to see that the package is properly registered. You might want to override this target (or use **post_verify**) to do additional checks to make sure the package is working properly. The current working directory should be *${PREFIX}/${WRKDIR}* unless called from the Reconfigure menu, in which case the current working directory should be the installed package directory.

- **post_verify**

  The **post_verify** target is called right after package verification. The current working directory should be *${PREFIX}/${WRKDIR}* unless called from the Reconfigure menu, in which case the current working directory should be the installed package directory. There is no default implementation for **post_verify**.

- **pre_cleanup**

  The **pre_cleanup** target is run just before the temporary package data is removed. The working directory should be *${PREFIX}/${WRKDIR}.* There is no default implementation for the **pre_cleanup** target.

- **do_cleanup**

  The **do_cleanup** target is responsible for cleaning up the temporary data created by the extraction of the package. The current working directory should be *${PREFIX}/${WRKDIR}* when **do_cleanup** is called, but it will be changed to *${PREFIX}* during execution. In general, this target should not be overridden.

- **post_cleanup**

  The **post_cleanup** target is called just after **do_cleanup**. The current working directory should be ${PREFIX}. The **post_cleanup** target has no default implementation.

- ~~**do_uninstall**~~

  The **do_uninstall** target is responsible for uninstalling an already installed EASy package. It **cannot** be overridden. It is documented here simply for completeness. If you want to control what this function does, create a **pkguninstall** file in your package. If you do not want the default **do_uninstall** code to run, simply call **return** from your **pkguninstall** code.

- **do_exit**

  The **do_exit** target is used to exit out of the EASy Installer main menu. This is the last target executed before EASy Installer exits. The target prints out a simple exit menu and then sets the **MENU_DONE** global variable. It does not assume any specific working directory.

- **show_descr**

  The **show_descr** target is used to display the contents of the **pkgdescr** file. This target is executed on demand from the EASy Installer menu. It does not assume any specific working directory.

- **show_pkgmsg**

  The **show_pkgmsg** target is used to display the contents of the **pkgmessage** file. This target is run just after installation is performed (after the **post_package** target is executed). It does not assume any default working directory.

EASy Installer Utility Functions

EASy Installer includes some other functions that can be invoked from **pkgconfig** and **pkguninstall**. These functions are not used as targets but serve other purposes, such as doing dependency checking, debugging, getting system information, etc. These functions are listed below.

- **paginate**

Argument Name	Mandatory?	Default Value	Description
l	Yes	N/A	List of strings to display.
enum	No	*$EASY::FALSE*	If *$EASY::TRUE*, the line number is prepended to the front of the line.

  The **paginate** function displays a list of strings on 24-line pages. At the end of each page, the line *Hit enter to continue*... is displayed.

- **wrap_text**

Argument Name	Mandatory?	Default Value	Description
text	Yes	N/A	String of text to wrap. This can be any arbitrary length.
wrap	Yes	N/A	The number of characters at which wrapping will occur. A good recommendation is 75.

  The wrap_text function breaks a string of text up into multiple lines, each of wrap characters (or fewer) in length. The function returns the wrapped text suitable for printing.

- **write_error**

Argument Name	Mandatory?	Default Value	Description
code	Yes	N/A	The message severity code. Allowed values are *$EASY::INFO, $EASY::WARN*, or *$EASY::ERROR*.
msg	Yes	N/A	Message string to display.

  The **write_error** function prints a message to the user with the specified severity code. If the severity code argument is $EASY::ERROR, **write_error** will return a Tcl **error** code.

- **prompt**

Argument Name	Mandatory?	Default Value	Description
msg	Yes	N/A	The message to display to the user.
result	Yes	N/A	Variable in which to hold the response specified by the user.
default	No	"" (empty string)	Default value returned if the user simply presses Enter.

The **prompt** function prompts the user with a specified message and then stores the response in the specified variable. If a default value is provided, **prompt** will return that value if the user presses Enter without typing anything else. The **prompt** function does not return any value.

- **clear_screen**

The **clear_screen** function clears the current vty screen. This function takes no arguments and returns no value.

- **supports_eem1_0**

The **supports_eem1_0** function checks to see if the device supports EEM version 1.0. If it does, **supports_eem1_0** returns $EASY::TRUE; otherwise it returns $EASY::FALSE. The **supports_eem1_0** function takes no arguments.

- **supports_eem2_1**

The **supports_eem2_1** function checks to see if the device supports EEM version 2.1. If it does, **supports_eem2_1** returns $EASY::TRUE; otherwise it returns $EASY::FALSE. The **supports_eem2_1** function takes no arguments.

- **supports_track_ed**

The **supports_track_ed** function checks to see if the device supports the EEM track event detector (ED). While most devices that support EEM version 2.2 support this detector, some do not (such as the Cisco Catalyst® 6500 Series). Therefore, a specific function has been included to perform this check. If the device supports the track ED, **supports_track_ed** returns $EASY::TRUE; otherwise it returns $EASY::FALSE. The **supports_track_ed** function takes no arguments.

- **supports_eem2_4**

The **supports_eem2_4** function checks to see if the device supports EEM version 2.4. If it does, **supports_eem2_4** returns $EASY::TRUE; otherwise it returns $EASY::FALSE. The **supports_eem2_4** function takes no arguments.

- **supports_eem3_0**

The **supports_eem3_0** function checks to see if the device supports EEM version 3.0. If it does, **supports_eem3_0** returns $EASY::TRUE; otherwise it returns $EASY::FALSE. The **supports_eem3_0** function takes no arguments.

- **supports_eem3_1**

  The **supports_eem3_1** function checks to see if the device supports EEM version 3.1. If it does, **supports_eem3_1** returns $EASY::TRUE; otherwise it returns $EASY::FALSE. The **supports_eem3_1** function takes no arguments.

- **supports_eem_version_X**

Argument Name	Mandatory?	Default Value	Description
version	Yes	N/A	An EEM version number in dotted notation (1.0, 2.1, 3.2, etc.).

  The **supports_eem_version_X** function checks to see if the device supports an arbitrary version of EEM. If it does, **supports_eem_version_X** returns $EASY::TRUE; otherwise it returns $EASY::FALSE.

- **is_snmp_enabled**

  The **is_snmp_enabled** function checks to see if the Simple Network Management Protocol (SNMP) manager is enabled on the device. If it is, **is_snmp_enabled** returns *$EASY::TRUE*; otherwise it returns *$EASY::FALSE*. The **is_snmp_enabled** function takes no arguments.

- **get_if_list**

Argument Name	Mandatory?	Default Value	Description
pattern	No	*[^\s]+*	A regular expression specifying which interfaces to return. By default, all available interfaces are returned.

  The **get_if_list** function returns a list of interfaces available on the device. If the optional **pattern** argument is specified, only those interfaces that match the specified regular expression are returned in the list.

- **get_records**

Argument Name	Mandatory?	Default Value	Description
fname	Yes	N/A	The name of the file to open and read.

  The **get_records** function opens the specified file and returns its contents as a list of lines. It automatically trims the lines and removes empty lines.

- **eem_configured**

  The **eem_configured** function checks to see if an EEM user policy directory is configured. If it is, **eem_configured** returns the current user policy directory. Otherwise it returns a Tcl error code.

- **eem_lib_configured**

  The **eem_lib_configured** function checks to see if an EEM user library directory is configured. If it is, **eem_lib_configured** returns the current user library directory. Otherwise it returns a Tcl **error** code.

- **configure_eem**

Argument Name	Mandatory?	Default Value	Description
pdir	Yes	N/A	The EEM policy directory to configure.

The **configure_eem** function configures the requested EEM policy directory as the EEM user policy directory. If the directory does not exist, **configure_eem** creates it. This function returns no value.

- **configure_eem_lib**

Argument Name	Mandatory?	Default Value	Description
ldir	Yes	N/A	The EEM library directory to configure.

The **configure_eem_lib** function configures the requested EEM library directory as the EEM user library directory. If the directory does not exist, **configure_eem_lib** creates it. This function returns no value.

- **deploy_policies**

Argument Name	Mandatory?	Default Value	Description
policies	Yes	N/A	A list of policies to install onto the device.
pdir	Yes	N/A	The policy directory in which to install the policies.

The **deploy_policies** function installs each policy in the **policies** list into the specified EEM user policy directory. Once the policies are copied to the policy directory, **deploy_policies** registers the policies with the EEM policy director. This function returns no value.

- **set_envvar**

Argument Name	Mandatory?	Default Value	Description
var	Yes	N/A	The name of the environment variable to set.
value	Yes	N/A	The value for the environment variable.

The **set_envvar** function sets the specified environment variable to the specified value. This function returns no value.

- **exec_cli**

Argument Name	Mandatory?	Default Value	Description
cmd	Yes	N/A	The CLI command to execute.
no_fatal	No	$EASY::FALSE	If set to $EASY::TRUE, a Cisco IOS error will cause exec_cli to return a Tcl error code.

The **exec_cli** function executes a specified Cisco IOS command-line interface (CLI) EXEC mode command. Upon success, the result of the CLI command (that is, the output) is returned to the caller. If the **no_fatal** argument is $EASY:TRUE, then **exec_cli** returns a Tcl **error** code if a Cisco IOS error is encountered. Otherwise, **exec_cli** returns the empty string if an error is encountered.

- **get_envvar**

Argument Name	Mandatory?	Default Value	Description
var	Yes	N/A	Name of the environment variable to get.

The **get_envvar** function gets the value of the specified EEM environment variable. If the variable is not set, **get_envvar** returns the empty string; otherwise it returns the value of the variable.

- **get_cwd**

The **get_cwd** function returns the current working directory. The working directory value is guaranteed to have a trailing forward slash (/). The **get_cwd** function takes no arguments.

- **easy_mkdir**

Argument Name	Mandatory?	Default Value	Description
dir	Yes	N/A	Name of the directory to be created. If the directory name is not a fully qualified path, the current working directory will be prepended to the directory name.

The **easy_mkdir** function is a wrapper around **mkdir** that makes it easier and more reliable to create a new directory within **tclsh** on a wide variety of Cisco IOS Software versions.

- **easy_copy**

Argument Name	Mandatory?	Default Value	Description
src	Yes	N/A	Name of the source file to copy from. If the source filename is not a fully qualified path, the current working directory will be prepended to the filename.
dest	Yes	N/A	Name of the file or directory to which the source file is copied. If the destination name is not a fully qualified path, the current working directory will be prepended to the destination name.

The **easy_copy** function is a wrapper around **copy** that makes it easier and more reliable to copy files within **tclsh** on a wide variety of Cisco IOS Software versions.

- **is_package_installed**

Argument Name	Mandatory?	Default Value	Description
pname	Yes	N/A	Variable name that holds the package name for which to search in the installed package database. The package name can be specified with or without the **PKGVERSION** value appended.
pvers	No	"" (empty string)	Variable name that will hold the installed package version (if the package specified in **pname** is installed). The contents of this variable may be used by **is_package_installed** in the future.

The **is_package_installed** function checks to see if the specified EASy package is installed. If it is, **is_package_installed** returns *$EASY::TRUE*; otherwise it returns *$EASY::FALSE*. The **pname** argument should be the name of a variable that holds the EASy package name, with or without the version number. If the package is specified with the version number, the package name will be changed in the calling space to be the package name without the version number. For example, consider the following code:

```
set pkgname "green-ports-1.1"
if { [is_package_installed pkgname] } {
    puts "$pkgname is installed"
    # This will print "green-ports" if the package is installed.
}
```

- **file_prompt_quiet**

  The **file_prompt_quiet** function configures the *file prompt quiet* feature on the device if it is not already configured. If *file prompt quiet* is added to the device, the **FILE_PROMPT_QUIET** global variable will be set to *$EASY::TRUE*. If this is the case, it is up to the caller to call **no_file_prompt_quiet** when the feature is no longer required. The **file_prompt_quiet** function takes no arguments and returns no values.

- **no_file_prompt_quiet**

  The **no_file_prompt_quiet** function is the complement to **file_prompt_quiet**. If the **FILE_PROMPT_QUIET** variable is set to *$EASY::TRUE*, then *file prompt quiet* will be unconfigured from the device. The **no_file_prompt_quiet** function takes no arguments and returns no values.

Examples

Example of **pkgconfig**

```
set PKGNAME {custom-mib}
set PKGVERSION {1.1}
set PKGCOMMENT {Make a custom value accessible via SNMP}
set VERSION_CHECK {check_vers}

# Check for minimum requirements.  This package requires a device
# that supports EEM 2.1 at the very least.  The package can use
# either a CLI version or an SNMP version.  If the SNMP version is
# required, then the device must have SNMP configured.
proc check_vers { } {
    global CUSTOM_MIB_MODE

    if { ! [supports_eem2_1] } {
        write_error $EASY::WARN "A device with EEM version 2.1 is required to use
this package"
        return $EASY::FALSE
    }
    set CUSTOM_MIB_MODE "CLI"
    if { [catch {ios_config "snmp mib expression owner __custom-mib_owner name
__custom-mib_name"} result] } {
    if { ! [is_snmp_enabled] } {
            write_error $EASY::WARN "The snmp-server component is required for this
package"
            return $EASY::FALSE
```

```
        }
        set CUSTOM_MIB_MODE "SNMP"
    } else {
        catch {ios_config "no snmp mib expression owner __custom-mib_owner name
__custom-mib_name"} result
    }

    return $EASY::TRUE
}

# Override the pre_configure target to automatically add some extra
# environment variables based on the policy type being deployed
# (either SNMP or CLI).
proc pre_configure { } {
    global CUSTOM_MIB_MODE
    global ENVVARS

    if { ! [info exists CUSTOM_MIB_MODE] } {
        set CUSTOM_MIB_MODE [get_envvar "custom-mib_mode"]
    }

    set fd [open $ENVVARS "a"]

    if { $CUSTOM_MIB_MODE == "SNMP" } {
        puts $fd {name ip_address prompt {Enter a local IP address to poll with
SNMP} pattern {^\d+\.\d+\.\d+\.\d+$} default {}}
        puts $fd {name rw_community prompt {Enter a read-write SNMP community for
this device} pattern {.+} default {private}}
    } else {
        puts $fd {name exp_owner prompt {Enter the owner name for the custom
expression} default {custom-mib_owner} pattern {.+}}
        puts $fd {name exp_name prompt {Enter the name for the custom expression}
default {} pattern {.+}}
    }

    close $fd
}

# After configuration has been done, take the configured environment
# variables in $ENVVAR_VALS, and build a custom pkguninstall file
# to do some custom cleanup on uninstallation.
proc post_configure { } {
    global ENVVAR_VALS
    global PKGUNINSTALL
    global CUSTOM_MIB_MODE

    set owner ""
    set name ""
    set addr ""
    set comm ""
```

```
    foreach row $ENVVAR_VALS {
        array set envvar_val $row
        if { $envvar_val(name) == "exp_owner" } {
            set owner $envvar_val(value)
        } elseif { $envvar_val(name) == "exp_name" } {
            set name $envvar_val(value)
        } elseif { $envvar_val(name) == "ip_address" } {
            set addr $envvar_val(value)
        } elseif { $envvar_val(name) == "rw_community" } {
            set comm $envvar_val(value)
        }
    }

    set fd [open $PKGUNINSTALL "w"]
    puts $fd "ios_config \"no event manager environment custom-mib_mode\""

    if { $CUSTOM_MIB_MODE == "CLI" } {
        puts $fd "ios_config \"no snmp mib expression owner $owner name $name\""
    } else {
        puts $fd "exec_cli \"snmp set v2c $addr $comm oid
1.3.6.1.4.1.9.10.22.1.2.3.1.3.99.117.115.116.111.109.49 integer 6\""
    }

    close $fd
}

# Build a custom manifest file based on the policy type that will
# be deployed (either SNMP or CLI).
proc pre_install { } {
    global MANIFEST
    global CUSTOM_MIB_MODE

    set script "ExpressionMIB_${CUSTOM_MIB_MODE}.tcl"

    set fd [open $MANIFEST "w"]

    puts $fd $script

    close $fd
}

# Put the type of policy being used (either SNMP or CLI) in an EEM
# environment variable.  This will be used for reconfiguration.
# This will be cleaned up in the pkguninstall script.
proc post_install { } {
    global CUSTOM_MIB_MODE

    set_envvar "custom-mib_mode" $CUSTOM_MIB_MODE
}
```

```
proc post_verify { } {
  global CUSTOM_MIB_MODE

  if { ! [info exists CUSTOM_MIB_MODE] } {
    set CUSTOM_MIB_MODE [get_envvar "custom-mib_mode"]
  }

  if { ! [info exists CUSTOM_MIB_MODE] } {
    return -code error "Verification failed: required environment variables have
not been set"
  }

  set output [exec_cli "show event manager policy registered user | include
ExpressionMIB_${CUSTOM_MIB_MODE}.tcl"]
  if { $output == "" } {
    return -code error "Verification failed: required EEM policy has not been
registered"
  }
}
```

Example of **envvars**

```
name countdown_entry prompt {Enter the frequency with which to run the show
command} default {60} pattern {^\d+$}
name match_cmd prompt {Enter the show command to execute} default {} pattern {^show
.*}
name match_pattern prompt {Enter the regular expression to extract the custom
value} default {} pattern {.+}
name nok_msg prompt {Enter message to send via syslog if the expression is found}
default {Expression found} pattern {.+}
```

Example of **pkgdescr**

```
The package is able to extract a value from a show command using a configured
regular expression, and make that value accessible via SNMP using the EXPRESSION-
MIB.
```