

EASy Installer Guide

Introduction

About EASy Installer

The EASy Installer tool is a Tcl script designed to run within the 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 IOS was first introduced in **12.3(2)T** then merged into **12.2(25)S** and **12.2(33)SXH**. However, it is known to be present and functional on other 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 leverage the Embedded Event Manager. EEM was not introduced until **12.3(4)T**.

EASy Installer also requires that the device have at least one flash file system which 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` then EASy packages must be installed manually.

About EASy Packages

EASy packages are nothing more than tar archives. Within the archive, there are multiple files. Some of the files are [meta-data](#) files which tell EASy Installer how it should work. Others are the actual Embedded Event Manager 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 compress (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 will explain how to create EASy packages which can be used by the EASy Installer. It will also discuss 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 in IOS, the EASy Installer can reside on a network file server (e.g. TFTP, SCP, etc.) 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 one 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, EASy Installer can be invoked simply by typing `easy-installer` followed by the appropriate arguments.

Using EASy Installer

EASy Installer can be run without any arguments to get a short usage summary. For example, assuming you have defined an EXEC alias for it (see [above](#)):

```
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
```

More details about each argument can be seen by running 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 'custom-mib-1.1'
-----

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

Enter option:
```

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

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

green-port-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 about the installed package, or remove the package from the device.

To uninstall an EASy package, you can either launch the configure and uninstall menu as described above, or you can 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 which 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:

```
$ 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 meta-data files which drive the EASy Installer. Some of the files can be omitted, but may require additional code to facilitate their function. The meta-data files are described in the following table:

Filename	Mandatory?	Description
<code>pkgconfig</code>	YES	<code>pkgconfig</code> (Example) 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 which is sourced into EASy Installer.
<code>envvars</code>	NO	<p><code>envvars</code> (Example) 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:</p> <pre>condition <i>CONDITION</i> name <i>NAME</i> prompt <i>PROMPT</i> default <i>DEFAULT</i> pattern <i>PATTERN</i> range <i>RANGE</i></pre> <p>Where <i>CONDITION</i> is an optional global variable name. If the value of the variable is <code>\$EASY::FALSE</code> then that environment variable will be skipped during the <code>do_configure</code> phase. However, the variable will still be removed during the <code>do_uninstall</code> phase. The <code>condition</code> element can be omitted entirely. <i>NAME</i> is the name of the variable, <i>PROMPT</i> is the string to use to prompt the user for the variable value, <i>DEFAULT</i> is the default value for the variable if the user simply hits Enter at the prompt, and <i>PATTERN</i> 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 <code>range</code> key instead of <code>pattern</code>. The value of <i>RANGE</i> should then be in the form of <i>MIN-MAX</i>. The <code>pattern</code> and <code>range</code> keys are optional. If omitted any value entered by the user is allowed.</p>
<code>manifest</code>	NO	<code>manifest</code> contains a list of EEM policies to install and register. Each policy file should be put on a line by itself.
<code>manifest.optional</code>	NO	<code>manifest.optional</code> contains a list of EEM policies to install, but not register. Each policy file should be put on a line by itself.
<code>pkgdescr</code>	YES	<code>pkgdescr</code> (Example) contains a description of your package in as many words as necessary. This description is designed to be read by users prior to installing your package. It will be displayed from the EASy Installer menu.
<code>pkgmessage</code>	NO	<code>pkgmessage</code> 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.
<code>pkguninstall</code>	NO	<code>pkguninstall</code> contains Tcl code which will be run prior to 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. However, if there is additional cleanup required, then the `pkguninstall` file should be used.

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 below.

Package-provided variables

The following variables are required to be set in a package's `pkgconfig` file.

Variable Name	Mandatory?	Description
<code>PKGNAME</code>	YES	The <code>PKGNAME</code> is the name of the package. The value should contain letters, numbers, and dashes only . For example, <i>custom-mib</i> , <i>green-ports</i> , or <i>trap2email</i> are valid EASy package names. The name of the package file should be the same as <code>PKGNAME</code> with <code>.tar</code> appended.
<code>PKGVERSION</code>	YES	The <code>PKGVERSION</code> is the version of the package. Valid versions include, <i>1.0</i> , <i>2.1.a</i> , <i>0.71</i> .
<code>PKGCOMMENT</code>	YES	The <code>PKGCOMMENT</code> 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 <code>PKGCOMMENT</code> is displayed when one views the list of installed packages.
<code>MINRAM</code>	NO	The <code>MINRAM</code> value is used when EASy Installer checks the requirements for the package. If specified the device must have at least <code>MINRAM</code> kilobytes of memory in order for the package to install.
<code>MINFLASH</code>	NO	The <code>MINFLASH</code> value is used when EASy Installer checks the requirements for the package. If specified, the device must have at least one flash partition that is at least <code>MINFLASH</code> kilobytes in size in order for the package to install.
<code>VERSION_CHECK</code>	NO	The <code>VERSION_CHECK</code> variable, if specified, must point to a Tcl function name in <code>pkgconfig</code> which 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 minimum IOS or minimum versions of EEM.
<code>OPTIONS</code>	NO	The <code>OPTIONS</code> variable, if specified, is a list of extra items to add to the EASy Intaller menu. The extra items are added just before the Exit item. The format of each item in <code>OPTIONS</code> is a list in itself with the following elements: <i>NAME DEPENDENCIES TARGETS</i> Here, <i>NAME</i> is the name of the menu item that will be displayed to the user, <i>DEPENDENCIES</i> is a list in the format of <i>CHECK_VAR DEP_TARGETS</i> where <i>CHECK_VAR</i> is a variable to check to see whether or

not the targets specified in `DEP_TARGETS` need to be executed; finally, `TARGETS` is a list of [target functions](#) to run to complete the desired tasks for this item.

Global variables

Global variables are those provided by EASy Installer which are available to your code in `pkgconfig`, and can be overridden as needed. The complete list of global variables and their default values can be found in the following table.

Variable Name	Default Value	Description
<code>PREFIX</code>	N/A	The <code>PREFIX</code> 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.
<code>DIST_URL</code>	N/A	The <code>DIST_URL</code> is available only when performing a package installation. It points to the URL of the package tar archive.
<code>WRKDIR</code>	N/A	The <code>WRKDIR</code> points to the location where the package contents can be found prior to installation or when performing a reconfiguration. A package is extracted into <code>WRKDIR</code> then EASy Installer changes directory to <code>WRKDIR</code> . Therefore, you can reference files relative to <code>WRKDIR</code> in your <code>pkgconfig</code> code.
<code>EXTRACT_DONE</code>	0	The <code>EXTRACT_DONE</code> variable is a boolean which indicates whether or not package extraction has been performed. While the default value of this variable is 0, it will always be set to 1 EASy Installer sources <code>pkgconfig</code> .
<code>CONFIGURE_DONE</code>	0	The <code>CONFIGURE_DONE</code> variable is a boolean which indicates whether or not package configuration has been performed. If your <code>pkgconfig</code> code overrides the <code>do_configure</code> target, you should set this variable to 1 after completing the configuration phase.
<code>INSTALL_DONE</code>	0	The <code>INSTALL_DONE</code> variable is a boolean which indicates whether or not package installation has been performed. If your <code>pkgconfig</code> code overrides the <code>do_install</code> target, you should set this variable to 1 after completing the installation phase.
<code>PACKAGE_DONE</code>	0	The <code>PACKAGE_DONE</code> variable is a boolean which indicates whether or not the installed package creation has been performed. After the package is installed, the necessary meta-data is created to facilitate reconfiguration and uninstallation of the package. This phase is called installed package creation. If your <code>pkgconfig</code> code overrides the <code>do_package</code> target, you should set this variable to 1 after completing the installed package creation phase.
<code>VERIFY_DONE</code>	0	The <code>VERIFY_DONE</code> variable is a boolean which 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, working, etc. If you override the

		<p><code>do_verify</code> target, you should set this variable to <code>1</code> after completing the package verification phase.</p>
<code>CLEANUP_DONE</code>	<code>0</code>	<p>The <code>CLEANUP_DONE</code> variable is a boolean which indicates whether or not the temporary package data has been cleaned up. If your <code>pkgconfig</code> code overrides the <code>do_cleanup</code> target, you should set this variable to <code>1</code> after completing the cleanup phase.</p>
<code>UNINSTALL_DONE</code>	<code>0</code>	<p>The <code>UNINSTALL_DONE</code> variable is a boolean which indicates whether or not the uninstallation phase has been performed. Packages cannot override the <code>do_uninstall</code> target, so your code should never set this variable.</p>
<code>MANIFEST</code>	<code>manifest</code>	<p>The <code>MANIFEST</code> variable is the name of the manifest file which contains the list of EEM policies to install. While this variable can be overridden by your <code>pkgconfig</code> code, it is usually not necessary to do so.</p>
<code>MANIFEST_OPT</code>	<code>manifest.optional</code>	<p>The <code>MANIFEST_OPT</code> variable is the name of the optional manifest file which contains 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.</p>
<code>PKGDESCR</code>	<code>pkgdescr</code>	<p>The <code>PKGDESCR</code> variable is the name of the package description file which contains the full description of your EASy package. While this variable can be overridden by your <code>pkgconfig</code> code, it is usually not necessary to do so.</p>
<code>PKGDB</code>	<code>\${PREFIX}/pkgdb</code>	<p>The <code>PKGDB</code> variable holds the name of the package database which lists what EASy packages have been installed on a device. The value is always a fully-qualified path. This variable should not be overridden.</p>
<code>PKGMESSAGE</code>	<code>pkgmessage</code>	<p>The <code>PKGMESSAGE</code> 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. While this variable can be overridden by your <code>pkgconfig</code> code, it is usually not necessary to do so.</p>
<code>PKGUNINSTALL</code>	<code>pkguninstall</code>	<p>The <code>PKGUNINSTALL</code> variable is the name of the package uninstall file. The contents of this file are Tcl code which will be run prior to uninstalling the package. While this variable can be overridden by your <code>pkgconfig</code> code, it is usually not necessary to do so.</p>
<code>ENVVARS</code>	<code>envvars</code>	<p>The <code>ENVVARS</code> variable is the name of the file which contains the list of environment variables used by your package. While this variable can be overridden by your <code>pkgconfig</code> code, it is usually not necessary to do so.</p>
<code>ENVVAR_VALS</code>	N/A	<p>The <code>ENVVAR_VALS</code> variable holds a list of environment variables and their values used by your policy. This variable is set in the <code>do_configure</code> target. Therefore, if you override this target, make sure you set <code>ENVVAR_VALS</code> appropriately if you require such data.</p>

MENU_DONE	0	The <code>MENU_DONE</code> variable is a control variable which determines when to exit the main EASy Installer menu. If this variable is set to <code>1</code> the EASy Installer will exit. This variable is set to <code>1</code> by the <code>do_exit</code> target. If your <code>pkgconfig</code> code overrides this target, be sure to set <code>MENU_DONE</code> to <code>1</code> to exit cleanly from EASy Installer.
SAVEDIR	N/A	The <code>SAVEDIR</code> 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 directory back to this directory after it finishes its work. You should not override this variable.
CONFIG_CHANGED	<code>\$EASY::FALSE</code>	The <code>CONFIG_CHANGED</code> variable is a boolean which indicates whether the config was changed. If your package provides a function or target which changes the running config, this variable should be set to <code>\$EASY::TRUE</code> 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 which can be accessed by your `pkgconfig` and `pkguninstall` code using the `EASY::` calling convention. The list of variables can be found in the table below.

Variable Name	Value	Settable?	Allowed Values	Description
<code>EASY::UNTAR</code>	<code>archive tar /xtract</code>	NO	N/A	The <code>EASY::UNTAR</code> 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.
<code>EASY::INFO</code>	N/A	NO	N/A	The <code>EASY::INFO</code> variable is used as the first argument to the <code>write_error</code> function to specify the message to print is an informational message.
<code>EASY::WARN</code>	N/A	NO	N/A	The <code>EASY::WARN</code> variable is used as the first argument to the <code>write_error</code> function to specify the message to print is a warning message.
<code>EASY::ERROR</code>	N/A	NO	N/A	The <code>EASY::ERROR</code> variable is used as the first argument to the <code>write_error</code> function to specify that the message to print is an error message. Error messages will cause <code>write_error</code> to return a Tcl error code which must be caught, or they will eventually

				terminate the EASy Installer.
<code>EASY::FALSE</code>	<code>0</code>	NO	N/A	The <code>EASY::FALSE</code> variable is the boolean value false (or 0). If your code calls for a boolean value of false, you should use this variable.
<code>EASY::TRUE</code>	<code>1</code>	NO	N/A	The <code>EASY::TRUE</code> variable is the boolean value true (or 1). If your code calls for a boolean value of true, you should use this variable.
<code>EASY::DEBUG</code>	<code>\$EASY::FALSE</code>	YES	<code>\$EASY::FALSE</code> <code>\$EASY::TRUE</code>	The <code>EASY::DEBUG</code> variable controls whether or not debugging is enabled for EASy Installer. If set to <code>\$EASY::TRUE</code> then 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 <code>\$EASY::TRUE</code> if EASy Installer is called with the <code>--debug</code> argument.
<code>EASY::VERSION</code>	N/A	NO	N/A	The <code>EASY::VERSION</code> variable holds the current version of EASy Installer.

EASy Installer Targets

The way EASy Installer goes about performing its operations is 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 does the steps which are required.

Target functions take no arguments, and 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, then `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 meta-data 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 meta-data files needed to manage and uninstall the EASy package. This target will copy these files to the desired installation location. It may be required to override this target if you override `do_install`, but it is not

recommended. Instead, you should try to create the meta-data 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 if a package has been properly installed. The current working directory should be `${PREFIX}/${WRKDIR}` unless called from the Reconfigure menu, then 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. One 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, then 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, then 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 cleaning up 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 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 simply documented here for completeness. If you want to control what this function does, create a `pkguninstall` file in your package. If you do not want to 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, 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. The list of functions can be found below.

- **paginate**

Argument Name	Mandatory?	Default Value	Description
<code>l</code>	YES	N/A	List of strings to display
<code>enum</code>	NO	<code>\$EASY::FALSE</code>	If <code>\$EASY::TRUE</code> , prepend the line number in 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
<code>text</code>	YES	N/A	String of text to wrap. This can be any arbitrary length.
<code>wrap</code>	YES	N/A	The number of characters at which wrapping will be done. A good recommendation is 75.

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

- **write_error**

Argument Name	Mandatory?	Default Value	Description
<code>code</code>	YES	N/A	The message severity code. Allowed values are <code>\$EASY::INFO</code> , <code>\$EASY::WARN</code> , or <code>\$EASY::ERROR</code> .
<code>msg</code>	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` then `write_error` will return a Tcl `error` code.

- **prompt**

Argument Name	Mandatory?	Default Value	Description
<code>msg</code>	YES	N/A	The message to display to the user
<code>result</code>	YES	N/A	Variable in which to hold the response specified by the user

<code>default</code>	NO	"" (empty string)	Default value returned if the user simply hits enter
----------------------	----	-------------------	--

The `prompt` function prompts the user with a specified message, then stores the response in the specified variable. If a default value is provided, then `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 `$EASY::TRUE`, else it returns `$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 `$EASY::TRUE`, else it returns `$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. While most devices that support EEM version 2.2 support this detector, some do not (e.g. Catalyst 6500s). Therefore, a specific function has been included to perform this check. If the device supports the track ED, `supports_track_ed` returns `$EASY::TRUE`, else it returns `$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 `$EASY::TRUE`, else it returns `$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 `$EASY::TRUE`, else it returns `$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 `$EASY::TRUE`, else it returns `$EASY::FALSE`. The `supports_eem3_1` function takes no arguments.

- **supports_eem_version_X**

Argument Name	Mandatory?	Default Value	Description
<code>version</code>	YES	N/A	An EEM version number in dotted notation (e.g. 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 `$EASY::TRUE`, else it returns `$EASY::FALSE`.

- **is_snmp_enabled**

The `is_snmp_enabled` function checks to see if the SNMP manager is enabled on the device. If it is `is_snmp_enabled` returns `$EASY::TRUE`, else it return `$EASY::FALSE`. The `is_snmp_enabled` function takes no arguments.

- **get_if_list**

Argument Name	Mandatory?	Default Value	Description
<code>pattern</code>	NO	<code>[^\s]+</code>	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, then only those interfaces which match the specified regular expression are returned in the list.

- **get_records**

Argument Name	Mandatory?	Default Value	Description
<code>fname</code>	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 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. Else 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. Else it returns a Tcl `error` code.

- **install_pkgIndex**

Argument Name	Mandatory?	Default Value	Description
<code>index</code>	YES	N/A	The index file to install

The `install_pkgIndex` function reads the file specified by `index` and installs the contents into the library `pkgIndex.tcl` file. The EEM library directory must be configured prior to using this function. After calling this function and installing all of the library files, the EEM library directory must be reconfigured by calling the `reconfigure_eem_lib` function.

- **configure_eem**

Argument Name	Mandatory?	Default Value	Description
<code>pdir</code>	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
<code>ldir</code>	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.

- **reconfigure_eem_lib**

The `reconfigure_eem_lib` function refreshes the EEM library files by unconfiguring and reconfiguring the EEM user library directory. This function takes no arguments and will return an error if the EEM user library directory is not already configured.

- **deploy_policies**

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

The `deploy_policies` function installs each policy 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
<code>var</code>	YES	N/A	The name of the environment variable to set
<code>value</code>	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
<code>cmd</code>	YES	N/A	The CLI command to execute
<code>no_fatal</code>	NO	<code>\$EASY::FALSE</code>	If set to <code>\$EASY::TRUE</code> , an IOS error will cause <code>exec_cli</code> to return a Tcl <code>error</code> code

The `exec_cli` function executes a specified IOS cli EXEC mode command. Upon success, the result of the CLI command (i.e. the output) is returned to the caller. If the `no_fatal` argument is `$EASY::TRUE`, then `exec_cli` returns a Tcl `error` code if an IOS error is encountered. Else `exec_cli` returns the empty string on error.

- **get_envvar**

Argument Name	Mandatory?	Default Value	Description
<code>var</code>	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, then `get_envvar` returns the empty string, else 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 '/'. The `get_cwd` function takes no arguments.

- **easy_mkdir**

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

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

- **easy_copy**

Argument Name	Mandatory?	Default Value	Description
<code>src</code>	YES	N/A	Name of the source file to copy from. If the source file name is not a fully-qualified path, the current working directory will be prepended to the file name.
<code>dest</code>	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, then the current working directory will be prepended to the destination name.

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

- **is_package_installed**

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

The `is_package_installed` function checks to see if the specified EASy package is installed. If it is, then `is_package_installed` returns `$EASY::TRUE` else it returns `$EASY::FALSE`. The `pname` argument should be the name of a variable which 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 *file prompt quiet* feature on the device **IF** it is not already configured. If *file prompt quiet* is added to the device, then the `FILE_PROMPT_QUIET` global variable will be set to `$EASY::TRUE`. If this is the case, then it is up to the caller to call `no_file_prompt_quiet` when they no longer require the feature. The `file_prompt_quiet` function takes no arguments, and returns no values.

- **no_file_prompt_quiet**

The `no_file_prompt_quiet` function is the compliment 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 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
# which 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 $fd {name ip_address prompt {Enter a local IP address to poll with SNMP} pattern {^d+\.d+\.d+\.d+$} c
        puts $fd {name rw_community prompt {Enter a read-write SNMP community for this device} pattern {.+} default {f
    } else {
        puts $fd {name exp_owner prompt {Enter the owner name for the custom expression} default {custom-mib_owner} p
        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 inte"
    }

    close $fd
}

# Build a custom manifest file based on the policy type which 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 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 `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.

\$Id: easy-installer.html,v 1.30 2010/01/17 23:27:13 marcus Exp \$