SetupHelper package development

Kevin Windrem kwindrem@icloud.com

This document provides guidance in developing a package suitable for PackageManager management. When this guide is followed, PackageManager should be able to download the package from GitHub or from media inserted in the GX device and install it (bring it into operation under Venus OS).

PackageManager is part of the SetupHelper package. It monitors GitHub for package updates, downloads and installs them automatically (if desired). It also provides a mechanism to add and remove packages from those PackageManager will manage. The user interface for PackageManager is a set of menus in the Venus OS GUI located at Menu / Settings / PackageManager.

SetupHelper includes a set of utilities intended to handle the bulk of package installation and uninstallation and also insures the package is reinstalled. A Venus OS update will restore any modified files to factory defaults, so any modifications must be reinstalled following the Venus OS update.

There are a few basic requirements for a package to be recognized by PackageManager:

1. The package name must conform to unix file naming convention.
   To be most compatible, limit the package name to Upper and lower case alpha characters or numbers. No spaces are allowed.
2. The package must contain a file named ‘version’. Specifics of the version are described later
3. The package should contain a file named ‘setup’. This should be an executable shell script that will install, uninstall or reinstall the package. SetupHelper includes unix bash shell script extensions that must be included in the setup script in order to be properly managed for reinstall after a Venus OS update. More details on the SetupHelper extension later.
4. The package should contain a file named gitHubInfo which should contain the Git Hub user name and default branch separated by : E.g., kwindrem:latest
5. The primary purpose of the setup script is to install modified files to the Venus OS file system to implement the desired functionality change. It is important that the package setup script is also written to **uninstall** the package, restoring any modified files to the factory settings. SetupHelper provides a number of utilities for this.
6. In order for PackageManager to download the package from the internet, it must be contained in a GitHub repository. At a minimum, the tag ‘latest’ must be created that points to the most recent released version. Additional tags matching a version number may be created in order to allow download of specific versions. Tags such as ‘beta’, etc may also be created. PackageManager can also use branch names for specific downloads.
7. The package script is run automatically only if a reinstall is required. Code that needs to run constantly or once each boot should use a service.
8. A package may modify (replace) files in the Venus OS to enhance or change behavior. These replacement files should be stored in “file sets” one for each Venus OS version so that the package can be installed regardless of the Venus OS version currently running. More about file sets later.
9. A package may optionally restrict its operation to a range of Venus OS versions. A package may also restrict its operation to a specific platform. Currently, the only restriction is for RaspberryPi platforms.
10. The package may include a ReadMe file describing the package and how to install it.
11. The package may contain a change log indicating what changed in each version.

SetupHelper

SetupHelper oversees the installation, uninstallation or reinstallation of the package. The package setup script must call ("source") CommonResources, a shell script extension before doing any work to install or uninstall the package:

```
#### following lines incorporate SetupHelper utilities into this script
# Refer to the SetupHelper ReadMe file for details.

source "/data/SetupHelper/CommonResources"

#### end of lines to include SetupHelper
```

CommonResources validates the package for the current Venus OS version and platform, builds a file set for the current Venus OS version if necessary then hands of control to the setup script which then performs modifications to the executable files needed to activate the package.

After all install/uninstall work has been performed, the script should call the 'endScript' function (in CommonResources) in order to exit the script with the proper exit codes. These exit codes are necessary for proper behavior of PackageManager and the automated reinstall operations performed at system boot when the Venus OS is updated.

Setup script command line options

CommonResources checks for optional command line parameters before passing control to the body of the setup script.

reinstall indicates this is to be a reinstall. Installed version is compared to the version in the package directory and skips reinstall if they match. User prompting is skipped. (These version checks are now redundant since **reinstallMods** which controls package reinstalls following a Venus OS update also check versions before calling the package setup script.)

install indicates that the prompts should be skipped and the installation begun with stored options (more about this later). **uninstall** indicates the prompts should be skipped and the package uninstalled.

force overrides the version checks normally performed with **reinstall** and causes an install without user prompting. This option has been depreciated as **install** has the same behavior as **reinstall force**

auto silences all console messages. Progress of the setup is logged. Without the **auto** option progress is also written to the console.

deferReboot instructs endScript to skip the automatic system reboot and return indicating a future reboot is needed.

deferGuiRestart is as above but for automatic GUI restarts.

Without any options, CommonResources assumes a user has run the script and sets up the environment to prompt for user input to control additional execution.

When the setup script regains control following CommonResources, the variable 'scriptAction' will be set to either NONE, INSTALL or UNINSTALL. NONE indicates that the script should prompt for user input to control further execution. These prompts should be skipped if scriptAction is either INSTALL or UNINSTALL. Note that if an install action fails, scriptAction will be set to UNINSTALL, so scriptAction must be tested again after the install section and perform the uninstall if necessary. This prevents a partial install from disrupting the system operation. Note that this behavior is not automatic and must be written into all package setup scripts.

Venus OS is a "dual root fs" system. That is, the operating system and executable parts of the system reside on one of two root partitions. One partition is active and the other inactive. A Venus OS firmware update installs files to the inactive partition. Then when the update has been verified, the active and inactive portions are swapped and the system rebooted to execute the updated code. If the update is unsuccessful, the swap does not occur and the old executable files continue to run. This prevents a partial or corrupted firmware update from bricking the system. A third partition (/data) holds any persistent information (settings, etc).

Packages are stored in the data partition so they survive a Venus OS firmware update, however any system files will be overwritten.

At boot time, the **reinstallMods** script is called. It looks to see of the package is already installed and is the *same* version. If not the package setup script is called with the **reinstall**, **auto**, **deferReboot** and **deferGuiRestart** options. **reinstallMods** will then trigger a system reboot or GUI restart if any package setup scripts have requested those actions.

PackageManager also calls the setup script with **install** or **uninstall** and also **auto**, **deferReboot** and **deferGuiRestart**. PackageManager then acts on requests for system reboot or GUI restarts. PackageManger provides a user choice to defer reboots and GUI restarts when installs/uninsalls are performed manually from there.

SetupHelper utilities

SetupHelper provides a set of utilities to simplify the installation of modified files (called a "replacement") into the active root file system. It pulls the correct replacement file from a "file set" for the current Venus OS version, moves the original out of the way and installs the replacement file. On uninstall, the original file is moved back to the active location (file name) leaving the system in an unmodified state.

updateActiveFile is a function that installs a replacement file as described above. Typically, the replacement file has the same name as the original so the simple form of the command can be used. For example, to replace a GUI file:

```
updateActiveFile "/opt/victronenergy/gui/qml/main.qml"
```

If however, the replacement file has a different file name, a second form is used. This may be necessary if the setup script has to build or modify the replacement file.

```
updateActiveFile " $scriptDir/foo.qml" "/opt/victronenergy/gui/qml/main.qml"
```

restoreActiveFile is a function that undoes the above operation and is called during uninstall with the same file names as were used with updateActiveFile during install:

```
restoreActiveFile "/opt/victronenergy/gui/qml/main.qml"
```

installService is a function that installs and starts a dameon service. The service will be placed in the /service directory (or /opt/victronenergy/service for v2.80 or later). A folder in the package folder named 'service' must contain the files that will end up in /service under the service name

```
installService FooService
```

Will copy the service directory from the package directory into /service/FooService

removeService is a function that removes the service which is necessary during an uninstall to restore the system to factory.

```
removeService FooService
```

logMessage is a function that creates an entry in the SetupHelper log file: /data/log/SetupHelper. Logging is encouraged as it helps debug systems in the field (and while developing the package). Without the **auto** option on the call to the setup script, these messages are also output to the console.

```
logMessage "this text will end up in the SetupHelper log"
```

endScript Function to finish up, prompt the user (if not reinstalling) and exit the script

If **$runningAtBoot** is true (false when CommonResources is sourced) the script will exit with **EXIT_REBOOT** if **$rebootNeeded** is true otherwise, the script will exit with **EXIT_SUCCESS** **endScript** NEVER RETURNS to the caller

If **$runningAtBoot** is false (script was run manually), user interaction controls further action If **$rebootNeeded** is true, the function asks if the user wishes to reboot now. If they respond yes, the system will be rebooted. The user may choose to not reboot now if additional installations need to be done first

If **$rebootNeeded** is false, the function notifies the user of any needed actions

If **$restartGui** is true the gui service will be restarted

The following functions simplify the task of getting user input

standardActionPrompt displays a menu of actions and asks the user to choose

It sets scriptAction accordingly and returns
It also handles displaying setup and package logs then asks for an action again
It also handles quitting with no action - the function *exits the shell script* without returning in this case
The basic action prompt includes install, reinstall, quit, display logs (2 choices)
A reinstall option is enabled if the optionsSet option exists
When reinstall is enabled, selecting install, returns a scriptAction of NONE indicating additional prompting may be needed to complete the install
At the end of these prompts, the main script should set scriptAction to INSTALL
If reinstall is selected, the script action is set to INSTALL and the main script should then skip additional prompts and allow options set previously to control the install

yesNoPrompt "prompt "

Asks the user to answer yes or no to the question
Any details regarding the question should be output before calling yesNoPrompt

yesNoPrompt sets **$yesResponse** to true if the answer was yes and false if the answer was no

A set of utilities manages dbus Settings: creating, removing, updating. It is sometimes necessary for the setup script to create dbus Settings so GUI has access to them. This is often the case when the package doesn't run its own service.

The following functions will create dbus settings if they do not already exist or update their value if they do

```
updateDbusStringSetting "/Settings/StringSetting" "the new string"
updateDbusIntSetting "/Settings/IntegerSetting" 5
updateDbusRealSetting "/Settings/FloatingPointSetting" 6.0
```

setSetting is a function that updates the value of an existing dbus Setting. It is faster than calling one of the above update… functions. The new value can be any data type but strings must be quoted.

```
setSetting "/Settings/foo" "new string"
setSetting "/Settings/bar" 18
```

The following function removes the settings. Limit the number of settings to about 20 to avoid some being missed (not sure why). It is faster to remove multiple settings at the same time than it is to call 'removeDbusSettings' for each one.

```
removeDbusSettings "/Settings/foo" "/Settings/bar"
```

SetupHelper variables

SetupHelper manages or tests a set of variables that control script executions:

$scriptAction provides direction for the setup script and has the following values:

NONE - setup script should prompt the user for the desired action and set scriptAction accordingly
EXIT - the setup script should exit immediately
INSTALL - the setup script should execute code to install the package
UNINSTALL - the setup script should execute the code to restore the Venus files to stock If installation errors occur within functions in CommonResources, scriptAction will be changed to UNINSTALL.

The setup script MUST retest scriptAction after all installation code has been executed so the package can be removed, rather than leaving the system with a partially installed package.

$rebootNeeded - true signifies a reboot is required after the script is run. The setup script should set **rebootNeeded** to true if a reboot is needed following install/uninstall

The following variables contain useful information but should not be changed by the setup script:

$scriptDir - the full path name to the startup script the script's code can use this to identify the location of files stored in the package

$scriptName - the basename of the setup script ("setup")

$reinstallScriptsList - the file containing a list of scripts to be run at boot to reinstall packages

after a Venus software update (by default, this is /data/reinstallScriptsList)

$installedVersionFile - the name of the installed version file

$venusVersion - the version of VenusOS derived from /opt/victronenergy/version

$fileSets - the standard location for the replacement files

$pkgFileSets - is the locaiton of version-dependent files for the current Venus version

$runningAtBoot - true if the script was called from reinstallMods (at boot time)
signifying this is to be an unattended (automatic) installation
CommonResoures sets this variable based on command line options

$setupOptionsDir - the location of any files that control installation
These options are maintained in a separate directory so reinsalling the package does not remove them so that a reinstall can proceed without prompting again

$obsoleteVersion - prevents installation starting with this Venus OS version

$firstCompatibleVersion - prevents installation *before* with this Venus OS version

File sets

It is often necessary to create a replacement file for different Venus OS versions. This allows the package to be installed regardless of the Venus version. These different replacement file versions are contained in a "file set": a directory with the version number as it's name. The collection of file sets is stored in the 'FileSets' directory in the package directory.

The fileList file in the FileSets directory is a list of each *version-dependent* file in the package.

Some files in a package may not be tied to specific Venus OS versions. For example, if a service is added to the system, it will most likely be the same for all versions of Venus OS. These files would not be located in a version file set but rather in the FileSets directory in the package and NOT included in the fileList.

In order to identify changed files, the original file for each replacement must be compared against ALL versions of Venus OS. When changes are detected, a new file set version directory needs to be created and a new ...orig file copied from Venus OS. This process runs on the computer managing the package, not on the GX device.

In order to run the necessary checks, Venus OS versions need to be available to the managing computer. While not all Venus OS versions will need file sets, it won't be clear which ones will need replacement files, so my recommendation is to start with the latest version of each major release and all versions of the current release and all versions of the current beta cycle: v2.66, v2.27, v2.80, v2.81, … v2.89, v2.90~12, v2.90~14, v2.90~18, v2.90~20, v2.90~22, v2.90~24, v2.90~26. You can then weed out versions you don't need.

I use the raspberry Pi images from http://updates.victronenergy.com/feeds/venus/ because these contain the compete file system. Alternatively, the file system can be copied from a running GX device ***prior*** to installing any packages. I create a directory on the managing computer called OriginalFiles, then create a directory for each Venus OS version: v2.81, v2.90~12, etc.

Next, I copy the /etc, /opt and /var/www/venus/styling directories from the Venus OS image to the OriginalFiles/vX.Y~Z directory.
I limit OriginalFiles to these directories to minimize storage space on the system used to generate the package files.
99% of the files likely to be modified by a package are located there.
This is an artificial limit and other parts of the file system may be included if needed.

The **updateFileSets** script included in the SetupHelper can be used to create file sets for each package during development. It runs through all StockFiles version directories and all package directories and creates file sets in each package. The comments at the top of the **updateFileSets** file provide additional details. Note that this is a unix bash script and should run on all unix and Mac OS computers. Windows will not run this script natively. However Windows 10 apparently supports bash:

https://www.howtogeek.com/249966/how-to-install-and-use-the-linux-bash-shell-on-windows-10/

Before running this script, you need to edit the FileSets/fileList file to include the files your package will modify. Use full path names to avoid issues.

Update: File sets created with SetupHelper v.5's **updateFileSets** script will contain ALL replacement files (or symbolic links to an identical replacement in another file set) in every file set. This change prevents a problem where a matching original file could not be found even though a file set does exist for the current Venus OS version. This resulted in an "incomplete file set" error and failure to install the package. To clear the error, it was necessary to reinstall the package and/or the Venus OS firmware.

Update: **_checkFileSets** in CommonResources now checks for existing replacement files in a file set before looking for original files and a match to the installed file. With the above change, the replacement file (or symbolic link) will generally be valid and no search will be necessary. (In previous versions of SetupHelper, a test was made for original files *before* looking for a replacement file. This causes the file set to be invalidated if no original file exists!)

File sets created with **updateFileSets** from an older version of SetupHelper only contain replacement files when the original file changes between versions. Also, the package may not contain a file set for the current Venus OS version. **_checkFileSets** will attempt to fill in missing files or build a missing file set by comparing the original files in other file sets with the active file installed by Venus OS. If a match is found, this means the replacement file from the other file set also applies to the current Venus OS version.

After running this script, you will find empty file sets populated with ...NO_REPLACEMENT files. These indicate where you need to create replacement files for your packages. You will need to add your changes to each replacement file in each file set.

Naming:

the replacement file has the extension of the actual file, e.g., PageMain.qml
the original file adds a .orig extension, e.g, PageMain.qml.orig
if no original exists, an empty file with the .NO_ORIG file will be created,
e.g., PageMain.qml.NO_ORIG

Existence of a .NO_ORIG file after running updateFileSets indicates a significant problem. What this says is that the replacement file has no equivalent in a stock system. If the replacement file is the same for all Venus OS versions, simply remove it from fileList and place the replacement in the FileSets directory, not in a version directory.

However, if the replacement file differs between Venus OS versions, an alternate original file needs to be used as a reference. For example, if you are creating a new file PageMainEnhanced.qml, then you can probably use PageMain.qml as the alternate original. Create a file in FileSets named PageMainEnhanced.qml.ALT_ORIG with a single line with the full path to the alternate original:

```
/opt/ victronenergy/qui/qml/PageMain.qml
```

Sometimes, a replacement file is needed in SOME versions of Venus OS but in others. An empty file in the file set will instruct SetupHelper to use the orig, e.g., PageMain.qml.USE_ORIGINAL

Version file

A package must contain a version file. This is the *package* version, not the Venus OS version. The package version is used by PackageManager to decide if an automatic download is needed by comparing the version from GitHub with the version stored on the system. Likewise, the stored version is compared to the installed version to trigger an automatic install.

The version file is a text file with a single line of the form: v1.2, v1.2~3 v1.2a3.

Versions that include a ~ or lettered version are treated as pre-release.

'd' represents a development release
'a' represents an alpha release 'b' or '~'
represents a beta release none of the above
represents a released version

Version numbers are prioritized: 'a' is "newer" than 'd', etc.

"newer" versions will replace older versions when automatically downloading. Exception: if the branch/tag set in PackageManger is a specific version (e.g., v.4.6) the stored version must match rather than being older than. Installs always occur if the versions do not match.

Restricted install

The package may optionally contain files that place restrictions on which Venus OS versions or platforms the package may be installed on. If any of these tests fail, the install will also fail!

If present **obsoleteVersion** identifies the first version that are not compatible with this package. E.g., of obsoleteVersion contains v7.2 and the current Venus OS version if v8.0, then the package can't be installed.

If present **firstCompatibleVersion** identifies the first version that IS compatible with this package. E.g., if firstCompatibleVersion contains v8.0 and the current Venus OS version is v7.2, the package can't be installed.

Note that if both **firstCompatibleVersion** and **obsoleteVersion** are included in the package directory, the obsoleteVersion must be greater than firstCompatibleVersion.

If the file **raspberryPiOnly** exists in the package directory, the platform (aka 'machine') MUST be raspberrypi2 or raspberrypi4. If not, installation will fail.

endScript

The **endScript** function must be called at the end of the setup script. This function tests shell variables to determine the next corse of action. Course of action also depends on whether the script is being run by a human from the command line or autonomously from PackageManager or the reinstall mechanism.

Many of the scripts modify the GUI files, so the GUI needs to be restarted following install or uninstall. SetupHelper can't make this determination, so if this is the case the setup script must set the **restartGui** shell variable to true prior to calling endScript

Likewise, the GX device may need to be rebooted following install/uninstall. setting **rebootNeeded** to true will insure that the GX device is rebooted.

If the setup script is run from the command line (no command line options), **endScript** will prompt the user for a reboot or GUI restart if one is needed. The user can choose to trigger the action now or wait and do it manually later.

However, if the script is running autonomously, the action will be triggered from within endScript, *unless* the script was run with **deferrReboot** or **deferGuiRestart** on the command line. In this case, the action is not performed but the script exits with the appropriate exit code. PackageManager and **renstallMods** use the exit code to choose a course of action following all automatic operations. For manual install/uninstall from PackageManger, the user is given the choice to perform the GUI restart or reboot now or do it later. If deferred, a message will be displayed indicating the package isn't fully active ("reboot needed").

For reinstalls following a Venus OS update, **reinstallMods** will reboot the system or restart the GUI after all setup scripts have been run and a reboot/reset has been requested by any script. If a reboot is needed, the GUI will not be restarted even if needed since the reboot will do that.

When the setup script completes an install operation, **endScript** writes the package version to a file in /etc/venus. **endScript** deletes this file during an uninstall. The installed version file written to /etc/venus tells PackageManger which version of each package is installed and running. It also tells the reinstall mechanism to skip the package install. So reinstall only happens if the installed version file is NOT present or the installed version differs from the package version itself. The latter may be the case if a Venus OS firmware update has occurred.

Some packages may need to reboot in the middle of the installation process. For example, if an overlay is needed to test for a specific condition, the setup script should install the overly, but skip the remaining setup, then set the **runAgain** shell variable before calling endScript. endScript then removes the installed version file so the next boot will run the package's setup script again.