PackageTools Documentation
Version 1.2

Jeffrey J Weimer

September 26, 2010

1 Introduction

The Igor Pro package PackageTools is designed for programmers to have a consistent way to manage packages that are developed for general use. It provides functions to set up, get help for, and remove a package. It also includes functions that will show a list or log of installed packages.

2 Benefits

If you are a general user of Igor Pro (rather than a package developer), your benefits from installing this package are that you will gain administrative tools to oversee what packages have been set up in your experiment. When a programmer designs a package to follow the conventions of PackageTools, certain things will happen automatically. First, ever time you install a compliant package, you will get a print out in the history window showing that package has been set up. Secondly, you will be able to use functions on the command line to show a log or list of installed packages and get help for any of them.

Package developers are encouraged to make use of the functionalities provided within PackageTools to help their users administer their packages with a consistent framework of commands.

3 Installation

This package requires at least Igor Pro version 6.20.

Install this package by putting its folder into the Igor Procedures folder. This will assure that it is always available for every experiment. Never put this into the User Procedures folder and use an #include statement to install this package as part of an experiment. Doing so will lead to unpredictable results.

4 Functions

PackageTools contains static (restricted) and non-static (common) functions.

4.1 Restricted (Static) Functions

The static functions are specifically designed for users (rather than programmers) who have installed PackageTools. They return information about the packages that have been set up in compliance with the conventions in PackageTools. To access the functions below, you must preface the command with the noun PackageTools#.

ShowLog()

This functions shows a notebook page that contains a log of when packages where setup, updated, or removed using PackageTools.

ListPackages()

This functions shows a notebook page that lists all information given for all packages that have been set up using PackageTools.

PackageKey(name, key)

This string functions returns the *key* value for the package named *name*.

Example:
PackageTools#PackageKey("PackageTools","author") - returns my name

---

4.2 Common (Non-Static) Functions

The non-static functions can be called directly from the command line or within a function. They are really designed for programmers, not for a general user of PackageTools. The general exception is the PackageHelp function, as described below.

PackageSetup(name, [folder, version, info, author, file, hasHelp, removable, more, removable, quiet])

This function is for programmers to setup a package using PackageTools.

name string name of the package

folder optional string denoting the location of the package folder
default: root:Package:PackageName

version optional floating point variable of the package version number

info optional short string describing the package (shown in place of help)

author optional string name of author

file optional string name of procedure file (including extension)

hasHelp optional variable denoting help file exists (1) or does not exist (0)
default: missing value is the same as having no help

more optional string list of additional *keyword:value* information for the package

removable optional variable stating whether package is allowed to be removed (1) or not (0)
default: missing value says the package cannot be removed (*removable = 0*)

quiet optional variable to print information in history (0) or not (1)
default: missing value is the same as *quiet = 0*

This function does only one thing. It writes all the information that is provided to it into a *key* string that is stored with the experiment. The *key* string is stored in a data folder within the experiment at root:Packages:PackageTools:Keys and has the name of the package. Confirmation is given in the history window and log of when a package is set up.

Example:

```
PackageSetup("PackageTools",version=1.2,author="Jeffrey J Weimer", more="email:jjw@usa.world")
```

PackageUpdate(name, [folder, version, info, author, file, hasHelp, removable, more, removable, quiet])

This function is for programmers to update a package using PackageTools.

name string name of the package

folder optional string denoting the location of the package folder
default: root:Package:PackageName

version optional floating point variable of the package version number

info optional short string describing the package (shown in place of help)

author optional string name of author

file optional string name of procedure file (including extension)

hasHelp optional variable denoting help file exists (1) or does not exist (0)
default: missing value is the same as having no help

more optional string list of additional *keyword:value* information for the package

removable optional variable stating whether package is allowed to be removed (1) or not (0)
default: missing value says the package cannot be removed (*removable = 0*)

quiet optional variable to print information in history (0) or not (1)
default: missing value is the same as *quiet = 0*

This function does two things. First, it checks that any of the information that is provided to is different from what is already stored. Then, if it is different, it writes the new information into the *key* string that is stored with the experiment. Confirmation is given in the history window and log of when a package is updated.

Example:
PackageUpdate("PackageTools",version=1.3,info="A package to manage packages")

PackageHelp(name, [key])

This function is for programmers to show help for their package using additional features in PackageTools. When you have installed PackageTools in your experiment, you can also type this command on the command line to get help on any package that follows the conventions of PackageTools.

name string name of the package

key optional string with one of the keys *folder*, *version*, *info*, *author*, *hasHelp*, or *removable*
additional keys given with *more* may also be accessed this way

When key is not given and when hasHelp = 1 for the package, this function will show the help file for the package. In this way, it is equivalent to the Igor Pro command DisplayHelpTopic. However, when a help file does not exist or when hasHelp = 0, the info string for the package is shown. Finally, when the help file and info string both do not exist, a notice is also given that no help is available.

Example:
PackageHelp("PackageTools") - shows help

Example:
PackageHelp("PackageTools", key="email") - shows email (if it exists for this package)

PackageRemove(name, [quiet])

This function is for programmers to remove a package using PackageTools.

name string name of the package

quiet optional variable to tell whether to print to history (0) or not (1)
default: missing value is same as printing to history (*quiet = 0*)

A package is removed in the following way:

- First, a check is done whether the package has been set up using PackageTools. If not, nothing is done.
- Next, a check is done if removable exists for the package. If not, nothing is done.
- Next, a check is done if removable = 0. If so, nothing is done.
- At this point, validation is requested to remove the package. If removing the package is permitted, the function then continues.
- First, it kills the package folder (as set by folder key in the PackageSetup(...) function).
- Then, it deletes any #include statements within the main procedure window that reference to the procedure file for the package (as set by file key in the PackageSetup(...) function).
- Finally, it closes and removes the procedure file for the package (as set by file key in the PackageSetup(...) function).

When quiet = 0 (default), this function will request conformation from a user to remove a given package. Otherwise it proceeds without confirmation to try to remove the package using the above steps. Confirmation is given in the history window and in the log at the end of successfully removing a package,

Example:
PackageRemove("PackageTools") - shows warning because this package has *removable = 0*

PackageExists(name)

This function returns 0 if the package has not been set up with PackageTools or 1 otherwise. It also sets a global V_exists in the current data folder to the same value.

---

5 For Programmers

5.1 Introduction

You may have a package that will make use functions in PackageTools and wish to require that your users install it along with your package. Please be sure to make your users aware of exactly what is needed in this case. You should thoroughly check the installation on your own system before distributing instructions to your users.

You may instead want to use functions in PackageTools but be able to handle those cases where your users do not install PackageTools or install it incorrectly. I call this mode of use "benefiting by but not requiring PackageTools" for operation. In such cases, when you are going to call any of the functions in PackageTools, you should do your own internal error checking after the function call and/or protect the function calls by using an Execute/Q/Z statement.

When you want your package to comply with PackageTools, you should do two things. First, you should establish the proper header portion of your procedure file. Then, you should set up a proper function that will be run by PackageTools after Igor Pro compiles.

5.2 Example Header for a Procedure File

This shows the header portion of the procedure file for PackageTools. See also the example that is distributed with the package. This portion of code belongs at the top of your procedure file. Make changes to the quoted strings and variables to fit your procedure file.

```
// pragmas for the procedure
// (see the Igor Pro manual for details)

#pragma rtGlobals=1
#pragma IgorVersion=6.20
#pragma version=1.2
#pragma hide=1

// you may or may not have a module or independent module name for your procedure

#pragma ModuleName=PackageTools

// package parameters to work with PackageTools
// define whichever of these you need for your package

Static StrConstant thePackage="PackageTools"
Static StrConstant thePackageFolder="root:Packages:PackageTools"
Static StrConstant theProcedureFile = "PackageTools.ipf"
Static StrConstant thePackageInfo = "Tools to manage packages"
Static StrConstant thePackageAuthor = "Jeffrey J Weimer"
Static Constant thePackageVersion = 1.2
Static Constant hasHelp = 0
Static Constant removable = 0

// here is one way to define other PackageTools parameters

Static StrConstant more = "email:jjw@usa.world; phone:888-888-8888;techsupport:NONE;"
```

5.3 Setting Up or Updating Your Package

As a programmer, you have a number of choices that you can use to tap into the set up or update functions using PackageTools.

5.3.1 Calling the Routines Directly

You are free to call any of the functions in PackageTools at any point from any place in your procedure. For example, when you have a package that installs a panel for the user as the first step, you might call the function PackageSetup(...) or PackageUpdate(...) just before or just after the panel is displayed. Any time you want to display a help file or the information about your package, you can call the PackageHelp(...) function. Finally, anytime you want to assure that your package is removed properly, you can call the PackageRemove(...) function.

5.3.2 Using a PTAfterCompile_ Hook Function

After Igor Pro compiles, PackageTools is set to run all functions that have the form shown below.

```
Function PTAfterCompile_Name()
```

In the function name, the only part that must be different is the *Name* portion. This is a unique (short) name that you give to distinguish your function from every other function. You may want for example to make *Name* be an abbreviation for your package, such as

```
Function PTAfterCompile_MyFitRoutine()
```

Be aware of three things in setting up your function. First, the name must be unique to all possible variations that all possible users of PackageTools may eventually have. Secondly, the entire function name (including the prefix) must be shorter than the 31 character limit established in Igor Pro. Finally, the function itself must not be a Static function (it must be visible in scope when encased in a module or an independent module).

The example code that follows shows a function to run after compiling that will either set up or update your package parameters using PackageTools. It presumes that you have set up a header file in your procedure using the terminology given in the previous section. The code is specifically designed to run correctly regardless of how a user decides to include your procedure file in the experiment. In particular, this code will run when a) a user opens the procedure file and then compiles Igor Pro, b) uses #include to include your procedure file from the User Procedures or Igor Procedures folder, or c) has a #include statement within his or her own procedure file to include your procedure file, even when the originating procedure file is a module or independent module. The code is also designed to return without executing an initialization if the PackageTools package is not installed properly or has not yet been initialized. The code also presumes that your procedure may be in a module named MyModuleName. Otherwise set that portion to be blank.

Note: Using an #include within an independent module to include another independent module will never work properly, as the Igor Pro manual clearly states. This case is therefore never considered in the outline of the code given.

```
// the hook function called by PackageTools after a compile
Function PTAfterCompile_MyPackage()
        string theCmd
        string GIMN = GetIndependentModuleName()
        // set myModuleName to ""  when not using a ModuleName pragma
        // (such as when this code is in an IndependentModule)
        // or remove the line and modify the the command after it appropriately
        string myModuleName = "MyModuleName#"
        sprintf theCmd, "%sInitialize()", myModuleName
        if (strlen(GIMN)!=0)
                sprintf theCmd, "%s#%s", GIMN, theCmd
        endif
        Execute/Q/Z theCmd
        return 0
End

// the initialization function called by the hook function
// the static designation is not needed when this function is in an IndependentModule
Static Function Initialize()
        string theCmd
        sprintf theCmd, "ProcGlobal#PackageExists(\"%s\")", thePackage
        Execute/Q/Z theCmd
        NVAR V_exists
        if (NVAR_exists(V_exists))
                sprintf theCmd,"\"%s\"", thePackage
                sprintf theCmd, "%s,folder=\"%s\"", theCmd, thePackageFolder
                sprintf theCmd, "%s,file=\"%s\"", theCmd, theProcedureFile
                sprintf theCmd, "%s,info=\"%s\"", theCmd, thePackageInfo
                sprintf theCmd "%s,author=\"%s\"", theCmd, thePackageAuthor
                sprintf theCmd, "%s,version=%f", theCmd, thePackageVersion
                sprintf theCmd, "%s,hasHelp=%d", theCmd, hasHelp
                switch(V_exists)
                        case 0:
                                sprintf theCmd, "ProcGlobal#PackageSetup(%s)", theCmd
                                break
                        case 1:
                                sprintf theCmd, "ProcGlobal#PackageUpdate(%s)", theCmd
                                break
                endswitch
                Execute/Q/Z theCmd
        endif
        return 0
End
```

5.3.3 General Guidelines

- Setting up a package using PackageTools is not the same as creating a specific Package folder for it. That is the responsibility of your package to do.
- As best possible, you should have only one place where your procedure file calls on the functionality of PackageTools. Especially avoid designing duplicate ways to call the setup or update functions at more than one point in your procedure file. Some further suggestions on this point are as follows:
  - When your package creates its own Package folder in an experiment as part of its own internal initialization process, you should call PackageSetup(...) or PackageUpdate(...) directly afterward to have PackageTools record it in its log.
  - As a general rule, you should use the PTAfterCompile_ hook function method to set up your package with PackageTools only when your package does not create its own Package folder as part of its initialization process. Otherwise, use the method suggested above.
  - You may consider writing your own AfterCompiledHook() function as provided by Igor Pro and call the setup routines in PackageTools from within that function. When you do this, do not also put a PTAfterCompile_ hook function in your procedure file. No danger is expected, however no guarantee is given that no problems will ever arise by having two redundant ways to handle hooks after compile.
- Calls to PackageSetup or PackageUpdate are supposed to return as directly as possible. When the *key* for a package already exists, a call to PackageSetup will return immediately with no changes made. Calls to PackageUpdate check for the existence of every possible (optional) *key* but update only those that are given.
- Checking for the existence of V_exist after calling PackageExists confirms whether PackageTools has or has not been installed properly. When PackageTools is installed properly, V_exists will exist. This check will work even with a blank package name, as in PackageExists("").
- Checking for the value of V_exist after calling PackageExists only confirms that PackageTools has or has not set up the package. It does not always confirm that a package has been set up properly by its own internal initialization process.