PackageManager

Easily download and install GAP packages

0.2.3

22 February 2019

Michael Torpey

Michael Torpey
Email: mct25@st-andrews.ac.uk
Homepage: http://www-groups.mcs.st-andrews.ac.uk/~mct25/
Address: School of Computer Science
University of St Andrews
Jack Cole Building, North Haugh
St Andrews, Fife, KY16 9SX
United Kingdom

Contents

Chapter 1

Introduction

1.1 What does the PackageManager package do?

This package provides the ability to install or remove a package using a single command: `InstallPackage` (2.1.1) or `RemovePackage` (2.2.1). The user can specify a package to install using its name, or using a URL to an archive, a repository, or a `PackageInfo.g` file. When installing, PackageManager also attempts to compile the package, and install the latest version of any required packages that are not already available.

1.2 What does the PackageManager package not do?

At present, PackageManager is fairly basic, without many of the advanced features available in package managers such as pip or apt. For instance, the user cannot install a particular version of a package, nor automatically update all packages. Removing a package will not remove any of its dependencies, since we do not track how packages were installed. When a package is installed, no tests are run to ensure that it is compatible with the installed version of GAP. Any of these features might be added in the future. Other feature requests can be posted on the issue tracker at https://github.com/gap-packages/PackageManager/issues.

1.3 A quick example

To install the latest deposited version of the Digraphs packages, use the following:

Example

```
gap> InstallPackage("digraphs");
```

To uninstall it later, use the following:

Example

```
gap> RemovePackage("digraphs");
```

PackageManager also supports version control repositories. To install the latest version of the curlInterface package from GitHub, use the following:

Example

```
gap> InstallPackage("https://github.com/gap-packages/curlInterface.git");
```

Chapter 2

Commands

2.1 Installing packages

2.1.1 InstallPackage

▷ `InstallPackage(`*`string[, interactive]`*`)` (function)

Returns: true or false

Attempts to download and install a package. The argument *`string`* should be a string containing one of the following:

- the name of a package;
- the URL of a package archive, ending in `.tar.gz`;
- the URL of a git repository, ending in `.git`;
- the URL of a mercurial repository, ending in `.hg`;
- the URL of a valid `PackageInfo.g` file.

The package will then be downloaded and installed, along with any additional packages that are required in order for it to be loaded. If this installation is successful, `true` is returned; otherwise, `false` is returned.

By default, packages will be installed in user's home directory at `~/.gap/pkg`. Note that this location is not the default user pkg location on Mac OSX, but it will be created on any system if not already present. Note also that starting GAP with the `-r` flag will cause all packages in this directory to be ignored.

Certain decisions, such as installing newer versions of packages, will be confirmed by the user via an interactive shell - to avoid this interactivity and use sane defaults instead, the optional second argument *`interactive`* can be set to `false`.

To see more information about this process while it is ongoing, see `InfoPackageManager`.

Example

```
gap> InstallPackage("digraphs");
true
```

2.1.2 InfoPackageManager

▷ `InfoPackageManager` (info class)

Info class for the PackageManager package. Set this to the following levels for different levels of information:

- 0 - No messages
- 1 - Problems only: messages describing what went wrong, with no messages if an operation is successful
- 2 - Problems and directories: also displays directories that were used for package installation or removal
- 3 - Progress: also shows step-by-step progress of operations
- 4 - All: includes extra information such as whether curlInterface is being used

Set this using, for example `SetInfoLevel(InfoPackageManager, 1)`. Default value is 3.

2.1.3 InstallPackageFromName

▷ `InstallPackageFromName(`*`name[, interactive]`*`)` (function)

Returns: true or false

Attempts to download and install a package given only its name. Returns `true` if the installation was successful, and `false` otherwise.

Certain decisions, such as installing newer versions of packages, will be confirmed by the user via an interactive shell - to avoid this interactivity and use sane defaults instead, the optional second argument *`interactive`* can be set to `false`.

2.1.4 InstallPackageFromInfo

▷ `InstallPackageFromInfo(`*`info`*`)` (function)

Returns: true or false

Attempts to download and install a package from a valid PackageInfo.g file. The argument *`info`* should be either a valid package info record, or a URL that points to a valid PackageInfo.g file. Returns `true` if the installation was successful, and `false` otherwise.

2.1.5 InstallPackageFromArchive

▷ `InstallPackageFromArchive(`*`url`*`)` (function)

Returns: true or false

Attempts to download and install a package from an archive located at the given URL. Returns `true` if the installation was successful, and `false` otherwise.

2.1.6 InstallPackageFromGit

▷ `InstallPackageFromGit(`*`url`*`)` (function)

Returns: true or false

Attempts to download and install a package from a git repository located at the given URL. Returns `true` if the installation was successful, and `false` otherwise.

2.1.7 InstallPackageFromHg

▷ `InstallPackageFromHg(`*`url`*`)` (function)

Returns: true or false

Attempts to download and install a package from a Mercurial repository located at the given URL. Returns `true` if the installation was successful, and `false` otherwise.

2.2 Removing packages

2.2.1 RemovePackage

▷ `RemovePackage(`*`name[, interactive]`*`)` (function)

Returns: true or false

Attempts to remove an installed package using its name. The first argument *`name`* should be a string specifying the name of a package installed in the user GAP root. The second argument *`interactive`* is optional, and should be a boolean specifying whether to confirm interactively before any directories are deleted (default value `true`).

Returns `true` if the removal was successful, and `false` otherwise.

Example

```
gap> RemovePackage("digraphs");
Really delete directory /home/user/.gap/pkg/digraphs-0.13.0 ? [y/N] y
true
```

Index