Package Management for System i

Concept (Draft)

Mihael Schmidt

0.1

02.05.2009

Table of Contents

1. Package repository

The package repository consists of the repository meta information, package list information, package meta information and the packages itself.

One repository can support multiple repository levels, for example stable, unstable and testing.

1.1 Protocol

The repository is either accessed via the HTTP or FTP protocol if it is a remote repository or directly via file APIs if it is a local repository.

1.2 Structure

The repository has the following file and directory structure:

- **site.xml** - the different levels of the repository are described here.
- **level directory** - every level is represented by a separate directory. The level name must be unique.
- **header.tar.gz** - every level directory contains a header archive file which contains the meta information of all packages in this level.
- **packages** - every level has a sub directory where the version directories of a package are located.
- **version** - every version of a package has its own subdirectory where the package files and package header file are stored.

The top level directory may also contain other files like **index.html** to display some information about the repository.

2. Package

A package is an archive file which contains a manifest file which describes the content of the archive and an action script which holds all the necessary information for executing the various actions on the archive (like install, update and remove). The archive name follows the following syntax: **package name_version-revision_target release.ipkg**

Package names are always completely lower case. The software version syntax depends on the software maintainer. Package names cannot have underscores in their names (hyphen is valid). Package versions cannot contain a hyphen.

Example: linked-list_1.4-5_5.2.ipkg

The example package name is to be interpreted in the following way:

- Software package: Linked List
- Software version: 1.4
- Package revision: 5
- Target release: V5R2

With this naming schema there can be multiple packages of the same package for different target releases in the same repository at the same level.

If a package does not depend on a specific target release the target release can be omitted (including the leading underscore).

2.1 Package types

There are three types of packages:

- **Binary packages**, which contain executables, configuration files, info pages, copyright information, and other documentation.
- **Source packages**, which contain the original unmodified source in a tar file and usually a **.diff.gz** file that contains the specific changes for this package management system.
- **Development packages**, which contains the necessary prototypes and copy books of the according program or service program.

2.2 File compression

Packages are archived with the tar library and compressed with the zlib compression library.

2.3 Manifest file

The manifest file describes the archive file. It is always named **manifest** and is a simple text file.

- Manifest-Version: version of the manifest file (starting with 1.0)

- Package-Name: Short name of the package
- Package-Title: Long descriptive name of the package
- Package-Maintainer-Name: Name of the package maintainer
- Package-Maintainer-Email: Email address of the package maintainer
- Category: category of the package (f. e. network or web)
- Package-Dependencies: Lists the dependecies of this package in the format: package name (operator version) , f. e. linked-list (>= 1.2). Multiple dependencies are separated by a newline.
- Package-Description: Detailed description of the package

2.4 Control file

A package might have a control file which can contain commands which should be executed either before or after a file operation.

For source packages the control file contains the compile commands (but can also contain any other command).

The control file can be absent or empty. Both situations are also valid.

Commands can be specified on an OS version level.

The configuration files of a package should be marked as such and not be overwritten on an update of the package.

3. Service program

The service program should supply procedures for every aspect of dealing with the repository and packages. The procedure signature should be compatible with the PCML specifications to provide native and PC PCML-based Utilities access to the package management functions on the configured server.

3.1 Procedures

3.1.1 List configured repositories

Gets a list of all configured repositories on the server.

3.1.2 Add repository

Adds a repository to the configuration.

3.1.3 Remove repository

Removes a repository from the configuration.

3.1.4 Change repository

Changes the parameter of a configured repository.

3.1.5 Update headers

Gets the header information from the configured repositories.

3.1.6 List installed packages

Lists all installed packages. A parameter defines the level of detail.

- ***NORMAL** - Only the short package name for each installed package will be returned.
- ***EXTENDED** - The short package name, full package name and version number will be returned for each installed package.
- ***FULL** - All available package information will be returned for each installed package.

3.1.7 Check package requirements

Check if it is possible to install the passed packages from the configured repositories. It returns a list of all packages which would be installed with the information if it is a new, updated, removed or dependend package.

3.1.8 Install package

Installs one or more packages. The procedure will install and/or upgrade all required packages. If a requirement cannot be met for a package the whole installation will be aborted.

3.1.9 Upgrade package

Upgrades one or more packages. The procedure will install and/or upgrade all required packages. If a requirement cannot be met for a package the whole upgrade will be aborted.

3.1.10 Search packages

Searches through the package headers for a given search string. The search scope can be selected: title, description or both.

3.1.11 Get package info

Returns all available information about a package.

4. Configuration

The configuration for the package management system consists of the two files i-get.conf and repository.conf which contain configuration data for the package management tool itself and the repositories which are accessed. Both files reside on the IFS in the path **/etc/i-get/**.

4.1 Repository configuration

The repository configuration file - **repository.conf** - contains one line for each repository. Comments are supported. If a line starts with # it is interpreted as a comment and will be ignored.

The configuration line consists of the following elements:

- **url** - valid url to the repository (including the protocol)
- **level** - repository level

The elements are separated by one or more spaces. The order in which the repositories are listed does not matter.

Example:

```
http://packages.rpgnextgen.com/unstable/ unstable
ftp://www.rpgnextgen.com/packages/stable/ stable
file:///usr/local/packages/ devel
```

4.2 Package management configuration

The configuration file **i-get.conf** can have the following entries:

- **Source-Base-Directory** - base directory for all source files. Every package has its own subdirectory.
- **Configuration-Base-Directory** - base directory for all coniguration file, default: **/etc**.
- **Include-Base-Directory** - base directory for all include files (header files/copybooks). Every package may have its own subdirectory, f. e. /usr/local/include/vector/vector_h.rpgle.
- **Object-Library** - library for all objects
- **Databasefile-Library** - library for all database objects (may be the same as the objects library).
- **Global-Binding-Directory** - the global binding directory which can be used for easy binding. All service programs and marked modules will be added automatically to it during installation.

TODO: add needed configuration settings

4.3 Compile options

Compile options can be configured globally for every compile command and on a per package level.

It is still undefined where and how compile options are stored.

5. Frontend

5.1 Command line interface

5.1.1 Functions

5.1.1.1 Update headers - update

The headers from the configured repositories are downloaded and stored in the IFS.

Usage: `i-get update`

No parameters are need for this command. All available configured repositories are queried. If one repository is not available it will be display to the user but the downloading of the header files from the other repositories is continued.

5.1.1.2 Install package - install

The specified packages and its depending packages are resolved and installed. If the packages are not locally available they are installed from the net.

Usage: `i-get install <package> [<package>]`

The `install` command accepts one or more packages separated by a space. The availablilty, status and dependencies for these packages are checked. If there are any changes to the install (f. e. adding of depending packages) the user has to confirm the installation.

5.1.1.3 Upgrade package - upgrade

For every specified package a check is made with the local header files to see if the installed version is the most current version available.

Usage: `i-get upgrade all | <package> [<package>]`

The `upgrade` command accepts the keyword `all` or one or more package names separated by a space. On the keyword `all` all installed packages are checked.

5.1.1.4 Remove package - remove

The specified packages are removed and deleted.

Usage: `i-get remove <package> [<package>]`

The `remove` command accepts one ore more package names separated by a space.

5.1.1.5 Clean local files - clean

All downloaded package files are deleted.

Usage: `i-get clean`

This command needs no further parameters.

5.2 GUI

GUI for package management and for building packages.

5.2.1 Package manager

Eclipse RCP display dependencies in a tree with graph viz

5.2.2 Package builder

Eclipse RCP Integration into RPG Next Gen gzip => in jre since 1.4.2 tar in org.apache.ant

5.2.2.1 Functions

The following functions should be implemented:

- addRepository - adds a repository to the local configuration
- removeRepository - removes a repository from the local configuration
- listRepositories - lists all configured repositories
- buildPackage - builds a package from the selected resources and asks for the meta information (wizard)
- getPackageList - gets the package list from a repository
- uploadPackage - uploads the package to a repository (can only be done if the repository supports that)
- removePackage - removes a package from a repository (con only be done if the repository supports that)

5.2.2.2 UI Components

The following components should be implemented:

5.2.2.2.1 Repositories view

The Repositories view lists all configured repositories. The menu bar has a shortcut button to add a repository via a wizard.

The context menu of the view has the following menu entries:

- create a new repository
- delete a repository
- get package list
- show repository properties

The child nodes of a repository are the package categories of the packages from the repository. The information must be downloaded from the repository prior to browsing the tree via the function of the repository item get package list. The child nodes of the categories are the packages.

This view should be in a general plugin which can be used by both the package manager and the package builder.

5.2.2.2.2 Package build wizard

The package build wizard is the central function of this application. It lets the user select the resources he wants to package and asks for the package meta information.

5.2.2.2.3 User information dialog

To ease the process of creating packages the user information part of the meta information of the package should be stored in the workspace and retrieved on package building. By this way the user does not have to enter his user information every time.

The user information could be stored in the preferences.

The main menu bar should have a menu entry to directly access the user information.

5.2.2.2.4 Repository wizard

A local or i5 repository should be created with this wizard.

The user should be able to configure the levels for the repository on an extra wizard page.

5.2.2.3 Plugin dependencies

The application depends on the following plugins (besides those from the Eclipse RCP framework):

- com.ibm.as400
- de.rpgng
- com.ice.tar
- org.apache.ant

6. Utility Programs

The following utility programs or service programs are needed for the frontend or backend:

- PROPERTIES - loading and saving configuration files
- HTTPAPI - downloading files from the repository

7. Software candidates for packaging

- DSM
- Linked List
- Linked List Utilities
- Vector
- Vector Utilities
- Linked Map
- Properties
- zlib
- libtar
- Proc4Name - Reflection

8. Links

- zlib
- tar
- HTTPAPI
- Graph Viz
- Debian Package Maintainer Guide

9. Further notes and ideas

1. Local repository - as most servers are located behind a firewall and some don't even have access to the internet the setup process of a local repository should be made very easy. The local repository should be updateable either through the net or via an archive file.
2. For those who don't have access to the internet and don't want to setup a server there should be a command to add a downloaded package to the repository meta information. The package should be put at a central folder in the IFS.

   ```
   i-get update-rep <ifs path to package>
   ```
3. A repository server or web application which lets the user add and remove packages via a simple call (f. e. SOAP). The admin user should also be able to move a package to the next level (f. e. from unstable to stable).
4. For an easier repository setup process there should be a command to create and administrate a repository.
5. CRTSRVPGM does not support a binder source file in the IFS other than the QSYS.LIB file system. So for creating a service program with a binder source file the binder source file has to be copied to the QSYS.LIB filesystem.