Maemo Diablo Reference Manual for maemo 4.1

Packaging, Deploying and Distributing

December 22, 2008

Contents

Chapter 1

Packaging, Deploying and Distributing

For installing and managing application packages, maemo uses the popular Debian package management system. From the user's perspective this is quite invisible, as the package management is performed using the Application Manager. It presents a flexible package framework that enables developers to easily create installable and manageable packages, without having to concentrate on the actual implementation details of the management.

Deb Installation Packages

The Debian package management system uses deb-packages which, in addition to files that will be installed, consist of metadata describing the package, dependencies to other packages and optional installation and removal scripts. The process of creating the packages is fortunately made quite simple by various package development tools.

The process of creating packages is described in section 1.2.

Package Repositories

The package management system makes use of package repositories, which are essentially web or FTP sites that contain packages and some information about them. It is not mandatory for a package to exist in any repository, but it has significant advantages: the user can easily update packages, other developers can use the packages as automatically installable dependencies and the repository's packages can be found using the Application Manager.

Information for working with repositories can be found in the Debian Repository HOWTO[3].

One-Click-Install

Maemo also has a option to create installation instruction files that enable the users to install applications simply by clicking a link on a web site. The .install files contain the required repository and package names. The format of these files is really simple, and it is described in section 1.2.12.

1.1 Creating Debian Packages

When creating a Debian package, the first step is to put all the files (the source code, and the png and other graphics, desktop and service files) in an empty directory called my-application-1.0.0. The directory name should follow the <package-name>-<app-version> convention. This means that the package that is being created for this application is called my-application.

Go to the source directory:

```
cd my-application-1.0.0
```

Set full name environment variable:

```
export DEBFULLNAME="Your Name"
```

Make initial debianization:

```
dh_make -e your.name@example.org
```

Next dh_make will ask a question and print a summary of the package:

```
Type of package: single binary, multiple binary, library, kernel module or cdbs?
 [s/m/l/k/b] s

Maintainer name : Your Name
Email-Address   : your.name@example.org
Date            : Fri, 30 Nov 2007 13:20:48 +0200
Package Name    : my-application
Version         : 1.0.0
License         : blank
Type of Package : Single
Hit  to confirm:
```

The dh_make command creates a debian subdirectory containing multiple configuration text files, most of which are templates that can be removed, since the application does not use them. The table below lists the needed files (others can be removed):

File in ./debian	**Description**
changelog	Application's change log
compat	Debian helper compatibily version. Leave it as it is.
control	Describes the packages to be made. For more information, see the paragraphs below the table.
copyright	Copyright text. Fill in the blanks.
rules	A makefile containing the rules to build all kinds of packages (such as source and binary)

The key files in ./debian are control and rules. They contain a generic template showing what they must look like. In control, the blanks simply have to be filled in, and in rules, the unwanted and unnecessary code will have to be removed.

The following example illustrates what the control file for the example application must contain:

```
Source: my-application
Section: user/other
Priority: optional
Maintainer: Your Name <your.name@example.org>
Build-Depends: debhelper (>= 4.0.0)
Standards-Version: 3.6.0

Package: my-application
Architecture: any
Depends: ${shlibs:Depends}
Description: A simple test application
 A very simple application with a short description.
 Which spans multiple lines actually.
XB-Maemo-Icon-26:
 iVBORw0KGgoAAAANSUhEUgAAABoAAAAaCAYAAACpSkzOAAAABmJLR0QA/wD/AP+g
 vaeTAAAACXBIWXMAAAsTAAALEwEAmpwYAAAAB3RJTUUH1gURDQoYya0JlwAAAU9J
 REFUSMftlL1KA0EUhb/NZl/ggnHQxsJUxt5CUucVJCCkDfgyKdIGG5/A0s5HEBtJ
 EdDAQGBgmw0YJmMzgXXYza5CtNkDW9zZw5z7c+ZCgwb/Ai3i9sVl/Bq8RIs4LRK1
 gJDsKvJyNXmJMuYTsMoY1zpgozaABdYArQNPZQ1kfyGU7SpqVwxzAMwABWhgpIwp
 4vWBB+AUWAI3ypjnfEXtPU4bLKx9vErTeCeiRSYF+fTn1j5dp2myE9EiU+DSi3wX
 ymeqRQAmZ3EcA5E/fgO6BULT8zhOcrwXoJdrXRa2Lgps2y2odAUcBUIXQdz78YyC
 SldAp8b7+bXrIv91qjZBietqCc2DjbAt4b2WxJkyZljVujlwp0U0cPxuLcAIuC+4
 dKxFlsDJarvdAGP/b6hFnDImYs+uG3hbO2AB3Jbsur63tQM+fFx3bzZocEB8AdV2
 gJBZgKTwAAAAAElFTkSuQmCC
```

The XB-Maemo-Icon-26 field contains the application icon file (in this case, hello_icon_26x26.png) encoded in base64. This is the icon that is shown in the Application Manager, next to the package name. To perform this encoding in Linux, either uuencode or openssl can be used; however, there may also be more suitable applications. The following example encodes an icon file using uuencode inside Scratchbox:

```
[sbox-DIABLO_X86: ~] > uuencode -m icon.png icon.png > icon.png.en
```

In the above example, the result of the encoding is printed into icon.png.en file. Do not forget to put a white space at the beginning of each line containing the icon-encoded text. The white space indicates a multi-line field. The same rule stands for the long package description (A very simple application[...]).

The Application Manager only shows packages in the user section. Thus, the Section: field in the control file must have the Section: user/<SUBSECTION> syntax, where <SUBSECTION> is arbitrary. See section 1.2.6 for information on the suggested values.

The rules file usually does not need to be changed. However, if the created .deb package is empty, the "install:" section should be checked to make sure that the DESTDIR environment variable is pointing to the right place, i.e. where the rest of the packaging scripts are looking for the application to be packaged.

Once the application is properly 'Debianized', building the application is performed easily with the command:

```
[sbox-DIABLO_X86: ~/my-application-1.0.0] > dpkg-buildpackage -rfakeroot
```

The system displays some output, including a couple of warnings near the end of it (about XB-Maemo-Icon-26), but that is normal. The parent directory now has a my-application_1.0.0-0_i386.deb file - the Debian package. This is the file that is distributed to maemo devices and installed using the Application Manager.

To test the package in the SDK, type:

```
[sbox-DIABLO_X86: ~/my-application-1.0.0] > cd ..
[sbox-DIABLO_X86: ~] > fakeroot dpkg -i my-application_1.0.0-0_i386.deb
```

Replace 'my-application...' with the actual name of the package. Packages can be installed in both X86 and ARMEL targets. The package architecture must match the Scratchbox target.

For more information about making Debian packages, see Debian New Maintainers' Guide [1]. For further information about creating application packages for maemo, see the next section 1.2.

N.B.

If the application is deploying icons to /usr/share/icons/hicolor, then the
gtk-update-icon-cache /usr/share/icons/hicolor
command should be run in the postinst script, otherwise the icons might not show up until the device is restarted!

1.2 Making Application Packages

This section explains how to make software packages that the end user can install to the Internet Tablet using the Application Manager tool in the device.

1.2.1 Prerequisites

This section assumes familiarity with the process of creating a .deb package. The basics of Debian packaging can be found in the previous section 1.1.

There is also the example "hello-world-app" package that can be used to get started. It can be installed with apt-get using the following command:

```
apt-get source hello-world-app
```

1.2.2 Application Manager

The Application Manager (also known as AM) is an end user friendly graphical front-end to the standard Debian package management infrastructure. When using the Application Manager, the end user does not have to use the apt-get tools.

The Application Manager uses the same backend tools as Synaptic, Aptitude, or apt-get (namely libapt-pkg), and it does it in the standard way, without imposing any constraints: packages are installed as root, and can touch the whole system, for example.

The normal way to distribute a package is, therefore, to put it into a repository and make it accessible to apt.

Either Application Manager or the apt-get tool can be used freely. Changes made to the system via apt-get or dpkg are picked up by the Application Manager without confusing it, and vice versa.

1.2.3 Packaging

Packages made for the Application Manager should follow a few extra rules, if they want to integrate nicely to the device. These rules are related to:

- general stuff
- dependencies
- sections
- icons
- the installation/removal policy of the Application Manager
- limited feedback to the user, no controlling terminal
- warning about removing running applications
- utilities to use in the maintainer scripts

These are explained in detail below.

1.2.4 General

All strings coming from the control information of a package are interpreted in UTF-8 when they are shown in the UI. If a string is not valid UTF-8, all bytes above 127 are replaced with '?' before displaying it.

1.2.5 Dependencies

This field should now contain all the dependencies for package, such as `${shlibs:Depends}`.

1.2.6 Sections

By default, the Application Manager only shows to the user packages in certain sections. This has been done to hide the existence of the hundreds of system packages making up the IT OS itself. The AM is, at this point, not intended to let the user manage the whole system, only a smaller set of third-party applications.

The AM only shows packages in the "user" section. Thus, the "Section:" field in the control file should be of the form

```
Section: user/<SUBSECTION>
```

where SUBSECTION is one of:

- user/accessories Accessories
- user/communication Communication
- user/games Games
- user/multimedia Multimedia
- user/office Office
- user/other Other
- user/programming Programming
- user/support Support

- user/themes Themes
- user/tools Tools

Thus, if the package is wanted in the Office subsection, the field Section: user/office should be included in the control information.

1.2.7 Icons

A package can have an icon displayed next to its name by the AM. Icons are included in the control information of a package as a base64 encoded field named "Maemo-Icon-26".

The image format of the icon can be anything understood by GdkPixbufLoader, but most commonly the PNG format is used.

The image should be 26x26 pixels with a transparent background.

The hello-world package shows an example of this.

The way to get these fields into the .deb files is to include them with a "XB-" prefix in the debian/control file. For more information, see the Debian Policy Manual, section 5.7 [2].

1.2.8 Installation and Removal Policy

The Application Manager has its own rules for automatically installing and removing packages, in addition to the ones specified by the user. These rules are tuned to eliminate most surprises for simple package management operations, but this makes them less useable for complicated things like "apt-get dist-upgrade". When designing the conflicts of packages, these rules should be accounted for.

Specifically, the AM will never automatically remove a user package.

If a conflict caused by installing a package could be resolved by removing a package, the AM will not do the removal, but will refuse the installation request instead.

When removing a package, all packages that are a direct or indirect dependency of the removed package will be considered for removal. They will, in fact, be removed when they are a non-user package, have been automatically installed by the AM to satisfy a dependency, and are no longer needed.

Unfortunately, the AM is not very good in reporting conflicts: when the package conflicts with a non-user package, the problem reported by the AM will blame the conflict on that non-user package, instead of on the user packages that depend on it.

1.2.9 Feedback from Maintainer Scripts

When the Application Manager runs the maintainer scripts, they have no controlling terminal; their standard input is connected to /dev/null. The DISPLAY variable is set correctly.

The Application Manager collects a transcript of the installation/uninstallation process, including the output of maintainer scripts. However, this output is hidden away in the "Log", and users should not be expected to look there and understand its contents.

Thus, it is the responsibility of the package creator to make sure that the maintainer scripts will not fail. This naturally does not mean that errors should be ignored, but that only things that have a very high chance of succeeding should be done. The simpler, the better.

1.2.10 Removing or Upgrading Running Applications

The Application Manager can run a script provided by the package, before removing or upgrading it. That script can tell the Application Manager to cancel the operation.

The canonical use for this feature is to warn the user when they try to remove or upgrade an application that is currently running. The utility 'maemo-application-running' can be used to perform this test. (See below for details.)

When uninstalling or upgrading a package named PACKAGE, the Application Manager will run the program named /var/lib/hildon-application-manager/info/PACKAGE.checkrm, if it exists. If this program exists with code 111, the operation will be canceled. In all other cases, including when the program terminates with a signal, the operation is carried out.

The arguments given to the *.checkrm program are either:

```
foobarexampleonly.checkrm remove
```

when the package is going to be removed, or

```
foobarexampleonly.checkrm upgrade VERSION
```

when it is going to be upgraded to version VERSION

1.2.11 Utilities to Use in Maintainer Scripts

There are some utilities available that can be used in the maintainer scripts to interact with the user:

```
 maemo-select-menu-location <app>.desktop [default-folder]
```

When the package contains a .desktop file, and consequently has an entry in the Desktop menu for this file, it can call maemo-select-menu-location in its postinst script to let the user choose a location for the entry.

The "app.desktop" parameter is the name of the .desktop file, without any directories. The default-folder parameter is optional, and when given, determines the default folder of the menu entry. If omitted, the menu entry will appear in "Extras".

The way to specify a folder that is provided by the system is by giving its logical name as listed in the /etc/xdg/menus/applications.menu file, NOT by giving its English name. Examples of logical names are

```
tana_fi_games
tana_fi_tools
tana_fi_utilities
```

When using a folder name that does not yet exist, it will be created. In that case, a logical name should NOT be used, since there will likely be no translations available for that logical name. When creating a new folder, a plain-text name should be used, in an appropriate language. However, it is advisable to use existing folders as much as possible.

Thus, if the package installs the file /usr/share/applications/hildon/foobarexampleonly.desktop, and it is wanted to go to the "Utilities" menu, this invocation should be put into the postinst script:

```
maemo-select-menu-location foobarexampleonly.desktop tana_fi_utilities
```

If it is wanted to go into a non-existing folder, the code used should be something like

```
maemo-select-menu-location foobarexampleonly.desktop "Cute hacks"
```

In order to use maemo-select-menu-location in postinst, Depends should be included in the "maemo-select-menu-location" package.

It is advisable to skip calling maemo-select-menu-location, when merely upgrading, as opposed to installing from scratch.

```
  maemo-application-running -x executable-file
  maemo-application-running -d app.desktop
```

This utility checks whether the application specified on the command line is currently running. If it is running, it exits with code 0. If it is not running, it exits with code 1. If an error occurs, it exits with code 2.

When using the -x option, the utility checks whether any process is currently executing that file, by looking into /proc/PID/exe.

When using the -d option, the utility uses the given .desktop file to find the service name of the application und queries D-BUS whether this service is currently registered. If there is no service name in the .desktop file, the utility uses the executable file as with the -x option.

In order to use maemo-application-running in postinst, Depends should be included in the "maemo-installer-utils" package.

```
  maemo-confirm-text [title] file
```

Displays the contents of FILE in a dialog with "Ok" and "Cancel" buttons. The default title of the dialog is "License agreement".

When the user clicks "Ok", this utility exits with code 0; when they click "Cancel", it exits with code 1; and when an error occurs, it exits with code 2.

If a license agreement is not shown in the postinst script, it is probably not a good idea to make the postinst script fail when the user does not agree to the license terms. Instead, the application could be configured in such a way that it will ask the user to agree to the license agreement again when the application is started, and refuse to run when they disagree.

In order to use maemo-confirm-text in postinst, Depends should be included in the "maemo-installer-utils" package.

1.2.12 Controlling Installation

In Application Installer (old name for Application Manager), there was a beta stage Single Click Install feature. Application Manager is backwards compatible with it, so the old .install files still work, but there are some new features available. This section will give an overview of the .install file usage and functionality.

The Application Manager offers three kinds of functionality with .install files: adding catalogues, installing packages from remote catalogues and installing packages from a memory card. The .install file follows the GKeyFile

format, as described by GLib. A GKeyFile consists of a number of groups, and each group contains key/value pairs. Each of the previously mentioned three functionalities is initiated by a specifically named group in the .install file, as described in the next chapters.

Adding Catalogues

Adding catalogues is performed with the "catalogues" group. This group has one similarly named mandatory key, "catalogues". The "catalogues" key is a list of strings referring to the catalogue groups that describe the catalogues to be added. Each catalogue is considered in turn, and the user is asked whether to add it or not. If it should be added, and a catalogue is already configured in the Application Manager that is equal to the one considered, the configured catalogue is removed first. When the user declines the adding, the next catalogue is considered. After considering every catalogue, the user is asked whether to "Refresh the list of applications". Here is an example:

```
[catalogues]
catalogues = extras; sdk

[extras]
name = maemo Extras catalogue
uri = http://repository.maemo.org/extras
components = free non-free

[sdk]
name = maemo SDK catalogue
uri = http://repository.maemo.org/
components = free non-free
```

A Group describing a catalogue can contain the following keys:

- name: This key gives the display name of the catalogue as shown to the user in the "Tool > Application catalogues" dialog. This key can be localised by following the GKeyFile conventions. If this key is omitted, the catalogue will have an empty name.
- uri: The URI part of the deb line that will be added to sources.list for this catalogue. This key is required for all catalogues except those used with the "card_catalogues" key.
- file_uri: When using file_uri instead of uri, the URI part of the deb line will use the "file://" method and the file_uri gives the actual pathname, relative to the location of the .install file. This key is required for all catalogues that are used with the "card_catalogues" key.
- dist: The distribution of the deb line that will be added to sources.list for this catalogue. If it is omitted, it will default to the distribution corresponding to the IT OS release on the device. The fact that the distribution should be selected automatically is remembered by the Application Manager. For example, if a backup is made, containing a catalogue with automatic distribution selection, and restored on a different IT OS release, the distribution for the new version will be used automatically.
- components: The components part of the deb line that will be added to sources.list for this catalogue. If it is omitted, the components part will be empty.

- filter_dist: This catalogue will be ignored when the distribution corresponding to the IT OS release on the device does not match.

When catalogues are compared, they are considered equal when their uri, dist and components fields are equal.

Installing Packages

In order to install packages with an .install file, the file must have an "install" group. The group has one mandatory key, "package", and one optional key, "catalogues". The "catalogues" key is handled as follows: each catalogue is considered in turn. The user is asked for confirmation if a catalogue that is not already configured is encountered. Alternatively, when a catalogue is already present but disabled, the user is informed that it needs to be enabled and is asked for a confirmation. If the user declines, the processing of the .install file stops, and the changes to the configured catalogues that have been made for it are reverted. After the list of catalogues has been processed, the list of applications is refreshed automatically, and the package is offered to the user for installing. Here is an example:

```
[install]
catalogues = foobar
package = maemofoo

[foobar]
name = Foobar Catalogue
name[en_GB] = Foobar Catalogue
name[de_DE] = Foobar Katalog
uri = http://foobar.com/repository
components = main
```

Installing Packages from Memory Card

Installing packages from a memory card is governed by the "card_install" group. It has two mandatory keys, "packages" and "card_catalogues", and an optional one, "permanent_catalogues". The "packages" key lists the names of packages that can be installed from the memory card, using the "card_catalogues". Installation of the packages happens in a temporary environment: in this environment, the normally configured catalogues are not available, only the catalogues listed by the "card_catalogues" key are configured. All of these catalogues must use "file_uri" instead of "uri".

The user gets to select the packages from a list, after which the installation proceeds one package after another. In case the installation of a package fails, an error note is displayed, and the processing stops. After the installation has completed, the optional group "permanent_catalogues" is processed. It functions similarly as the previously described catalogue adding.

Whenever a memory card is inserted that contains a file called .auto.install, that file is processed by the Application Manager. Usually, the .auto.install file contains a "card_install" group, of course. For example:

```
$ mkdir .repo
$ cp somewhere/*.deb .repo/
$ (cd .repo && apt-ftparchive packages . >Packages)
```

A matching .install file could look like this:

```
[card_install]
card_catalogues = repo
packages = app-1; app-2

[repo]
file_uri = .repo
dist = ./
```

The .repo and the .install files typically go to the root of the memory card. To make the memory card auto-installing, make a copy of the .install file and name it .auto.install.

1.2.13 Signing Package

In order to sign packages, use debsign tool inside Scratchbox. Please refer to debsign manual page [4] for instructions on how to use the tool.

1.3 Deploying Packages

1.3.1 Installing Application Packages to SDK

When installing Application Manager package to maemo SDK, debian package installer, like dpkg, should be used:

```
[sbox-DIABLO_X86: ~] > fakeroot dpkg -i application_i386.deb
```

Replace 'application' with the actual name of the package. Packages can be installed in both PC and ARMEL targets. The package architecture must match the Scratchbox target.

1.3.2 Installing Application Packages to Device

Debian package installer can also be used when installing application packages in maemo devices.

```
BusyBox v1.6.1 (2008-03-06 11:36:58 EET) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

/home/user # dpkg -i application_armel.deb
```

For maemo device the package architecture must be always armel.

1.3.3 Red Pill Mode

The Hildon Application Manager has a special hidden mode that makes it more power user friendly, and gives access to features that are not yet considered to be ready for everybody.

Activation

Go to "Tools > Application catalogue", click "New", enter "matrix" into the "Web Address" field, click "Cancel". Choosing the red pill will activate the red pill mode, obviously, and chosing the blue one will deactivate it.

Settings

After activating the red pill mode, the following additional settings are available in "Tools > Settings".

- *Clean Apt Cache*

  If activated, the equivalent of `apt-get clean` is performed after every install or update. (This is the behavior for blue pill mode.)

- *Assume Net Connection*

  This will not ask for an active IAP before downloading. This is useful when there is a network connection, but the device connectivity APIs are not available or do not know about it.

- *Break Locks*

  This will break needed locks instead of failing. This is done by default in blue pill mode, so that users would not lock themselves out, when a crash leaves a stale lock behind.

- *Show Dependencies*

  This adds another tab to the details dialog with some dependencies from the package.

- *Show All Packages*

  This will not filter out packages that are not in the "user" section. It will also allow installing packages from any section.

- *Show Magic System Package*

  This will include the "magic:sys" package in the list of packages. Updating that package will do something similar to `apt-get upgrade`. It is not yet fully defined what it will do exactly. This feature might become available in blue-pill mode at one point.

Bibliography

[1] Debian New Maintainers' Guide.
http://www.debian.org/doc/manuals/maint-guide/.

[2] Debian Policy Manual. http://www.debian.org/doc/debian-policy/.

[3] Debian Repository HOWTO. http://www.debian.org/doc/manuals/repository-howto/repository-howto.

[4] debsign manual page.
http://www.penguin-soft.com/penguin/man/1/debsign.html.