The New QPKG and Management

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author(s) | Comments |
| 0.1 | 13 May 2014 | Doro | Initial |
| 0.2 | 15 May 2014 | Doro |  |

1. Problems to Solve 2

2. QPKG 2

Package Source Layout 2

Localization 4

Open With (MIME) 4

Details of Design 5

Package Dependencies 5

Package Status and Procedure 5

Installation 6

Removal and Purging 7

Upgrade 8

3. Package Management 9

API 10

Design Detail 10

4. Build QPKG on Ubuntu by qdk2 10

Commands 11

5. QNAP Inter-Application Communication (QIAC) 11

Tool 12

Object: Core 12

Core/methods/Register 13

Core/signals/objectsChanged 13

Object: QnapNetwork 14

QnapNetwork/methods/GetHostname 14

QnapNetwork/methods/SetHostname 15

QnapNetwork/signals/hostnameChanged 15

QnapNetwork/signals/statusChanged (TODO) 16

Object: QnapPower 16

QnapPower/methods/RegisterPoweroffCallback 16

QnapPower/methods/RegisterSuspendCallback 17

Object: QnapDevice 17

QnaDevice/signals/event 18

6. Resource Limit 18

7. Application Container 18

8. Application Backup 19

# Problems to Solve

* Package Management
  + Dependency
* Pack QPKG easily
  + Tool chain
* Inter-Application Communication
* Backup or Migration between NAS, consider
  + Different NAS Hardware and version
  + File permissions and owner
* LXC/Docker
* Resource Limit

# QPKG

## Package Source Layout

Package source layout complies with Debian Package Rule, but replacing debian folder to *QNAP* folder. For example, the layout of application foobar looks like

foobar

├── QNAP

│   ├── changelog

│   ├── control

│   ├── rules

│   ├── foobar.conf

│   ├── foobar.dirs

│   ├── foobar.init

│   ├── foobar.install

│   ├── foobar.links

│   ├── foobar.mime

│   ├── foobar.postinst

│   ├── foobar.postrm

│   ├── foobar.preinst

│   └── foobar.prerm

├── etc

│   └── config

│   └── foobar.conf

├── share

│   └── foobar.txt

├── src

│   └── foobar.c

└── bin

└── foobar.sh

The file purpose in QNAP describes in Chapter 4 & 5 of Debian New Maintainers' Guide, <https://www.debian.org/doc/manuals/maint-guide/>.

The etc/config and usr folders will be automatically copied to Application root folder in NAS, such as /share/CACHEDEV1\_DATA/.qpkg/foobar/. But etc/config/ is an exception that will be copied to /etc/config/ in NAS.

QNAP/control is the most important file describing the package. The fields are defined in Debian Policy Manual, <https://www.debian.org/doc/debian-policy>. The field, which prefix is “Q-”, is used and defined by QNAP. Following is an example of *control* file,

Source: foobar

Section: python

Vcs-Bzr: lp:~oem-solutions-group/dell/template-abba

Homepage: http://123.com.tw/

Priority: extra

Build-Depends: cdbs (>= 0.4.43),

debhelper (>= 6),

python,

python-support (>= 0.6.4),

python-distutils-extra (>= 2.10)

Maintainer: Doro Wu <dorowu@qnap.com>

Package: foobar

Architecture: all

Q-AppName: Foobar Application

Q-Firmware: qnap (>= 4.1), qnap (<< 4.2)

Q-Platform: TS-870, TS-470, TS-?69\*

Q-Icon80: /share/CACHEDEV1\_DATA/.qpkg/foobar/a.png

Q-Icon100: /share/CACHEDEV1\_DATA/.qpkg/foobar/b.png

Q-PublishDate: 2014/05/01

Recommends: python-xlsxwriter

Depends: python (<< 3.0), python (>= 2.6),

python-launchpadlib | python-abc,

python-distutils-extra,

python-jinja2,

sshfs

Conflicts: lp-fish-init (<< 0.4)

Description: Utility scripts for OEM Dell enablement

UNKNOWN

## Localization

Some fields in QNAP/control, such as Q-AppName and Description, have to show in particular language. For example, in order to support Chinese traditional (zh\_tw), QNAP/control.zh\_tw is added as following,

Package: foobar

Q-AppName: 範例程式

Description: 一支範例程式

範例程式的詳細說明‧

The section is required to have field Package to specify which package is translating. Then the following fields are localized result.

## Open With (MIME)

Task: Migrate mime-support on Ubuntu to NAS

Open files in NAS with NAS applications registered by LXE/Docker Containers. The rules, which file suffix opens with which application, comply with MIME format of Ubuntu desktop. /etc/config/mime.types shows the relation between MIME type and file suffix, for example,

text/html html htm shtml

text/plain txt srt

The application, for example foobar, could add custom relation between MIME type and filename by foobar.xml located /etc/config/mime/packages/

specifies the action for dedicated MIME type. foobar has to put the command in the file /etc/config/mime/packages/foobar.xml, the file looks like,

<?xml version="1.0" encoding="UTF-8"?>

<mime-info xmlns="http://www.freedesktop.org/standards/shared-mime-info">

<mime-type type="application/vnd.sun.xml.draw">

<comment>OpenOffice.org 1.0 Drawing</comment>

<comment xml:lang="af">OpenOffice.org 1.0 Drawing</comment>

<comment xml:lang="zh-TW">OpenOffice.org 1.0 繪圖</comment>

<glob pattern="\*.sxd"/>

</mime-type>

</mime-info>

Besides, foobar could write its open rule. For example, create a file located QNAP/foobar.mime with the format <MIME Type>; <Command>, such as

application/vnd.sun.xml.draw; soffice -no-oosplash --draw '%s'; edit=soffice -no-oosplash --draw '%s'

When a file meets MIME type application/vnd.sun.xml.draw, “soffice…” will be one of open candidates.

### Details of Design

* foobar.mime rename to /etc/config/mime/packages/foobar
* When install/remove, run “update-mime” to update cache at /etc/config/mime/mailcap

## Package Dependencies

Ref: <https://www.debian.org/doc/debian-policy/ch-relationships.html#s-binarydeps>

* Depends, Pre-Depends
* Recommends
* Suggests
* Enhances
* Breaks, Conflicts
* Replaces
* Provides

DFS in depends, then Recommends

Depends: libc6 (>= 2.2.1), exim | mail-transport-agent

## Package Status and Procedure

Package status

* not-installed: The package is not installed on your system.
* config-files: Only the configuration files of the package exist on the system.
* half-installed: The installation of the package has been started, but not completed for some reason.
* Unpacked: The package is unpacked, but not configured.
* half-configured: The package is unpacked and configuration has been started, but not yet completed for some reason.
* triggers-awaited: The package awaits trigger processing by another package.
* triggers-pending: The package has been triggered.
* Installed: The package is unpacked and configured OK.

PACKAGE SELECTION STATES

* Install
* Hold
* Remove
* Purge

### Installation

### Removal and Purging

### Upgrade

# Package Management

Task: NAS tool qpt

Task: qscan-ng in qdk2

QNAP Packing Tool (QPT), similar to apt on Ubuntu, is a command-line tool for handling package on QNAP NAS. QPT provides following command to manage package:

qpt-install <package\_name>

qpt-remove <package\_name>

qpt-search <package\_name>

qpt-update

qpt-list [keyword]

qpt-show <package\_name>

The repositories were listed in /etc/config/qdt/sources.list. For example,

qpkg <http://update.qnap.com/qts/> 4.2 main extra

With above example, the program would try to fetch package during executing qpt-update using the URL as following,

* [http://update.qnap.com/qts/dists/4.2/main/binary-${arch}/Packages{.bz2](http://update.qnap.com/qts/dists/4.2/main/binary-$%7barch%7d/Packages%7b.bz2), .gz,}
  + [http://update.qnap.com/qts/dists/4.2/main/binary-${arch}/Packages\_zh\_tw{.bz2](http://update.qnap.com/qts/dists/4.2/main/binary-$%7barch%7d/Packages_zh_tw%7b.bz2), .gz,}
* [http://update.qnap.com/qts/dists/4.2/extra/binary-${arch}/Packages{.bz2](http://update.qnap.com/qts/dists/4.2/extra/binary-$%7barch%7d/Packages%7b.bz2), .gz,}

The packages file looks like:

Package: account-plugin-aim

Priority: optional

Section: gnome

Installed-Size: 941

Maintainer: Ubuntu Developers <ubuntu-devel-discuss@lists.ubuntu.com>

Architecture: amd64

Source: empathy

Version: 3.8.6-0ubuntu9

Depends: empathy (= 3.8.6-0ubuntu9), telepathy-haze, mcp-account-manager-uoa, unity-asset-pool (>> 0.8.24daily13.03.20.1)

Breaks: account-plugin-empathy

Filename: pool/main/e/empathy/account-plugin-aim\_3.8.6-0ubuntu9\_amd64.deb

Size: 8838

MD5sum: f7f709442600bb60bc5e230f905773ae

SHA1: 588d6aa693e4b9a9bda228360bc65dfab4527ed8

SHA256: 4a34e416bb37191d0b8e6855b27cdbf7cd63fec182ab415cc9e71b19cfe55e48

Description: Messaging account plugin for AIM

Homepage: <http://wiki.gnome.org/Empathy>

Description-md5: 1a2069e5dd5f4777061642b2d7c9a76a

Bugs: <https://bugs.launchpad.net/ubuntu/+filebug>

Origin: Ubuntu

Supported: 5y

Task: ubuntu-desktop, ubuntu-usb, edubuntu-desktop, edubuntu-usb

We need a new tool that works like qpkg-scanpackages to generate the Packages file. For more reference, refer to

* <https://www.debian.org/doc/manuals/repository-howto/repository-howto>
* <https://help.ubuntu.com/community/Repositories/Personal>

## API

Install/remove status report

/QnapQpt/methods/Install

/QnapQpt/methods/Remove

/QnapQpt/methods/Search

/QnapQpt/methods/Update

/QnapQpt/methods/List

/QnapQpt/methods/Status

/QnapQpt/signals/statusChanged

## Design Detail

Control files, where is QNAP/ of source package

/share/CACHEDEV1\_DATA/.qpkg/qpt/info/

/share/CACHEDEV1\_DATA/.qpkg/qpt/status

/share/CACHEDEV1\_DATA/.qpkg/qpt/lists/

# Build QPKG on Ubuntu by qdk2

Task: Package qdk2 on Ubuntu 12.04 and 14.04

qdk2 is a command-line tool to pack QPKG easily. It can be installed on Ubuntu 12.04 and 14.04 as following

sudo add-apt-repository ppa:fcwu-tw/ppa

sudo apt-get update

sudo apt-get install qpkg2

## Commands

$ qdk2 create –h

usage: qdk2 create [-h] [--fmt-qdk1] [-p package\_name] [-d directory] [-s]

optional arguments:

-h, --help show this help message and exit

--fmt-qdk1 QDK1 format

-p package\_name package\_name (Default: foobar)

-d directory destination folder (Default: $PWD)

-s, --sample-files .c, .so and doc samples

$ qdk2 build –h

“qdk2 create” would create dummy source package with name package\_name.

“qdk2 build” would build the package to \*.qpkg.

# QNAP Inter-Application Communication (QIAC)

Task: qiac, qiac-core, qiac-qnapnetwork in NAS

QIAC is based on ZeroMQ (<http://zeromq.org/>), comparing to others Advanced Message Queuing Protocol implementation, such as RabbitMQ and ActiveMQ, which is lightweight and fastest[[1]](#footnote-1).

The application that compatible with QIAC would publish its API with file socket by ZeroMQ. For example, the application foobar opens its socket at

/var/run/qiac/public/foobar/socket

, and optionally register itself by */methods/Register* of */Core* object. Any privilege APIs would open at

/var/run/qiac/privilege/foobar/socket

The API provider should confirm UID is 0 and permission is 700 of this socket file. When providers register its interface by */Core* object, */Core* would validate the permission and path.

## Tool

QIAC provides a command-line tool, qiac, to easily access this interface as shown as following,

qiac / # list all objects

qiac /QnapNetwork/ # list interfaces of QnapNetwork

qiac /QnapNetwork/Help # list interfaces of QnapNetwork

qiac /QnapNetwork/methods/GetHostname # call method GetHostname

qiac /QnapNetwork/methods/SetHostname ‘{“hostname”: “myhost”}’

qiac /QnapNetwork/signals/statusChanged # wait statusChanged signal

qiac –t 10 /QnapNetwork/signals/hostnameChanged # wait signal in 10 seconds

qiac /QnapNetwork/signals/ # wait all signals

## Object: Core

|  |  |  |  |
| --- | --- | --- | --- |
| Object | Interface | Arguments | Privilege |
| /Core | /methods/Register | {“object”: “str”, “program”: “str”} |  |
| /Core | /methods/Unregister | {“object”: “str”} |  |
| /Core | /methods/Call | {“interface”: “str”, “args”: “str”} |  |
| /Core | /Help |  |  |
| /Core | /signals/objectsChanged |  |  |

qiac /Core

{

“/Core”: {

“methods”: [

{

“Register”: {

“object”: “str”

}

},

{

“Unregister”: {

“object”: “str”

}

},

{

“Call”: {

“interface”: “str”,

“args”: “str”

}

}

],

“signals”: [

“objectsChanges”

]

}

}

### /Core/methods/Register

Arguments:

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| object | str |  |
| program | str | /Core has to run this program if any would like to use object through /Core/methods/Call. The program must wait requests from stdin. The object would be first line, and args would be second line. |

Return value:

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| ret | int | * 0: OK * < 0: Error |

Example:

qiac /Core/Register ‘{“object”: “/QnapPower”}’

{“ret”: 0}

### /Core/signals/objectsChanged

Arguments:

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
|  |  |  |

Return value:

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| Ret | int | * 0: OK * < 0: Error |
| object | str |  |
| event | str | “add” or “remove” |

Example:

qiac /Core/signals/objectsChange

{“ret”: 0, “object”: “/QnapPower”, “event”: “add”}

## Object: QnapNetwork

|  |  |  |
| --- | --- | --- |
| Object | Interface | Privilege |
| /QnapNetwork | /methods/GetHostname |  |
| /QnapNetwork | /methods/SetHostname | ● |
| /QnapNetwork | /signals/hostnameChanged |  |
| /QnapNetwork | /signals/statusChanged |  |
| /QnapNetwork | /Help |  |

qiac /QnapNetwork/Help

{

"QnapNetwork": {

"methods": [

{

"GetHostname": {}

},

{

"SetHostname": {

"hostname": "str"

},

"privilege": true

}

],

"signals": [

"hostnameChanged",

"statusChanged"

]

}

}

### /QnapNetwork/methods/GetHostname

Arguments:

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |

Return value:

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| ret | int | * 0: OK * < 0: Error |
| hostname | str |  |

Example:

qiac /QnapNetwork/methods/GetHostname

{“ret”: 0, “hostname”: “MyNAS”}

### /QnapNetwork/methods/SetHostname

Arguments:

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| Hostname | str |  |

Return value:

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| Ret | int | * 0: OK * < 0: Error |

Example:

qiac /QnapNetwork/methods/SetHostname ‘{“hostname”: “MyNAS”}’

{“ret”: 0}

### /QnapNetwork/signals/hostnameChanged

Arguments:

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |

Return value:

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| ret | int | * 0: OK * < 0: Error |
| hostname | str |  |

Example:

qiac /QnapNetwork/signals/

{“ret”: 0, “hostname”: “MyNAS”}

### /QnapNetwork/signals/statusChanged (TODO)

Address/Connection changed

Arguments:

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |

Return value:

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| ret | int | * 0: OK * < 0: Error |
| hostname | str |  |

Example:

qiac /QnapNetwork/signals/hostnameChanged

{“ret”: 0, “hostname”: “MyNAS”}

## Object: QnapPower

|  |  |  |  |
| --- | --- | --- | --- |
| Object | Interface | Arguments | Privilege |
| /QnapPower | /methods/RegisterPoweroffCallback |  |  |
| /QnapPower | /methods/RegisterSuspendCallback |  |  |
| /QnapPower | /methods/UnregisterPoweroffCallback |  |  |
| /QnapPower | /methods/UnregisterSuspendCallback |  |  |
| /QnapPower | /methods/ListPoweroffCallback |  |  |
| /QnapPower | /methods/ListSuspendCallback |  |  |
| /QnapPower | /signals/poweroff |  |  |
| /QnapNetwork | /signals/suspend |  |  |

### /QnapPower/methods/RegisterPoweroffCallback

Arguments:

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| Type | str | * “program” * “qiac” |
| Path | str | **Program**  Output progress to stdout. The program would be killed(SIGTERM, SIGKILL) if no output in 30 seconds  Return   * 0: continues to poweroff procedure * < 0: revoke   **qiac**  With PULL/PUSH way, Callback server returns code and status as following format:  {“ret”: 1, “status”: “Do cleanup”}  Where ret is   * 0: continues to poweroff procedure * 1: status update * < 0: revoke |

Return value:

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| Ret | int | * 0: OK * < 0: Error |

Example:

qiac /QnapPower/methods/RegisterPoweroffCallback ‘{“type”: “program”, “path”: “/share/CACHEDEV1\_DATA/.qpkg/foobar/usr/bin/poweroff\_check”}’

cleanup

cleanup

echo $?

0

### /QnapPower/methods/RegisterSuspendCallback

As same as /QnapPower/methods/RegisterPoweroffCallback

## Object: QnapDevice

|  |  |  |  |
| --- | --- | --- | --- |
| Object | Interface | Arguments | Privilege |
| /QnapDevice | /signals/event |  |  |

### /QnapDevice/signals/event

Arguments:

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |

Return value:

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| ret | int | * 0: OK * < 0: Error |
| object | str |  |
| event | str | “add” or “remove” |

Example:

qiac /QnapDevice/signals/event

{“ret”: 0, “object”: “/block/sdf/sdf1”, “event”: “add”}

# Resource Limit

Limit application running under constrain hardware resources via UI of AppCenter. The managed resources include:

1. CPU
   1. Time Quota
   2. Relative share of CPU time
   3. Cores accessible
2. Memory
   1. Memory Usage
   2. Swap Usage
3. IO (Block IO device or specific device node)
   1. Weight
   2. Throttle in bps or iops
   3. Read write permission of device node
4. Network (Applications must run in a container)
   1. Bandwidth
   2. Access between containers

# Application Container

Wrapping application with a container leads the benefits

1. Almost no effort on packing software supported on Ubuntu/Fedora.
2. Complete backup application no matter where the data/config are

In addition, a service discovery daemon helps the application is able to leverage services providing by native applications or containers. For example, WordPress and MediaWiki need a backend database MySQL. They can choose to build their own MySQL in two distinct containers or leverage a common MySQL container.

## Create an application Docker container

* Docker image
  + docker pull: from docker’s repository
    - Image usually is very large
    - docker’s repository sometimes is not stable
    - Friendly
  + docker build: Given Dockerfile source tree, user build himself
    - build failed if remote resource, which is using in Dockerfile, is temporally unavailable
    - Friendly
  + docker commit: user, based on somewhat image, do and install operations, then commit it as image.
    - Advanced user only
    - Friendly
  + QNAP provides (“image depends” that deliver less data to user)
* QPKG
  + Image
    - Load image
    - Import root filesystem
  + Init script
    - Start: docker run --name ${APP\_NAME} -v /share/CACHEDEV1\_DATA:/share -p 80:10080 ${IMAGE\_NAME}; docker start ${CONTAINER\_ID}
    - Stop: docker stop ${CONTAINER\_ID}
  + Open
* Backup
  + QNAP/app.backup
  + docker diff $CONTAINER\_ID

demo1: QPKG install App Ubuntu-ssh-ttyjs

demo2: Commit custom Ubuntu service, such as gitlab

## Application Sandbox

* namespace (nsexec)
  + UTS
  + IPC
  + Network
  + PID
  + User
* overlayfs, aufs, unionfs to leverage rootfs
  + mount --make-rshared /

# Application Backup

Consider

* Different NAS Hardware and version
* File permissions and owner

Methods

* Application specific
  + (QNAP/control) Q-BackupProgram: Program Path
  + (QNAP/app.backup)
* Generic
  + Overlayfs + RTRR
  + (QNAP/app.md5sum)
  + (QNAP/app.backupWatchDirs)

NAS hardware and version should not affect backup result

File permission is simply replicated by -a of cp and rsync command

Via an owner map, do owner changing for each backup

1. <http://blog.x-aeon.com/2013/04/10/a-quick-message-queue-benchmark-activemq-rabbitmq-hornetq-qpid-apollo/> [↑](#footnote-ref-1)