



Synology DiskStation Manager

3rd-Party Package Developer Guide

THIS DOCUMENT CONTAINS PROPRIETARY TECHNICAL INFORMATION WHICH IS THE PROPERTY OF SYNOLOGY INCORPORATED AND SHALL NOT BE REPRODUCED, COPIED, OR USED AS THE BASIS FOR DESIGN, MANUFACTURING, OR SALE OF APPARATUS WITHOUT WRITTEN PERMISSION OF SYNOLOGY INCORPORATED

Table of Contents

Package Developer Guide	1.1
Release Notes	1.2
Breaking Changes	1.3
Getting Started	1.4
System Requirements	1.4.1
Prepare Environment	1.4.2
Your First Package	1.4.3
Synology Toolkit	1.5
Build Stage	1.5.1
Pack Stage	1.5.2
Sign Package (only for DSM6.X)	1.5.3
References	1.5.4
Synology Package	1.6
INFO	1.6.1
Necessary Fields	1.6.1.1
Optional Fields	1.6.1.2
package.tgz	1.6.2
scripts	1.6.3
Script Environment Variables	1.6.3.1
Script Messages	1.6.3.2
conf	1.6.4
privilege	1.6.4.1
resource	1.6.4.2
PKG_DEPS	1.6.4.3
PKG_CONX	1.6.4.4
LICENSE	1.6.5
Synology DSM Integration	1.7
FHS	1.7.1
Desktop Application	1.7.2
Application Config	1.7.2.1
Application Help	1.7.2.2
Application I18N	1.7.2.3
Application Authentication	1.7.2.4
Privilege	1.7.3
Privilege Config	1.7.3.1
Resource	1.7.4
Resource Config	1.7.4.1
Resource Timing	1.7.4.2
Resource Update	1.7.4.3

Resource List	1.7.4.4
/usr/local linker	1.7.4.4.1
Apache 2.2 Config	1.7.4.4.2
Data Share	1.7.4.4.3
Docker	1.7.4.4.4
Index DB	1.7.4.4.5
Maria DB	1.7.4.4.6
PHP INI	1.7.4.4.7
Port Config	1.7.4.4.8
Systemd User Unit	1.7.4.4.9
Syslog Config	1.7.4.4.10
Web Service	1.7.4.4.11
Port	1.7.5
Monitor	1.7.6
Package Examples	1.8
Open Source Tool: tmux	1.8.1
Open Source Tool: nmap	1.8.2
Docker package	1.8.3
Web Package: WordPress	1.8.4
Publish Synology Packages	1.9
Get Started with Publishing	1.9.1
Submitting the Package for Approval	1.9.2
Responding to User Issues	1.9.3
Appendix A: Platform and Arch Value Mapping Table	1.10
Appendix B: Compile Applications Manually	1.11
Download DSM Tool Chain	1.11.1
Compile	1.11.2
Compile Open Source Projects	1.11.3
Appendix C: Publication Review & Verification	1.12

Synology DSM 7.0 Developer Guide

Synology offers this developer guide with instructions on how to develop packages on Synology NAS products. You should have basic understanding of Linux programming. With this guide, you can familiarize yourself with the following procedures:

- Compile programs to run on a Synology NAS.
- Integrate packages with the Synology DiskStation Manager (DSM).
- Integrate packages with the DSM help.
- Integrate packages with the DSM desktop application.
- Integrate packages with the DSM firewall.
- Integrate packages with the DSM resource monitor.

THIS DOCUMENT CONTAINS PROPRIETARY TECHNICAL INFORMATION WHICH IS THE PROPERTY OF SYNOLOGY INCORPORATED AND SHALL NOT BE REPRODUCED, COPIED, OR USED AS THE BASIS FOR DESIGN, MANUFACTURING, OR SALE OF APPARATUS WITHOUT WRITTEN PERMISSION OF SYNOLOGY INCORPORATED

Copyright

Synology Inc. ® 2021 Synology Inc. All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Synology Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer for personal use only and to print copies of documentation for personal use provided that the documentation contains Synology's copyright notice.

The Synology logo is a trademark of Synology Inc.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Synology retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Synology-labeled computers.

Every effort has been made to ensure that the information in this document is accurate. Synology is not responsible for typographical errors.

Synology Inc. 9F., No.1, Yuandong Rd., New Taipei City 22063, Taiwan

Synology and the Synology logo are trademarks of Synology Inc., registered in the United States and other countries.

Marvell is registered trademarks of Marvell Semiconductor, Inc. or its subsidiaries in the United States and other countries.

Freescale is registered trademarks of Freescale. Intel and Atom is registered trademarks of Intel.

Semiconductor, Inc. or its subsidiaries in the United States and other countries.

Other products and company names mentioned herein are trademarks of their respective holders.

Even though Synology has reviewed this document, SYNOLOGY MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY. IN NO EVENT WILL SYNOLOGY BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Synology dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Synology Package Framework 7.0

Breaking Changes

Find more details please refer to breaking changes [Breaking Changes In 7.0](#).

1. Package Framework

- Force lower privilege for package
- Force some INFO fields to be necessary
- Remove package signing
- Remove `run-as system` from privilege
- Change default home path from `target` to `home`
- Change `PACKAGE_ICON.PNG` from 72x72 to 64x64
- Change FHS directory owner according to privilege settings
- Change package log location to `/var/log/packages/[package_name].log` and `/var/log/synopkg.log`
- Consider `prestart` script on bootup

2. Package Center

- Remove keyring
- Remove trust level

3. Commands

- `synopkg start` starts the package with its dependees
- `synopkg install` checks if the package can be installed

New Features

1. SDK Plugin

- Add `package_install` module
- Add `package_uninstall` module
- Add `package_start` module
- Add `package_stop` module

2. Package Framework

- Add `var` directory for FHS
- Add `tmp` directory for FHS
- Add `home` directory for FHS
- Add `prereplace` script
- Add `postreplace` script
- Add `install_on_cold_storage` to INFO
- Add `exclude_model` to INFO
- Add `dsmappage` to INFO
- Add `use_deprecated_replace_mechanism` to INFO
- Add multiple directories support for `dsmuidir` in INFO

3. Resource Worker

- Add `strong-dependence` to data-share worker for package who needs auto start after encrypted share mounted
- Add `systemd-user-unit` worker

Enhancements

1. Package Framework

- Restart package after repaired according to its original state
- Cannot continue to install package if spk checksum is incorrect

2. Package Center

- Be able to start a package with its dependees
- Be able to stop a package with its dependers
- Be able to uninstall a package with its dependers
- Be able to repair start-failed package via repair button
- Community sources should have same name / source

Breaking Changes in 7.0

In DSM 7.0, there're some breaking changes in package framework Package Center and commands. Also see [Release Notes](#).

Package Framework Changes

1. Force lower privilege for package

All packages should provide `conf/privilege` with `package` in `run-as` explicitly. Any privileged operation should be accomplished via resource worker.

2. Force some INFO fields to be necessary

Any package should have `package`, `version`, `os_min_ver`, `description`, `arch` and `maintainer` fields. Furthermore, the value of `os_min_ver` should be at least `7.0-40000` or you cannot install the package correctly.

3. Remove package signing mechanism

Packages are no longer able to do signing in packing stage.

4. Remove `run-as system` from privilege

Packages will not be able to use `run-as system` in `conf/privilege`. Instead, all packages should run as `package`.

5. Change default home path from `target` to `home`

The home directory of package is changed from `/var/packages/[package_name]/target` to `/var/packages/[package_name]/home` and its mode will be 0700.

6. Change `PACKAGE_ICON.PNG` from 72x72 to 64x64

Package should have `PACKAGE_ICON.PNG` in 64x64 above 7.0.

7. Change FHS directory owner according to privilege settings

FHS directories such as `target` will have new privilege settings according to `conf/privilege`.

8. Change package log location to `/var/log/packages/[package_name].log` and `/var/log/synopkg.log`

Package operation log is still at `/var/log/synopkg.log` but control script log will be at `/var/log/packages/[package_name].log`. Besides, when you are developing a package, you should always pay attention to the content of `/var/log/messages` to check if there are any warning or error.

9. Consider `prestart` script on bootup

The `prestart` script will run on bootup to check if a package can be started.

Package Center Changes

1. Remove keyring && Remove trust level

User are no longer be able to add / remove keyrings on package center since we have deprecated the codesign mechanism of spk. Similarly, there will be no trust level settings for user to choose. Any non-synology package will get alert on installation.

Command Changes

1. `synopkg start` starts a package with its dependees

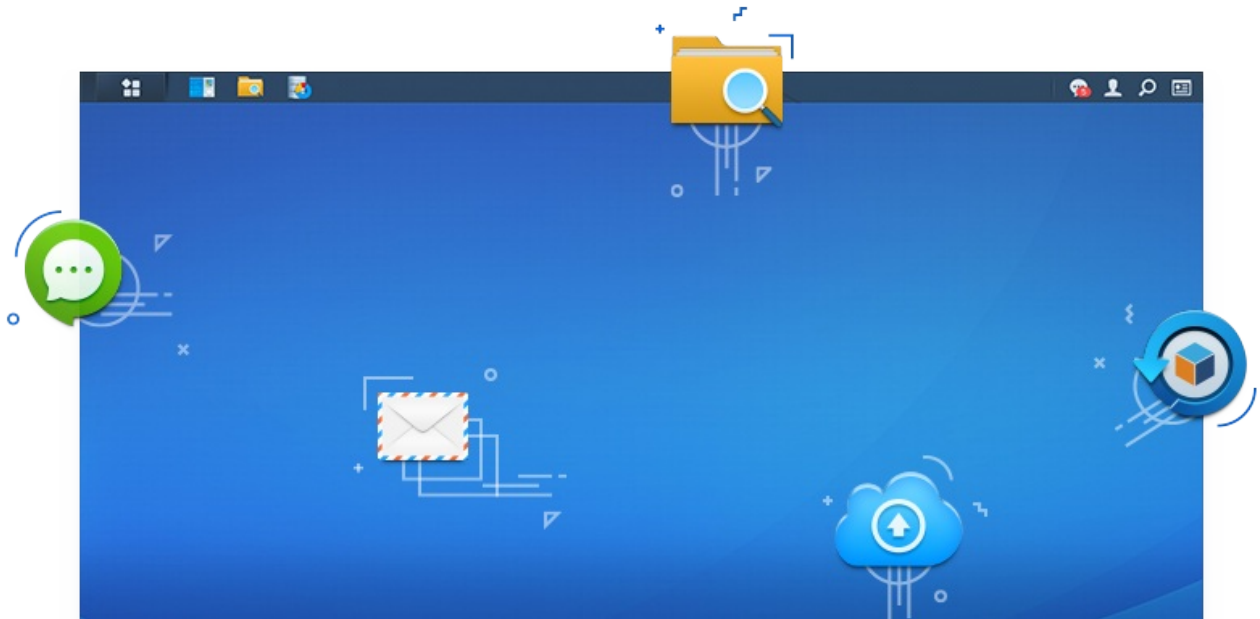
If A depends on B, run `synopkg start A` will also start B when B is not started.

2. `synopkg install` checks if package can be installed

The `synopkg install` command will have same constraints as UI installation.

Getting Started

Getting started to learn how to easily build packages just the way you like!



What can packages do ?

- access DSM API
- access owned data share folder
- integrate desktop application
- integrate help documents
- integrate firewall rules
- integrate resource monitor
- define lifecycle behaviour
- define relationship between packages
- define identity privilege

How to develop packages ?

To develop packages, you first need to know the entire working flow:

1. Prepare a NAS

You can choose one at our [official site](#) and buy it from [local synology partner](#). It is recommended to take one from the Plus Series.

2. Prepare environments for local development

Since our NAS is not always in `x86` or `x86_64` architecture, we should prepare corresponding environment to our NAS (for cross compiling if you are developing in C/C++). We provide tons of tools for creating different development environments of our NAS in an easy way.

3. Decide what you want to make

If you want to develop an application in `Node.js`, you can make your package depend on our official `Node.js` package. If you want to develop in `PHP`, you can still make your package depend on `PHP` package. We have already provided `Node.js`, `PHP`, `Perl`, `Python`, `Java` packages for language run time on DSM.

You can make great packages by leveraging our Package Framework to have stable, controllable and power saving properties. We provide complete toolkit for cross compiling and packing so you can also develop in an easy way.

4. Decide whether to publish packages onto official Synology Package Center

Begin to develop packages

In later topics, we will take a closer look at development. You can find articles such as

- [System Requirement](#)
- [Prepare Environment](#)
- [Your First Package](#)

System Requirements

Toolkit Requirements

- **64bit** generic linux environment with root permission (e.g., Ubuntu 18.04 LTS)
- bash ($\geq 4.1.5$)
- python ($\geq 2.7.3$)

Please **DO NOT install toolkit on Synology NAS** as your development environment. NAS is specialized for storage, and not for generic developing purpose. Instead, you can install `docker` package on NAS then setup a generic linux container to install the toolkit.

Runtime Requirements

- If your package is for DSM6 then you should have a DSM6 NAS.
- If your package is for DSM7 then you should have a DSM7 NAS.

Package for DSM6 is not compatible with DSM7

Prepare Environment

Install Toolkit

Toolkit Installation:

You need to clone the front-end scripts from this [link](#). We will use `/toolkit` as toolkit base in this document from now on.

```
apt-get install git
mkdir -p /toolkit
cd /toolkit
git clone https://github.com/SynologyOpenSource/pkgscripts-ng
```

Then you need to install a few tools to make the built tool work:

```
apt-get install cifs-utils \
python \
python-pip \
python3 \
python3-pip
```

At this moment, you can find toolkit files as the follows:

```
/toolkit
├─ pkgscripts-ng/
│   └─ include/
│   └─ EnvDeploy (deployment tool for chroot environment)
│   └─ PkgCreate.py (build tool for package)
└─ build_env/ (directory to store chroot environments)
```

Deploy Chroot Environment For Different NAS Target

For faster development, we have prepared several build environments of different architectures which contain some pre-built projects whose executable binaries or shared libraries are built in DSM, for example, `zlib`, `libxml2` and so on.

You can use `EnvDeploy` to deploy [corresponding environment](#) of your NAS. For example, if there is a NAS in `avoton` architecture, it is possible to use following commands to deploy a environment for `avoton` :

```
cd /toolkit/pkgscripts-ng/
git checkout DSM7.0
./EnvDeploy -v 7.0 -p avoton # for DSM7.0
```

It is possible to download environment tarballs manually. You have to put `base_env-7.0.txz`, `ds.{platform}-7.0.dev.txz` and `ds.{platform}-7.0.env.txz` into `toolkit/toolkit_tarballs`.

```
/toolkit
├─ pkgscripts-ng/
└─ toolkit_tarballs/
    ├─ base_env-7.0.txz
    ├─ ds.avoton-7.0.dev.txz
    └─ ds.avoton-7.0.env.txz
```

```
cd /toolkit/pkgscripts-ng/
```

```
./EnvDeploy -v 7.0 -p avoton -D # -D implies no download
```

As mentioned before, the deployed environment contains some pre-built libraries and headers which can be found under **cross gcc sysroot**. Sysroot is the default search path of compiler. If gcc cannot find header or library from the given path, it will then search `sysroot/usr/{lib,include}` .

```
/toolkit
├─ pkgscripts-ng/
│   └─ include/
│       └─ EnvDeploy
│           └─ PkgCreate.py
└─ build_env/
    ├─ ds.avoton-7.0/
    └─ ds.avoton-6.2/
        └─ usr/local/x86_64-pc-linux-gnu/x86_64-pc-linux-gnu/sys-root/
```

Available Platforms

You can use one of the following commands to show available platforms. If `-v` is not given, available platforms for all versions will be listed.

```
./EnvDeploy -v 7.0 --list
./EnvDeploy -v 7.0 --info platform
```

You may use any toolkit that belong to the same platform family to create spk for all platforms within the same platform family. e.g. you may use the toolkit for braswell to create package runs on all x86_64 compatible platforms. For platform family, please check [Platform and Arch Value Mapping Table](#).

Update Environment

Use `EnvDeploy` again to update the environment. For example, you can update avoton for DSM 7.0 as follows.

```
./EnvDeploy -v 7.0 -p avoton
```

Remove Environment

To remove a environment, you first need to unmount the `/proc` folder then remove the environment folder. The following commands illustrate how to remove an environment with version 7.0 and platform avoton.

```
umount /toolkit/build_env/ds.avoton-7.0/proc
rm -rf /toolkit/build_env/ds.avoton-7.0
```

Your First Package

Make sure you have [prepared the development environment](#) for your NAS.

Download the template package

You can download our template package from <https://github.com/SynologyOpenSource/ExamplePackages> and place the `ExamplePackages/ExamplePackage` directory at `/toolkit/source/ExamplePackage`.

```
/toolkit/
├─ build_env/
│   └─ ds.${platform}-${version}/
├─ pkgscripts-ng/
│   ├── EnvDeploy
│   └─ PkgCreate.py
└─ source/
    └─ ExamplePackage/
        ├── examplePkg.c
        ├── INFO.sh
        ├── Makefile
        ├── PACKAGE_ICON.PNG
        ├── PACKAGE_ICON_256.PNG
        ├── scripts/
        │   ├── postinst
        │   ├── postuninst
        │   ├── postupgrade
        │   ├── postreplace
        │   ├── preinst
        │   ├── preuninst
        │   ├── preupgrade
        │   ├── prereplace
        │   └─ start-stop-status
        └─ SynoBuildConf/
            ├── depends
            ├── build
            └─ install
```

Configure Build Configs

The steps to build package and pack package are configured under `${project_path}/SynoBuildConf/`. You can see three files:

- **depends:** configure dependencies between projects
- **build:** configure steps to build package
- **install:** configure steps to pack package into `.spk` file

This example will echo some messages by a program written in C language, so it is necessary to compile program in build stage. We apply `Makefile` in this example to help us doing cross compilation.

We do not concern what you do in `build` configuration so that it can even do nothing. The build system will just chroot into environment then call the corresponding `build`, `install` script according to the commands.

Configure Properties

The package information and its behavior are controlled by `INFO.sh` which will be translated into `INFO` file in `install`.

```
#!/bin/bash
# INFO.sh
source /pkgscripts/include/pkg_util.sh
```

```

package="ExamplePackage"
version="1.0.0000"
os_min_ver="7.0-40000"
displayname="ExamplePackage Package"
description="this is an example package"
arch="$(pkg_get_unified_platform)"
maintainer="Synology Inc."
pkg_dump_info

```

Configure Lifecycle Behaviour

The package control scripts can be found at `${project_path}/scripts/`. You can control the behaviour in each stage such as calling a `examplePkg` program on package start / stop.

```

#!/bin/sh
# scripts/start-stop-status
case $1 in
  start)
    examplePkg "Start"
    echo "Hello World" > $SYNOPKG_TEMP_LOGFILE
    exit 0
  ;;
  stop)
    examplePkg "Stop"
    echo "Hello World" > $SYNOPKG_TEMP_LOGFILE
    exit 0
  ;;
  status)
    exit 0
  ;;
esac

```

Write a program and configure its compilation and installation

It is common to bring compiled program into DSM via package. You can just write your program in C and add a Makefile to compile your programs.

```

// examplePkg.c
#include <sys/sysinfo.h>
#include <syslog.h>
#include <stdio.h>
int main(int argc, char** argv) {
    struct sysinfo info;
    int ret;
    ret = sysinfo(&info);
    if (ret != 0) {
        syslog(LOG_SYSLOG, "Failed to get info\n");
        return -1;
    }
    syslog(LOG_SYSLOG, "[ExamplePkg] %s sample package ...", argv[1]);
    syslog(LOG_SYSLOG, "[ExamplePkg] Total RAM: %u\n", (unsigned int) info.totalram);
    syslog(LOG_SYSLOG, "[ExamplePkg] Free RAM: %u\n", (unsigned int) info.freeram);
    return 0;
}

```

```

# Makefile
include /env.mak
EXEC= examplePkg
OBSJ= examplePkg.o
all: $(EXEC)
$(EXEC): $(OBSJ)
    $(CC) $(CFLAGS) $< -o $@ $(LDFLAGS)
install: $(EXEC)

```



```
mkdir -p $(DESTDIR)/usr/bin/
install $< $(DESTDIR)/usr/bin/
clean:
rm -rf *.o $(EXEC)
```

Any additional files (e.g., compiled program, media resources) should be packed into `package.tgz` file inside `.spk`. We provide several script commands to do such operations. In this example, we will pack compiled `examplePkg` executable via `install` build script.

```
# SynoBuildConf/install (partial)
create_package_tgz() {
    local firewere_version=
    local package_tgz_dir=/tmp/_package_tgz
    local binary_dir=$package_tgz_dir/usr/bin
    rm -rf $package_tgz_dir && mkdir -p $package_tgz_dir
    mkdir -p $binary_dir
    cp -av examplePkg $binary_dir
    make install DESTDIR="$package_tgz_dir"
    pkg_make_package $package_tgz_dir "${PKG_DIR}"
}
```

Build And Pack The Package

After you have finished preparing the package source code, you can use the following commands to build and pack the package into `.spk` at `/toolkit/result_spk/${package}-${version}/*.spk`.

```
cd /toolkit/pkgscripts-ng/
./PkgCreate.py -v 7.0 -p avoton -c ExamplePackage
```

```
/toolkit/
├─ pkgscripts-ng/
├─ build_env/
│   └─ ds.${platform}-${version}
└─ result_spk/
    └─ ${package}-${version}/
        └─ *.spk
```

Install And Test The Package

Go to **DSM > Package Center > Manual Install** then select your `.spk` file to do installation.



Synology Inc.

ExamplePackage

Running

Stop ▼

Other Information

Developer:
[Synology Inc.](#)

Installed version:
1.0.0-0001

Installed volume:
Volume 1

Once you have installed and started the package, you can see its message on UI and log at `/var/log/messages`.

Read More

- [Synology Toolkit](#)
- [Synology Package](#)
- [Synology DSM Integration](#)
- [Package Examples](#)

Synology Toolkit

In this section, we will explain the workflow of Package Toolkit. If you want to build a Synology Package without using Package Toolkit, you must:

- Prepare a cross compile tool chain
- Prepare a build environment
- Prepare metadata
- Compile source code
- Pack the package

Creating a package manually can be very complex for most developers, so we recommended using the Package Toolkit to make the package creation process easier.

```
/toolkit/  
├─ build_env/  
│   └─ ds.${platform}-${version}/  
└─ pkgscripts-ng/  
    ├─ EnvDeploy  
    └─ PkgCreate.py
```

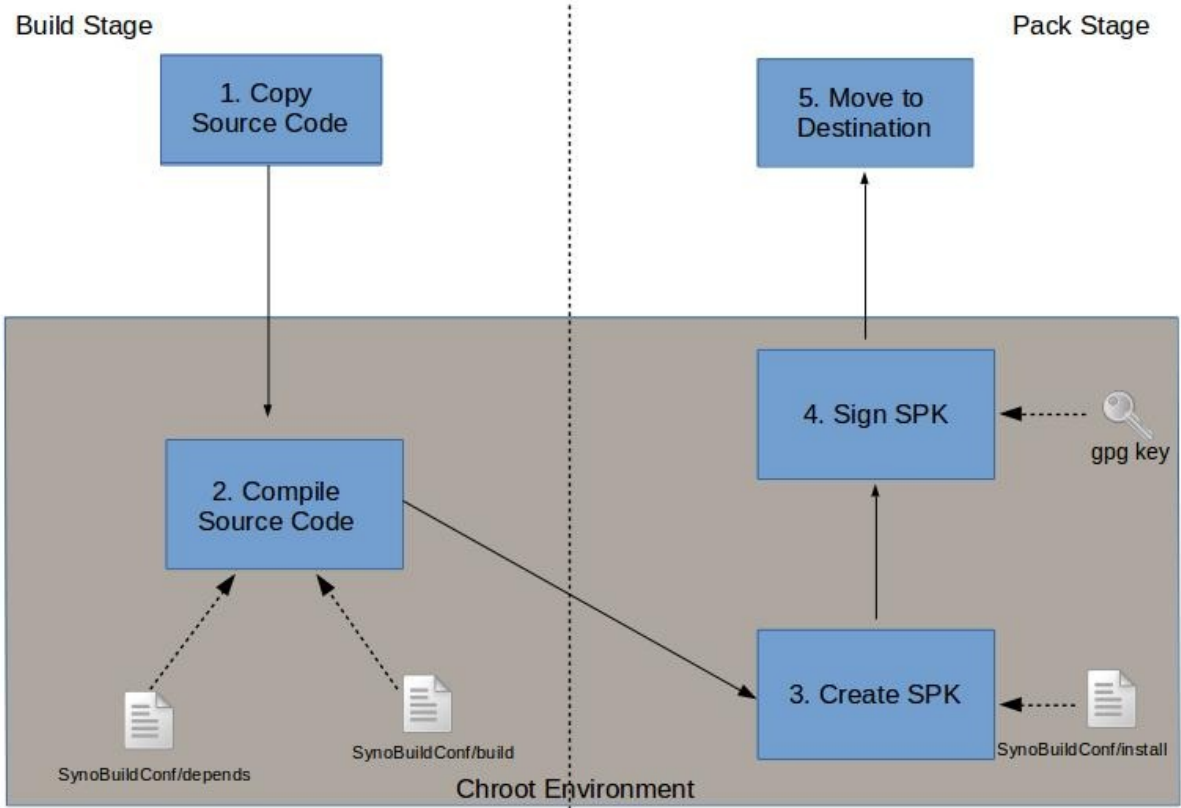
Create Package Workflow:

There are two stages in the `PkgCreate.py` package creation process:

- **Build Stage:** compile your project and all dependent projects in the correct order.
- **Pack Stage:** pack your project into an `.spk` file

To create your `.spk` file with `PkgCreate.py` properly, you need to provide additional configuration files and build scripts to describe how to build your project. These files are put in a folder named “**SynoBuildConf**” under your project.

- `SynoBuildConf/depends` : defines the dependency of your project. For further details, please refer to [Build Stage](#)
- `SynoBuildConf/build` : specifies `PkgCreate.py` on how to compile your project. For further details, please refer to [Build Stage](#)
- `SynoBuildConf/install` : specifies `PkgCreate.py` on how to pack your SPK file. For further details, please refer to [Pack Stage](#)
- `SynoBuildConf/install-dev` : similar to `SynoBuildConf/install`, but this will pack your `.spk` file in chroot environment rather than general DSM system. For further details, please refer to [Compile Open Source Project: nmap](#).



Build Stage:

In the Build Stage, `PkgCreate.py` will compile the project and its dependent projects. Please note that in this stage, `PkgCreate.py` depends on two build scripts (`SynoBuildConf/build` and `SynoBuildConf/depends`) to get the necessary information.

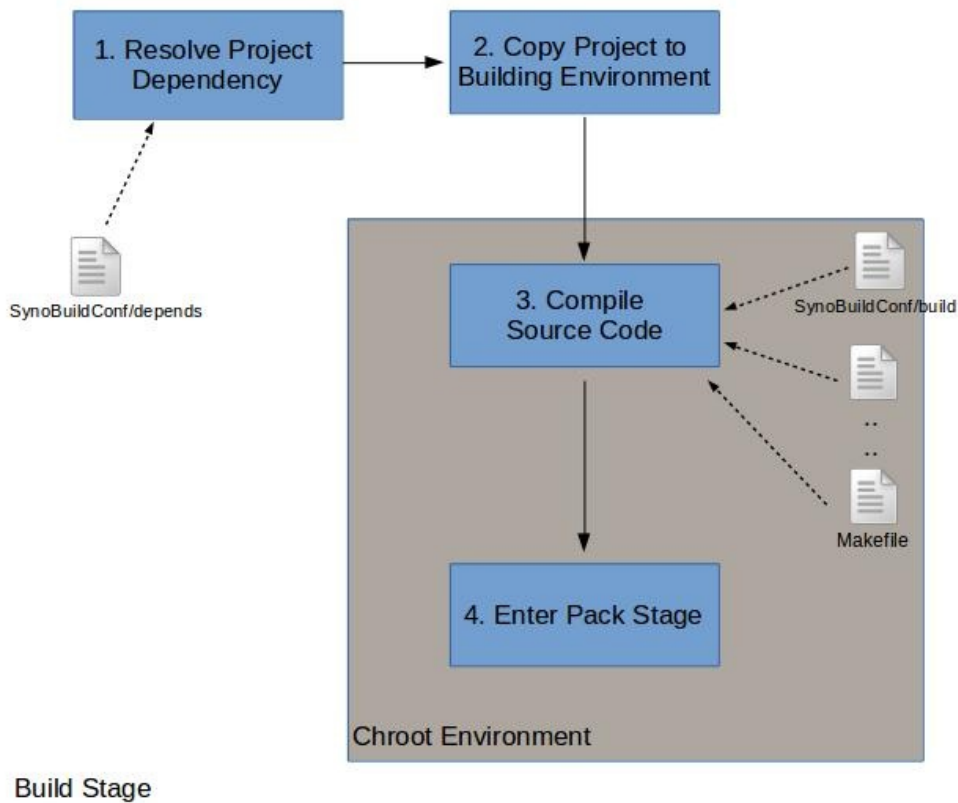
```
PkgCreate.py -v ${version} -p ${platform} ${project} # build project in specific platform version
```

```
/toolkit/  
├─ build_env/  
│   └─ ds.${platform}-${version}/  
├─ pkgscripts-ng/  
│   ├── EnvDeploy  
│   └─ PkgCreate.py  
└─ source/  
    └─ ${project}/  
        └─ SynoBuildConf/  
            ├── depends  
            ├── build  
            └─ install
```

Build Stage Workflow:

1. Based on your `SynoBuildConf/depend` , `PkgCreate.py` will locate the target DSM version from [default] section.
2. `PkgCreate.py` will resolve the projects you depend on.
3. Your project and the dependent projects which are placed under `/toolkit/source` will be hard-linked to `/toolkit/build_env/ds.${platform}/source` .
4. Their `SynoBuildConf/build` will be executed in order according to their dependency based on each `SynoBuildConf/depend` .
5. If your project is needed by other project for cross compiling, you may add `SynoBuildConf/install-dev` script. `install-dev` script will install cross compiled product into platform chroot.

Note: `SynoBuildConf/build` is executed under chroot environment `/toolkit/build_env/ds.${platform}`.



SynoBuildConf/depends

`PkgCreate.py` will resolve your dependency according to this configuration file. You need to specify your project dependency and the build environment of your project in this file. For example:

```
[BuildDependent]
# each line here is a dependent project

[ReferenceOnly]
# each line here is a project for reference only but no need to be built

[default]
all="7.0" # toolkit environment version of specific platform. (all platform use 7.0 toolkit environment)
```

There are three fields in `SynoBuildConf/depends` :

- **BuildDependent:** Describes other projects which are dependent on this project. For further details about this field, please refer to [Compile Open Source Project: nmap](#).
- **ReferenceOnly:** Describes other projects which are referred by this project, without the build process.
- **default:** Describes the toolkit environment. This section is a necessary field. It indicates each platform to build against some DSM version and the key "**all**" means all platform use this version by default.

You can use `ProjDepends.py` script to see whether the dependency order of your projects is correct. Option `-x0` will traverse all dependent projects of `${project}`.

```
cd /toolkit/pkgscripts-ng
./ProjDepends.py -x0 ${project}
```

If your application contains more than one project, put them in `/toolkit/source` and edit `SynoBuildConf` accordingly for each of them. For advanced usage, you may refer to [Compile Open Source Project](#) and [References](#).

SynoBuildConf/build

`SynoBuildConf/build` is a shell script that tells `PkgCreate.py` how to compile your project. The current working directory of this shell script is located in `/source/${project}` under chroot environment.

All pre-built binaries, headers, and libraries are under **cross compiler sysroot** in chroot environment. Since sysroot is the default search path of cross compiler, you do not need to provide `-I` or `-L` to `CFLAGS` or `LDFLAGS`.

Variables:

You can also find most of them in `/toolkit/build_env/ds.${platform}-${version}/{env.mak,env32/64.mak}`. They can be used in `SynoBuildConf/build`:

- **CC**: path of gcc cross compiler.
- **CXX**: path of g++ cross compiler.
- **LD**: path of cross compiler linker.
- **CFLAGS**: global cflags includes.
- **AR**: path of cross compiler ar.
- **NM**: path of cross compiler nm.
- **STRIP**: path of cross compiler strip.
- **RANLIB**: path of cross compiler ranlib.
- **OBJDUMP**: path of cross compiler objdump.
- **LDFLAGS**: global ldflags includes.
- **ConfigOpt**: options for configure.
- **ARCH**: processor architecture.
- **SYNO_PLATFORM**: Synology platform.
- **DSM_SHLIB_MAJOR**: major number of DSM (integer).
- **DSM_SHLIB_MINOR**: minor number of DSM (integer).
- **DSM_SHLIB_NUM**: build number of DSM (integer).
- **ToolChainSysRoot**: cross compiler sysroot path.
- **SysRootPrefix**: cross compiler sysroot concat with prefix /usr.
- **SysRootInclude**: cross compiler sysroot concat with include_dir /usr/include.
- **SysRootLib**: cross compiler sysroot concat with lib_dir /usr/lib.

```
# SynoBuildConf/build

case ${MakeClean} in
    [Yy][Ee][Ss])
        make distclean
        ;;
esac

make ${MAKE_FLAGS}
```

The above example calls the `make` command and compiles your project according to your **Makefile** located in `/source/${project}`.

Synology toolkit environment has included selected prebuild projects. You can enter the chroot and use following commands to check if needed header or project is provided by toolkit.

```
## inner chroot
dpkg -l # list all dpkg projects.
dpkg -l {project dev} # list project install files
dpkg -s {header/library pattern} # search header/library pattern.
```

For example, the project needs `zlib.h` and `libz.so` in the build stage. Use following command to check if zlib and its component are installed in chroot.

```

chroot /tookit/build_env/ds.avoton-7.0/
## inner chroot
>> dpkg -l | grep zlib
ii  zlib-1.x-avoton-dev      7.0-7274      all          Synology build-time library

>> dpkg -L zlib-1.x-avoton-dev
/.
/usr
/usr/local
/usr/local/x86_64-pc-linux-gnu
/usr/local/x86_64-pc-linux-gnu/x86_64-pc-linux-gnu
/usr/local/x86_64-pc-linux-gnu/x86_64-pc-linux-gnu/sys-root
/usr/local/x86_64-pc-linux-gnu/x86_64-pc-linux-gnu/sys-root/usr
/usr/local/x86_64-pc-linux-gnu/x86_64-pc-linux-gnu/sys-root/usr/lib
/usr/local/x86_64-pc-linux-gnu/x86_64-pc-linux-gnu/sys-root/usr/lib/libz.so
/usr/local/x86_64-pc-linux-gnu/x86_64-pc-linux-gnu/sys-root/usr/lib/libz.a
/usr/local/x86_64-pc-linux-gnu/x86_64-pc-linux-gnu/sys-root/usr/lib/pkgconfig
/usr/local/x86_64-pc-linux-gnu/x86_64-pc-linux-gnu/sys-root/usr/lib/pkgconfig/zlib.pc
/usr/local/x86_64-pc-linux-gnu/x86_64-pc-linux-gnu/sys-root/usr/lib/libz.so.1
/usr/local/x86_64-pc-linux-gnu/x86_64-pc-linux-gnu/sys-root/usr/lib/libz.so.1.2.8
/usr/local/x86_64-pc-linux-gnu/x86_64-pc-linux-gnu/sys-root/usr/include
/usr/local/x86_64-pc-linux-gnu/x86_64-pc-linux-gnu/sys-root/usr/include/zconf.h
/usr/local/x86_64-pc-linux-gnu/x86_64-pc-linux-gnu/sys-root/usr/include/zlib.h

>> dpkg -S zlib.so
zlib-1.x-avoton-dev: /usr/local/x86_64-pc-linux-gnu/x86_64-pc-linux-gnu/sys-root/usr/lib/libz.so
zlib-1.x-avoton-dev: /usr/local/x86_64-pc-linux-gnu/x86_64-pc-linux-gnu/sys-root/usr/lib/libz.so.1.2.8
zlib-1.x-avoton-dev: /usr/local/x86_64-pc-linux-gnu/x86_64-pc-linux-gnu/sys-root/usr/lib/libz.so.1

```

Some open source projects require to use other projects' cross compiled product while building their own . For example, `python` needs `libffi` and `zlib` while configure, we need to provide those two project before build `python` . You can install the cross compiled product into the destination you want in build script. Please refer to [Compile Open Source Project: nmap](#) for more information.

Makefile

The following example shows a `Makefile` . Most of the content contains typical makefile rules. Note that when writing your project `Makefile` , you can utilize pre-defined variables in `/env.mak` .

```

## You can use CC CFALGS LD LDFLAGS CXX CXXFLAGS AR RANLIB READELF STRIP after include env.mak
include /env.mak

EXEC= examplePkg
OBSJ= examplePkg.o

all: $(EXEC)

$(EXEC): $(OBSJ)
    $(CC) $(CFLAGS) $< -o $@ $(LDFLAGS)

install: $(EXEC)
    mkdir -p $(DESTDIR)/usr/bin/
    install $< $(DESTDIR)/usr/bin/

clean:
    rm -rf *.o $(EXEC)

```

For more detailed descriptions about makefile, please refer to the article [here](#).

Pack Stage:

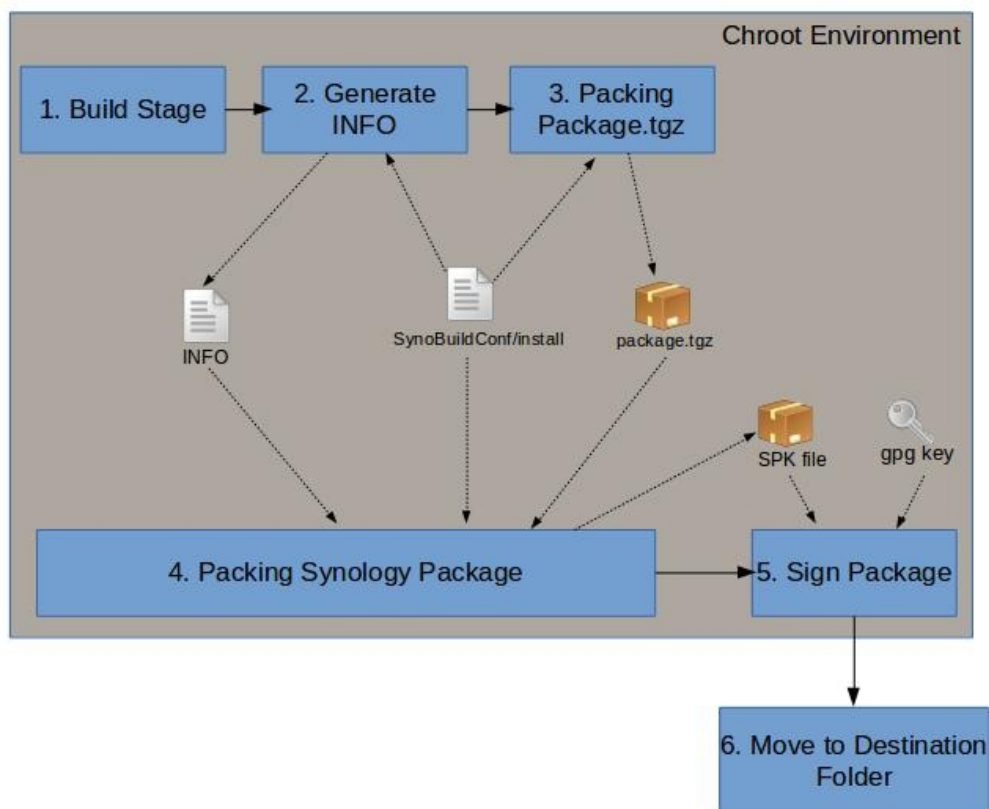
In the Pack Stage, `PkgCreate.py` packs all the necessary files according to your metadata and creates a `.spk` at `/toolkit/result_spk`. If you want `PkgCreate.py` to enter the Pack Stage without the Build Stage, simply run **PkgCreate.py** with the `-i` option.

```
cd /toolkit
pkgscripts-ng/PkgCreate.py -i ${project}
```

```
/toolkit/
├─ build_env/
│   └─ ds.${platform}-${version}/
├─ pkgscripts-ng/
│   ├── EnvDeploy
│   └─ PkgCreate.py
└─ source/
    └─ ${project}/
        └─ SynoBuildConf/
            ├── depends
            ├── build
            └─ install
```

Pack Stage Work Flow:

1. `PkgCreate.py` will execute the build script `SynoBuildConf/install`.
 - i. Create `INFO` file by using `INFO.sh`.
 - ii. Move necessary files to a temporary folder, `/tmp/_install`, for instance, and create `package.tgz`.
 - iii. Move necessary metadata and resources to the temporary folder, `/tmp/_pkg`, for instance, and create the `.spk` file.
2. `PkgCreate.py` will sign the newly created `.spk` file with a `gpg` key which is placed under `/root/` (the package signing mechanism is deprecated after DSM7.0).



SynoBuildConf/install

This file must be written in bash and indicates on how to pack your project. The current working directory is `/source/${project}` under chroot environment. If this is the top project of your package, this file will define how to create the `.spk` file, including directory structure and the `INFO` file.

```
#!/bin/bash
### Use PKG_DIR as working directory.
PKG_DIR=/tmp/_test_spk
rm -rf $PKG_DIR
mkdir -p $PKG_DIR

### get spk packing functions
source /pkgscripts-ng/include/pkg_util.sh

create_inner_tarball() {
    local inner_tarball_dir=/tmp/_inner_tarball

    ### clear destination directory
    rm -rf $inner_tarball_dir && mkdir -p $inner_tarball_dir

    ### install needed file into PKG_DIR
    make install DESTDIR="$inner_tarball_dir"

    ### create package.txz: $1=source_dir, $2=dest_dir
    pkg_make_inner_tarball $inner_tarball_dir "${PKG_DIR}"
}

create_spk(){
    local scripts_dir=$PKG_DIR/scripts

    ### Copy Package Center scripts to PKG_DIR
    mkdir -p $scripts_dir
    cp -av scripts/* $scripts_dir
}
```

```

### Copy package icon
cp -av PACKAGE_ICON*.PNG $PKG_DIR

### Generate INFO file
./INFO.sh > INFO
cp INFO $PKG_DIR/INFO

### Create the final spk.
# pkg_make_spk <source path> <dest path> <spk file name>
# Please put the result spk into /image/packages
# spk name functions: pkg_get_spk_name pkg_get_spk_unified_name pkg_get_spk_family_name
mkdir -p /image/packages
pkg_make_spk ${PKG_DIR} "/image/packages" $(pkg_get_spk_family_name)
}

create_inner_tarball
create_spk

```

At the beginning, the script called the **PrepareDirs** function which will prepare the necessary folder for the project.

After created the folder, the script called **SetupPackageFiles** to move necessary resource files to `$INST_DIR` and `$PKG_DIR`. In this step, we called the `INFO.sh` file to create the `INFO` file. Although you may put the codes that generate the `INFO` file in the `SynoBuildConf/install` script, we highly recommend that you create the **INFO** separately. Generally, we name it `INFO.sh`. You can see how to write `INFO.sh` in the following subsections.

After moving the resource file to the proper location, we called the `MakePackage` function to create the package. We included/sourced a script called `pkg_util.sh` which is located at `/pkgscripts-ng/include`. The `pkg_make_package` and `pkg_make_spk` defined in `pkg_util.sh` can help to create `package.tgz` and `.spk`.

- `pkg_make_inner_tarball $1 $2` : Create `packages.tgz` of \$2 from files in \$1.
- `pkg_make_spk $1 $2` : Create spk of \$2 from files in \$1.

INFO.sh

As mentioned earlier, `INFO.sh` is just an optional script. You can create the `INFO` file by hand or move the code to `SynoBuildConf/install`. However, we strongly recommend that you utilize `INFO.sh` so that you can create the `INFO` file separately from `SynoBuildConf/install`.

```

#!/bin/bash

source /pkgscripts-ng/include/pkg_util.sh

package="ExamplePkg"
version="1.0.0000"
displayname="Example Package"
maintainer="Synology Inc."
arch="$(pkg_get_unified_platform)"
description="this is a Example package"
[ "$(caller)" != "0 NULL" ] && return 0
pkg_dump_info

```

The above code is just an example to show some important variables for `pkg_dump_info`. If you want to know more details about the `INFO` file and each fields, please refer to [INFO](#).

Similar to `SynoBuildConf/install`, we must first include `pkg_util.sh`. After that, we can set up proper variables and call the `pkg_dump_info` to create the `INFO` file correctly. As you may have noticed, we used another helper function called `pkg_get_platform` to set the architecture variable. This variable indicates the current platform we are building.

- **pkg_get_spk_platform**: Return platform for “arch” in `INFO`.
- **pkg_dump_info**: Dump `INFO` according to given variables.

Remember to make `INFO.sh` be executable (e.g., `chmod +x INFO.sh`)

Spk Packing Functions

Synology package framework provides several functions to improve efficiency of packing packages. The functions such as generating architecture information in the `INFO` file, separating `.spk` name and creating `.spk` will be enabled after import `/pkgscripts-ng/include/pkg_util.sh`.

Spk Platform Functions

A `.spk` can be installed on one or more platforms. You can decide which platform can be installed via `INFO` file.

function name	Values	Description
(No function)	noarch	Package only contain scripts. spk can be run on all synology Models.
<code>pkg_get_platform_family</code>	x86_64 i686 armv7 armv5 ppc...	Unify platforms with same kernel into a platform family . The package can run on same family of synology models.
<code>pkg_get_spk_platform</code>	bromolow cedarview qoriq armadaxp...	Directly output the platform where the toolkit environment is used. The package can only run on the specific platform.

- If your package doesn't have any native binary, you can use `noarch` as the platform and write the scripts for your package. Package with `arch=noarch` can be installed onto any synology model.
- If your package doesn't have any kernel related functions, the package can run on the same architecture platforms. Use function `pkg_get_platform_family` to get platform family. Package can be installed on the models included in the same platform family. For example, package with `arch=x86_64` can be install onto `bromolow cedarview broadwell` models.
- If your package contains kernel related functions, every platforms will need a specific spk. Please use function `pkg_get_spk_platform` to get the platform(s) which is compatible with your environment.

Spk Naming Functions

After spk generated, we need to distinguish spk name by platform. We can use spk name functions:

Function name	Corresponding platform function	Example	Description
<code>pkg_get_spk_name</code>	<code>pkg_get_spk_platform</code>	examplePkg-bromolow-1.0.0000.spk / examplePkg-cedarview-1.0.0000.spk ...	Spk name depends on which toolkit environment is using.
<code>pkg_get_spk_name</code>	noarch	examplePkg-1.0.0000.spk	If the package has <code>platform="noarch"</code> , this function will output spk name without platform info.
<code>pkg_get_spk_family_name</code>	<code>pkg_get_platform_family</code>	examplePkg-x86_64-1.0.0000.spk	Spk name will be unified into platform family. Same platform family will generate the same spk name. i.e bromolow and x64 will have same spk name.

You need to use path of `INFO` as argument. If no path specified, the function will get `INFO` file from `$PKG_DIR/INFO` automatically.

Spk Creation Functions

Developer can use `pkg_make_spk` to create spk.

```
pkg_make_spk $source_path $dest_path $spk_name
```

- **source_path** is spk source directory. All spk files must copy into this directory before run `pkg_make_spk`.
- **dest_path** is target spk path.
- **spk_name** is spk name with/without platform info.

Example:

```
pkg_make_spk /tmp/_test_spk "/image/packages" $(pkg_get_spk_family_name)
```

Sign Package (only for DSM6.X)

Signing mechanism is deprecated after DSM7.0, you don't need this if you are developing package for DSM7.0

Between DSM5.1 and DSM6.X, we have a built-in code sign mechanism to ensure the package's publisher integrity. The toolkit has a `CodeSign.php` script to sign the package with **GnuPG** keys. If you do not have a GPG key, you will need to generate one.

Setup existing GPG key

If you have your own GPG key (**without a passphrase**) already, you will need to put the private key under `/root/.gnupg` of each platform (e.g., `/toolkit/build_env/ds.${platform}-6.2/root/.gnupg/`).

The package signing scripts now only support keys generated by GPG 2.1. If you don't have your own GPG key or you are using GPG keys in GPG 2.2 format, you need to prepare GPG tool and generate one.

Setup GPG tool provided by dist

If your dist provides GPG 2.1, install `gpg` with your package management tool in your dist. For ubuntu developers, you may run `apt-get install gpg gpg-agent` to setup GPG tool.

Make sure you are using GPG 2.1. If your dist does not provides GPG 2.1, Follow the instructions in the next section to prepare your GPG tool.

Setup GPG tool with docker

Assume you're developing on avoton platform with DSM version 6.2, and `/tmp/gpgkey` is the temporary folder saving the GPG key generated.

```
mkdir /tmp/gpgkey
docker run --rm -it -v /tmp/gpgkey:/root/.gnupg -e GPG_TTY=/dev/console vladgh/gpg:0.2.3 --gen-key
mv /tmp/gpgkey /path/to/build_env/ds.avoton-6.2/root/.gnupg
```

Generate GPG key with gpg

```
gpg --gen-key

> Please select what kind of key you want:
  (1) RSA and RSA (default)
> choose key size and enter your name, email
> enter a passphrase: just press Enter without typing any character
```

WARNING: Please make sure that you do not type any characters in the passphrase field, otherwise the build process will **FAIL**.

After completing the steps above, the key will be generated under `~/gnupg`. You need to move them into the chroot environment.

```
cp ~/gnupg/* /toolkit/build_env/ds.${platform}-6.2/root/.gnupg/
```

You can also use the following commands to verify whether the key has successfully imported or not.

```
cd /toolkit/build_env/ds.${platform}-6.2/
chroot .
```

```
gpg -K
```

The output may produce the following message:

```
/root/.gnupg/secring.gpg
-----
sec   2048R/145E0AFD 2015-12-21
uid           Synology Inc. <synology_inc@synology.com>
ssb   2048R/E0C20F11 2015-12-21
```

Sign the package

If you want `PkgCreate.py` to sign the package automatically, you can use the `PkgCreate.py` without the `--no-sign` option. For example, the following command indicates `PkgCreate.py` to build and install your project without a signature.

```
PkgCreate.py -i ${project}
```

In addition, if you want to sign the package on your own, you can use the following command to sign your package manually.

```
chroot /toolkit/build_env/ds.${platform}-${version}
php /pkgscripts-ng/CodeSign.php [option] --sign=package-path
```

Options:

- `--keydir=keyrings` directory (default is `/root/.gnupg`)
- `--keyfpr=key's fingerprint` (default is `""`. Under this circumstances, we will using the first key in the key directory to sign the package)

Examples:

```
php /pkgscripts-ng/CodeSign.php --sign=phpBB-3.0.12-0031.spk
php /pkgscripts-ng/CodeSign.php --keydir=/root/.gpg --keyfpr=C1BF63CD --sign=phpBB-3.0.12-0031.spk
```

References

This section illustrates advanced types of usage for the Package Toolkit.

PkgCreate.py Command Option List

The following table lists some of the PkgCreate.py commands.

Option Name	Option Purpose
(default)	Run build stage only which include link and compile source code. It's the same as -U option.
-p	Specify the platform you want to pack your project.
-x	Build dependent project level. Each project is built according to their own <code>SynoBuildConf/build</code> (e.g., -x0, -x1)
-c	Run both build stage and pack stage which include link source code, compile source code, pack package and sign the final spk.
-U	Run build stage only which includes link and compile source code.
-l	Run build stage only, but will only link your source code.
-L	Run build stage only, but will compile your source code only.
-I	Run pack stage only, which will pack and sign your spk.
--no-sign	Tells PkgCreat.py not to sign your spk file. for example, PkgCreat.py -I --no-sign \${project}
-z	Run all platforms concurrently.
-J	Compile your project with -J make command options.
-S	Disable silent make.

The following table shows the relationship between command options in different stages. You can choose the proper options based on your needs. Option `-c` is enough for most cases.

Stage	Action	(default)	-l	-L	-U	-I --no-sign	-I	-c
Build Stage	Link Source code	Yes	Yes	No	Yes	No	No	Yes
Build Stage	Compile Source code	Yes	No	Yes	Yes	No	No	Yes
Pack Stage	Pack Package	No	No	No	No	Yes	Yes	Yes
Pack Stage	Sign Package	No	No	No	No	No	Yes	Yes

Platform-Specific Dependency

Platform-specific dependency means you can have several dependent projects for different platforms by appending "**:\${platform}**" to the following sections: **BuildDependent** and **ReferenceOnly**. The following example shows 816x and armada370 projects that are on libbar-1.0.

```
# SynoBuildConf/depends

[BuildDependent]
libfoo-1.0

[BuildDependent:816x,armada370]
libfoo-1.0
```



```
libbar-1.0

[default]
all="7.0"
```

Collect the SPK File in Your Own Way

By default, **PkgCreate.py** will move the SPK file to **/toolkit/result_spk** according to **/toolkit/build_env/ds.\${platform}-\${version}/source/\${project}/INFO**. You can have your own collect operation by adding a hook, **SynoBuildConf/collect**. **SynoBuildConf/collect** can be any executable shell script (so remember to `chmod +x`) and **PkgCreate.py** will pass the following environment variables to it:

- **SPK_SRC_DIR**: Source folder of target SPK file.
- **SPK_DST_DIR**: Default destination folder to put SPK file.
- **SPK_VERSION**: Version of package (according to INFO).

The current working directory of **SynoBuildConf/collect** is **/source/\${project}** will be under chroot environment.

Package Introduction

In this section, you will learn the layout of synology package (.spk) and the meaning of each file.

```

spk
├-- INFO
├-- package.tgz
├-- scripts
│   ├── postinst
│   ├── postuninst
│   ├── postupgrade
│   ├── preinst
│   ├── preuninst
│   ├── preupgrade
│   └-- start-stop-status
├-- conf
│   ├── privilege
│   └-- resource
├-- LICENSE
├-- PACKAGE_ICON.PNG
└-- PACKAGE_ICON_256.PNG

```

Package Structure

A Synology package contains the following files:

File/Folder Name (case sensitive)	Required	Description	File/Folder Type	DSM Requirement
INFO	O	This file describes the properties of a package.	Properties File	2.0-0731
package.tgz	O	This is a compressed file containing all the files that should be extracted into the system, such as executable binaries, libraries, or UI files.	TGZ File	2.0-0731
scripts	O	This folder contains shell scripts which control the lifecycle of a package.	Folder	2.0-0731
conf	O	This folder contains additional configurations.	Folder	4.2-3160
LICENSE	X	The file content will show on UI in the installation procedure. It must be less than 1 MB.	Text File	3.2-1922
PACKAGE_ICON.PNG	O	PNG format image shown in Package Center For DSM 6.x, the dimension should be 72 x 72. For DSM 7.0 or above, the image dimension should be 64 x 64.	PNG file	3.2-1922
PACKAGE_ICON_256.PNG	O	PNG format image shown in Package Center. Its dimension should be 256 x 256.	PNG file	5.0-4400

To create such package layout, please refer to the [Pack Stage](#) for detailed steps.

INFO

This file describes the properties of a package

INFO Field Format:

Each property is defined as key/value pair separated by an equals sign

```
key=value
```

INFO Field List:

You can define properties according to the requirements:

- [Necessary Fields](#)
- [Optional Fields](#)

```
thirdparty="yes"
maintainer="mycompany"
description="mydescription"
distributor="mycompany"
package="mypackagename"
silent_install="yes"
silent_uninstall="yes"
silent_upgrade="yes"
os_min_ver="7.0-40000"
version="0.0.1-0001"
arch="noarch"
```

How to write an INFO File

Instead of writing the INFO file manually, you can use the helper functions in Package Toolkit to generate some fields programmatically. Please refer to [INFO.sh](#) for more information.

Field Name: package

- **Description:** Package identity. No more than one version of a package can exist at the same time in the end user's DSM; therefore, the identification is unique to identify your package. Besides, Package Center will create a /var/packages/[package identity] folder to put package files.

Note: This value of the key cannot contain any of these special characters :, /, >, <, | or =.

- **Value:** String
- **Default Value:** (Empty)
- **Example:**

```
package="DownloadStation"
```

- **DSM Requirement:** 2.0-0731

Field Name: version

- **Description:** Package version. End users can identify the package version.

Note:

1. Each version delimiter is only allowed to be . - or _.
2. Each version number which is delimited by delimiters is only allowed to be number
3. Version value should be in the format of [feature number]-[build number]. Build number should be increased on every version and it can be used to distinguish package versions between different DSM major versions.

- **Value:** String
- **Default Value:** (Empty)
- **Example:**

```
version="3.6-3263"  
version="1.2.3-0001"
```

- **DSM Requirement:** 2.0-0731

Field Name: os_min_ver

- **Description:** Earliest version of DSM that is required to run the package. The value should be at least 7.0-40000 after DSM 7.0.
- **Value:** X.Y-Z DSM major number, DSM minor number, DSM build number
- **Default Value:** None
- **Example:**

```
os_min_ver="7.0-40000"
```

- **DSM Requirement:** 6.1-14715

Field Name: description

- **Description:** Package Center shows a short description of the package.
- **Value:** String

- **Default Value:** (Empty)
- **Example:**

```
description = "Download Station is a web-based download application which allows you to download files from the Internet through BT, FTP, HTTP, NZB, Thunder, FlashGet, QQDL, and eMule, and subscribe to RSS feeds to keep you updated on the hot test or latest BT. It offers the auto unzip service to help you extract compressed files to your Synology NAS whenever files are downloaded. With Download Station, you can download files from multiple file hosting sites, and search for torrent files via system default search engines as well as self-added engines with the BT search function."
```

- **DSM Requirement:** 2.3-1118
- **DSM Requirement:** 4.2-3160

Field Name: arch

- **Description:** List the CPU architectures which can be used to install the package.
- **Value:** (arch values are separated with a space. Please refer [Appendix A: Platform and Arch Value Mapping Table](#) to more information)
- **Default Value:** noarch

Note:

1. "noarch" means the package can be installed and work in any platform. For example, the package is written in PHP or shell script.
2. Please not pack all binary files with different platforms to one package spk file.

- **Example:**

```
arch="noarch"
or
arch="x86_64 alpine"
```

- **DSM Requirement:** 2.0-0731

Field Name: maintainer

- **Description:** Package Center shows the developer of the package.
- **Value:** String
- **Default Value:** (Empty)
- **Example:**

```
maintainer="Synology Inc."
```

- **DSM Requirement:** 2.0-0731

Field Name: `displayname`

- **Description:** Package Center shows the name of the package.

Note: If `displayname` key is empty, Package Center will display the value of `package` key.
- **Value:** String
- **Default Value:** The value of `package` key
- **Example:** None
- **DSM Requirement:** 2.3-1118

Field Name: `displayname_[DSM language]`

- **Description:** Package Center shows the name in the DSM language set by the end-user. DSM supports the following languages:
 - enu (English)
 - cht (Traditional Chinese)
 - chs (Simplified Chinese)
 - krn (Korean)
 - ger (German)
 - fre (French)
 - ita (Italian)
 - spn (Spanish)
 - jpn (Japanese)
 - dan (Danish)
 - nor (Norwegian)
 - sve (Swedish)
 - nld (Dutch)
 - rus (Russian)
 - plk (Polish)
 - ptb (Brazilian Portuguese)
 - ptg (European Portuguese)
 - hun (Hungarian)
 - trk (Turkish)
 - csy (Czech)
- **Value:** String
- **Default Value:** package name
- **Example:**

```
displayname_enu="Hello World"
displayname_cht=""
```

- **DSM Requirement:** 2.3-1118

Field Name: `description_[DSM language]`

- **Description:** Package Center shows a short description in the DSM language set by the end-user. DSM supports the following languages:
 - enu (English)
 - cht (Traditional Chinese)
 - chs (Simplified Chinese)
 - krn (Korean)

- ger (German)
- fre (French)
- ita (Italian)
- spn (Spanish)
- jpn (Japanese)
- dan (Danish)
- nor (Norwegian)
- sve (Swedish)
- nld (Dutch)
- rus (Russian)
- plk (Polish)
- ptb (Brazilian Portuguese)
- ptg (European Portuguese)
- hun (Hungarian)
- trk (Turkish)
- csy (Czech)

- **Value:** String

- **Default Value:** description

- **Example:**

```
description_enu="Hello World"
description_cht=""
```

- **DSM Requirement:** 2.3-1118

Field Name: maintainer_url

- **Description:** If a package has a developer webpage, Package Center will show a link to let the user open it.

- **Value:** String

- **Default Value:** (Empty)

- **Example:**

```
maintainer_url="http://www.synology.com"
```

- **DSM Requirement:** 4.2-3160

Field Name: distributor

- **Description:** Package Center shows the publisher of the package.

- **Value:** String

- **Default Value:** (Empty)

- **Example:**

```
distributor="Synology Inc."
```

- **DSM Requirement:** 4.2-3160

Field Name: distributor_url

- **Description:** If a package is installed and has a distributor webpage, Package Center will show a link to let the user open it.

- **Value:** String

- **Default Value:** (Empty)

- **Example:**

```
distributor_url ="http://www.synology.com/enu/apps/3rd-party_application_integration.php"
```

- **DSM Requirement:** 4.2-3160

Field Name: support_url

- **Description:** Package Center shows a support link to allow users to seek technical support when needed.

- **Value:** String

- **Default Value:** (Empty)

- **Example:**

```
support_url="https://myds.synology.com/support/support_form.php".
```

Field Name: support_center

- **Description:** If set to “yes,” Package Center displays a link to make the end user launch Synology Support Center Application when your package is installed.

Note: If set to “yes,” the **report_url** link won’t show in Package Center.

- **Value:** "yes"/"no"

- **Default Value:** "no"

- **Example:** None

- **DSM Requirement:** 5.0-4458

Field Name: model

- **Description:** List of models on which packages can be installed in specific models. It is organized by Synology string, architecture and model name.

- **Value:** (models are separated with a space, e.g. synology_88f6281_209, synology_cedarview_rs812rp+, synology_x86_411+II, synology_bromolow_3612xs, synology_cedarview_rs812rp+, ...)

- **Default Value:** (Empty)

- **Example:**

```
model="synology_bromolow_3612xs synology_cedarview_rs812rp+".
```

- **DSM Requirement:** 4.0-2219

Field Name: exclude_arch

- **Description:** List the CPU architectures where the package can't be used to install the package.

Note: Be careful to use this **exclude_arch** field. If the package has different **exclude_arch** value in the different versions, the end user can install the package in the specific version without some arch values of **exclude_arch**.

- **Value:** (arch values are separated with a space. Please refer [Appendix A: Platform and Arch Value Mapping Table](#) to more information)

- **Default Value:** (Empty)

- **Example:** None

- **DSM Requirement:** 6.0

- **Example:**

```
exclude_arch="bromolow cedarview".
```

Field Name: checksum

- **Description:** Contains MD5 string to verify the package.tgz.
- **Value:** String
- **Default Value:** (Empty)
- **Example:** None
- **DSM Requirement:** 3.2-1922

Field Name: adminport

- **Description:** A package listens to a specific port to display its own UI. If the package is defined by a port, a link will be opened when the package is started.

Note: **adminprotocol**, **adminport** and **adminurl** keys are combined to **adminprotocol://ip:adminport/adminurl** link

- **Value:** 0~65536
- **Default Value:** 80
- **Example:**

```
adminport="9002"
```

- **DSM Requirement:** 2.0-0731

Field Name: adminurl

- **Description:** If a package is installed and has a webpage, a link will be opened when the package is started.

Note: **adminprotocol**, **adminport** and **adminurl** keys are combined to **adminprotocol://ip:adminport/adminurl** link

- **Value:** String
- **Default Value:** (Empty)
- **Example:**

```
adminurl="web"
```

- **DSM Requirement:** 2.3-1118

Field Name: adminprotocol

- **Description:** A package uses a specific protocol to display its own UI. If a package is installed and has a webpage, a protocol will be opened when the package is started.

Note: **adminprotocol**, **adminport** and **adminurl** keys are combined to **adminprotocol://ip:adminport/adminurl** link

- **Value:** http / https
(Separated with a space)
- **Default Value:** http
- **Example:**

```
adminprotocol="http"
```

- **DSM Requirement:** 3.2-1922

Field Name: dsmuidir

- **Description:** DSM UI folder name in package.tgz. The UI folder of the package in `/var/packages/[package name]/target/[dsmuidir]` will be automatically linked to the DSM UI folder in `/usr/syno/synoman/webman/3rdparty/[linkname]` to show your package's shortcut in DSM.

Note:

1. If only one path is provided, the path will be the relative path to dsmuidir in package target and the link name will be package name.
2. If multiple key:value pairs are provided, the key will be the name of link and the value will be the relative path to dsmuidir in package target.
3. Please refer [Integrate Your package into DSM](#) for more information.

- **Value:** String
- **Default Value:** (Empty)
- **Example:**

```
dsmuidir="ui"
dsmuidir="MyLinkName1:ui/app1 MyLinkName2:ui/app2"
```

- **DSM Requirement:** 3.2-1922 for single value 7.0-40731 for multiple values

Field Name: dsmappname

- **Description:** The value of each individual application will be equal to the unique property name in DSM's config file so as to be integrated into Synology DiskStation.

Note: Please refer [Config](#) in [Integrate Your package into DSM](#) chapter for more information.

- **Value:** (Separated with a space)
- **Default Value:** (Empty)
- **Example:**

```
dsmappname="SYNO.SDS.PhotoStation SYNO.SDS.PersonalPhotoStation"
```

- **DSM Requirement:** 3.2-1922

Field Name: dsmapppage

- **Description:** The application page to open when click on package open button (should be used with **dsmappname** key)
- **Value:** Page name

Note: page name corresponds to PageListAppWindow's fn value when calling SYNO.SDS.AppLaunch

- **Default Value:** (Empty)
- **Example:**

```
dsmappname="SYNO.SDS.AdminCenter.Application"
dsmapppage="SYNO.SDS.AdminCenter.FileService.Main"
```

- **DSM Requirement:** 7.0-40332

Field Name: dsmapplaunchname

- **Description:** The value will be used to launch desktop app, and it has higher priority than dsmappname.
- **Value:** App name
- **Default Value:** same as dsmappname

- **Example:**

```
dsmapplaunchname="SYNO.SDS.AdminCenter.Application"
```

- **DSM Requirement:** 7.0-40796

Field Name: checkpoint

- **Description:** Check if there is any conflict between the **adminport** and the ports which are reserved or are listening on DSM except web-service ports (e.g. 80, 443) and DSM ports (e.g. 5000, 5001).
- **Value:** "yes"/"no"
- **Default Value:** "yes"
- **Example:** None
- **DSM Requirement:** 3.2-1922

Field Name: startable

- **Description:** When no program in the package provides the end-user with the options to enable or disable its function. This key is set to "no" and the end-user cannot start or stop the package in Package Center.

Note: Deprecated after 6.1-14907, use [ctl_stop](#) instead.

If “startable” is set to “no”, **start-stop-status** script which runs in bootup or shutdown is still required.

- **Value:** "yes"/"no"
- **Default Value:** "yes"
- **Example:** None
- **DSM Requirement:** 3.2-1922

Field Name: ctl_stop

- **Description:** When no program in the package provides the end-user with the options to enable or disable its function. This key is set to "no" and the end-user cannot start or stop the package in Package Center.

Note: If “ctl_stop” is set to “no”, **start-stop-status** script which runs in bootup or shutdown is still required.

- **Value:** "yes"/"no"
- **Default Value:** "yes"
- **Example:** None
- **DSM Requirement:** 6.1-14907

Field Name: ctl_uninstall

- **Description:** If this key is set to "no", the end-user cannot uninstall the package in Package Center.
- **Value:** "yes"/"no"
- **Default Value:** "yes"
- **Example:** None
- **DSM Requirement:** 6.1-14907

Field Name: precheckstartstop

- **Description:** If set to "yes", let start-stop-status with prestart or prestop argument run before start or stop the package. Please refer to start-stop-status in [scripts](#) for more information.
- **Value:** "yes"/"no"

- **Default Value:** "yes"
- **Example:** None
- **DSM Requirement:** 6.0

Field Name: helpurl

- **Description:** If a package is installed and has a "help" webpage, Package Center will display a hyperlink to the user.
- **Value:** String
- **Default Value:** (Empty)
- **Example:**

```
helpurl="https://www.synology.com/en-global/knowledgebase"
```

- **DSM Requirement:** 3.2-1922

Field Name: beta

- **Description:** If this package is considered the beta version, the beta information will be shown in Package Center.
- **Value:** "yes"/"no"
- **Default Value:** "no"
- **Example:** None
- **DSM Requirement:** 6.0

Field Name: report_url

- **Description:** If a package is a beta version and has a "report" webpage, Package Center will display a hyperlink. If this package is considered the beta version, the beta information will be also be shown in Package Center.
- **Value:** String
- **Default Value:** (Empty)
- **Example:** None
- **DSM Requirement:** 3.2-1922

Field Name: install_reboot

- **Description:** Reboot DiskStation after installing or upgrading the package.
- **Value:** "yes"/"no"
- **Default Value:** "no"
- **Example:** None
- **DSM Requirement:** 3.2-1922

Field Name: install_dep_packages

- **Description:** Before a package is installed or upgraded, these packages must be installed first. In addition, the order of starting or stopping packages is also dependent on it. The format consists of a package name. If more than one dependent packages are required, the package name of the package(s) will be separated with a colon, e.g. `install_dep_packages="packageA"`. If a specific version range is required, package name will be followed by one of the special characters =, <, >, >=, <= and package version which is composed by number and periods, e.g. `install_dep_packages="packageA>2.2.2:packageB"`.

Note: >= and <= operator only supported in DSM 4.2 or newer. Don't use <= and >= if a package can be installed in DSM 4.1 or older because it cannot be compared correctly. Instead, the package version should be set lower or higher.

- **Value:** Package names

Note: Each package name is separated with a colon.

- **Default Value:** (Empty)

- **Example:**

```
install_dep_packages="packageA"  
or  
install_dep_packages="packageA>2.2.2:packageB"
```

- **DSM Requirement:** 3.2-1922

Field Name: `install_conflict_packages`

- **Description:** Before your package is installed or upgraded, these conflict packages cannot be installed. The format consists of a package name, e.g. `install_conflict_packages="packageA"`. If more than one conflict packages are required with the format, the name of the package(s) will be separated with a colon, e.g. `install_conflict_packages="packageA:packageB"`. If a specific version range is required, package name will be followed by one of the special characters `=`, `<`, `>`, `>=`, `<=` and package version which is composed by number and periods, e.g. `install_conflict_packages="packageA>2.2.2:packageB"`.

Note: `>=` and `<=` operator only supported in DSM 4.2 or newer. Do not use `<=` and `>=` if a package can be installed in DSM 4.1 because it can't be compared correctly. Instead, the package version should be set lower or higher.

- **Value:** Package names

Note: Each package name is separated with a colon.

- **Default Value:** (Empty)

- **Example:**

```
install_conflict_packages="packageA:packageB"  
or  
install_conflict_packages="packageA>2.2.2:packageB"
```

- **DSM Requirement:** 4.1-2851

Field Name: `install_break_packages`

- **Description:** After your package is installed or upgraded, these to-be-broken packages will be stopped and remain broken during the existence of your package. The format consists of a package name, e.g. `install_break_packages="packageA"`. If more than one to-be-broken packages are required with the format, the name of the package(s) will be separated with a colon, e.g. `install_break_packages="packageA:packageB"`. If a specific version range is required, package name will be followed by one of the special characters `=`, `<`, `>`, `>=`, `<=` and package version which is composed by number and periods, e.g. `install_break_packages="packageA>2.2.2:packageB"`.

- **Value:** Package names

Note: Each package name is separated with a colon.

- **Default Value:** (Empty)

- **Example:**

```
install_break_packages="packageA:packageB"  
or  
install_break_packages="packageA>2.2.2:packageB"
```

- **DSM Requirement:** 6.1-15117

Field Name: `install_replace_packages`

- **Description:** After your package is installed or upgraded, these to-be-replaced packages will be removed. The format consists of a package name, e.g. `install_replace_packages="packageA"`. If more than one to-be-replaced packages are required with the format, the name of the package(s) will be separated with a colon, e.g. `install_replace_packages="packageA:packageB"`. If a specific version range is required, package name will be followed by one of the special characters `=`, `<`, `>`, `>=`, `<=` and package version which is composed by number and periods, e.g. `install_replace_packages="packageA>2.2.2:packageB"`.

- **Value:** Package names

Note: Each package name is separated with a colon.

- **Default Value:** (Empty)

- **Example:**

```
install_replace_packages="packageA:packageB"
or
install_replace_packages="packageA>2.2.2:packageB"
```

- **DSM Requirement:** 6.1-15117

Field Name: `install_dep_services`

- **Description:** Before the package is installed or upgraded, these services must be started or enabled by the end-user.

- **Value:**

DSM 4.2 or older: `apache-web`, `mysql`, `php_disable_safe_exec_dir`

DSM 4.3: `apache-web`, `mysql`, `php_disable_safe_exec_dir`, `ssh`

DSM 5.0 ~ DSM 5.2: `apache-web`, `php_disable_safe_exec_dir`, `ssh`, `pgsql`

DSM 6.0: `ssh`, `pgsql`

DSM 7.0: `ssh-shell`, `pgsql`, `network.target`, `network-online.target`, `nginx.service`, `avahi.service`, `atalk.service`, `crond.service`, `nfs-server.service`

Note: Each service is separated with a space.

- **Default Value:** (Empty)

- **Example:**

```
install_dep_services="apache-web ssh"
```

- **DSM Requirement:** 3.2-1922

Field Name: `start_dep_services`

- **Description:** Before the package is started, these services must be started or enabled by the end-user. If `startable` is set to “no”, this value is ignored.

- **Value:**

DSM 4.2 or older: `apache-web`, `mysql`, `php_disable_safe_exec_dir`

DSM 4.3: `apache-web`, `mysql`, `php_disable_safe_exec_dir`, `ssh`

DSM 5.0 ~ DSM 5.2: `apache-web`, `php_disable_safe_exec_dir`, `ssh`, `pgsql`

DSM 6.0: `ssh`, `pgsql`

DSM 7.0: `ssh-shell`, `pgsql`, `network.target`, `network-online.target`, `nginx.service`, `avahi.service`, `atalk.service`, `crond.service`, `nfs-server.service`

Note: Each service is separated with a space.

- **Default Value:** (Empty)

- **Example:**

```
install_dep_services="apache-web ssh"
```

- **DSM Requirement:** 3.2-1922

Field Name: extractsize

- **Description:** This value indicates the minimal space to install a package. It will be used to prompt the user if there is enough free space to install it.

Note:

1. In DSM 5.2 or order, the size based on byte unit.
2. In DSM 6.0 or newer, the size based on kilobyte unit.

- **Value:** Size unit
- **Default Value:** The byte size of SPK file of package
- **Example:**

```
extractsize="253796"
```

- **DSM Requirement:** 4.0-2166

Field Name: support_conf_folder

- **Description:** In DSM 5.2 or order, if you want to use some special configuration files within a "**conf**" folder, this value must be set to "**yes**". More details are given in the "**conf**" section. However, in DSM 6.0 or newer, you don't need to define it anymore.

Note: Deprecated in DSM 6.0

- **Value:** "yes"/"no"
- **Default Value:** "no"
- **Example:**

```
support_conf_folder="yes"
```

- **DSM Requirement:** 4.2-3160 ~ 5.2

Field Name: install_type

- **Description:** If set to "system", your package will be installed in the root file system, `/usr/local/packages/@appstore/`, even if there is no volume.

Note: Be careful when setting this, as it may result in the DiskStation crashing if your package runs out of the space in the root file system.

- **Value:** "system"
- **Default Value:** (Empty)
- **Example:**

```
install_type="system"
```

- **DSM Requirement:** 5.0-4458

Field Name: silent_install

- **Description:** If set to "yes", your package is allowed to be installed without the package wizard in the background. This allows CMS (Central Management System) to distribute package installation to other NAS connected.

- **Value:** "yes"/"no"
- **Default Value:** "no"
- **Example:**

```
silent_install="yes"
```


- **DSM Requirement:** 5.0-4458

Field Name: `silent_upgrade`

- **Description:** If set to “yes”, your package is allowed to be upgraded without the package wizard in the background. End user cannot modify any information for upgrading. This allows not only your package to be upgraded automatically but also for CMS (Central Management System) to distribute package upgrades to other NAS connected.
- **Value:** "yes"/"no"
- **Default Value:** "no"
- **Example:**

```
silent_upgrade="yes"
```

- **DSM Requirement:** 5.0-4458

Field Name: `silent_uninstall`

- **Description:** If set to “yes”, your package is allowed to be uninstalled without the package wizard in the background. This allows CMS (Central Management System) to distribute package uninstallation to other NAS connected.
- **Value:** "yes"/"no"
- **Default Value:** "no"
- **Example:**

```
silent_uninstall="yes"
```

- **DSM Requirement:** 5.0-4458

Field Name: `auto_upgrade_from`

- **Description:** It is set to a version of your package. If your package is set to **`silent_upgrade="yes"`** and the value is set, Package Center only upgrades your package automatically from the installed package with the version or the newer version. However, if the end user install a older version than it, Package Center won't upgrade it automatically and the user must upgrade it by themselves.
- **Value:** (a package version)
- **Default Value:** (Empty string)
- **Example:**

```
auto_upgrade_from="2.0"
```

- **DSM Requirement:** 5.2-5565

Field Name: `offline_install`

- **Description:** If set to "yes", after the package is published in synology server, it won't be shown in the package list of Package Center from Synology server. However, the user can install the package manually.
- **Value:** "yes"/"no"
- **Default Value:** "no"
- **Example:**

```
offline_install="yes"
```

- **DSM Requirement:** DSM 6.0

Field Name: `thirdparty`

- **Description:** If set to “yes”, your package is a third-party package and isn't developed by Synology. In Package Center, third-party packages will be shown in another part.

Note: It's not used in DSM 5.0 or newer.

- **Value:** "yes"/"no"
- **Default Value:** "no"
- **Example:**

```
thirdparty="yes"
```

- **DSM Requirement:** 4.0~4.3

Field Name: `os_max_ver`

- **Description:** Maximum version of DSM that is capable to run the package.
- **Value:** X.Y-Z DSM major number, DSM minor number, DSM build number
- **Default Value:** None
- **Example:**

```
os_max_ver="6.1-14715"
```

- **DSM Requirement:** 6.1-14715

Field Name: `support_move`

- **Description:** If set to "yes", the package can be moved to a different volume after installation.
- **Value:** "yes"/"no"
- **Default Value:** "no"
- **Example:**

```
support_move="yes"
```

- **DSM Requirement:** 6.2-22306

Field Name: `exclude_model`

- **Description:** List the model names where the package can't be used to install the package.

Note: Be careful to use this `exclude_model` field. If the package has different `exclude_model` value in the different versions, the end user can install the package in the specific version without some model values of `exclude_model`.

- **Value:** model values are separated with a space.
- **Default Value:** (Empty)
- **Example:**

```
exclude_model="synology_cedarview_713+ synology_kvmx64_virtualdsm"
```

- **DSM Requirement:** 7.0-40329

Field Name: `use_deprecated_replace_mechanism`

- **Description:** if set to "yes", replacee will be uninstalled after replacer installed, and prereplace / postreplace scripts will not be

executed. Otherwise, replacee will be uninstalled before replacer installed, and prereplace / postreplace will be executed.

- **Value:** "yes"/"no"
- **Default Value:** "no"
- **Example:**

```
install_replace_packages="packageA"  
use_deprecated_replace_mechanism="yes"
```

- **DSM Requirement:** 7.0-40340

Field Name: `install_on_cold_storage`

- **Description:** if set to "yes", this package can be installed on cold storage, which has very large space for data storage.
- **Value:** "yes"/"no"
- **Default Value:** "no"
- **Example:**

```
install_on_cold_storage="yes"
```

- **DSM Requirement:** 7.0-40726

package.tgz

The **package.tgz** is a compressed file (tgz / xz) containing all the files you would need when bringing up your applications such as:

- executable files
- library files
- UI files
- configuration files

You can use [pkg_make_package](#) function to create the **package.tgz** instead of packing it manually.

Once the package is installed, your package.tgz will be extracted to `/volume?/@appstore/[your_pkg_name]/` or `/usr/local/packages/@appstore/[your_pkg_name]/` folder (depending on the *install_type* in INFO). In the meantime, there will be a soft link at `/var/packages/[your_pkg_name]/target` pointing to the assigned folder.

In addition to the `target` directory, system will also create other directories for package to store its data for different purposes.

Detailed information can be found [HERE](#).

scripts

This folder contains shell scripts controlling the lifecycle of a package.

Script Name	Required	Description
preinst	O	It can be used to check conditions before installation but not to make side effects onto the system. Package installation will be aborted for non-zero returned value.
postinst	O	It can be used to prepare environment for package after installed. Package status will become corrupted for non-zero returned value.
preuninst	O	It can be used to check conditions before uninstallation but not to make side effects onto the system. Package uninstallation will be aborted for non-zero returned value.
postuninst	O	It can be used to cleanup environment for package after uninstalled.
preupgrade	O	It can be used to check conditions before upgrade but not to make side effects onto the system. Package upgrade will be aborted for non-zero returned value.
postupgrade	O	It can be used to prepare environment for package after upgraded. Package status will become corrupted for non-zero returned value.
prereplace	X	It can be used to do data migration when install_replace_packages is defined in <code>INFO</code> for package replacement. Package replacement will be aborted for non-zero returned value.
postreplace	X	It can be used to do data migration when install_replace_packages is defined in <code>INFO</code> for package replacement. Package replacement will be aborted for non-zero returned value.
start-stop-status	O	It can be used to control package lifecycle.

The simplest implementation of script is just doing nothing:

```
#!/bin/sh

exit 0
```

Please refer to [Script Messages](#) for mechanism to show messages to users.

start-stop-status

```
#!/bin/sh

case "$1" in
  start)
    ;;
  stop)
    ;;
  status)
    ;;
  esac

exit 0
```

This script is used to start, stop a package and detect running status. DSM would call this script with different parameters in different scenario:

- **start:** When a user runs the package or the system is turning on, the package should do its start operation.
- **stop:** When a user stops the package or the system is turning off, the package should do its stop operation.

- **status:** When the package status is being checked, the following exit codes should be returned according to its status:

```
0: package is running.
1: program of package is dead and /var/run pid file exists.
2: program of package is dead and /var/lock lock file exists
3: package is not running
4: package status is unknown
150: package is broken and should be reinstalled.
```

- **prestart:** If `precheckstartstop` in `INFO` is set to `yes`, the package could check if it is allowed to be started.

Note: It will also run before starting a package at booting up after DSM 7.0.

- **prestop:** If `precheckstartstop` in `INFO` is set to `yes`, the package could check if it is allowed to be stopped.

Note: It won't run before stopping a package at shutting down.

Execution Order

Installation

1. `prereplace`
2. `preinst`
3. `postinst`
4. `postreplace`
5. `start-stop-status` with `prestart` argument if end user chooses to start it immediately
6. `start-stop-status` with `start` argument if end user chooses to start it immediately

Upgrade

1. `start-stop-status` with `prestop` argument if it has been started (old)
2. `start-stop-status` with `stop` argument if it has been started (old)
3. `preupgrade` (new)
4. `preuninst` (old)
5. `postuninst` (old)
6. `prereplace` (new)
7. `preinst` (new)
8. `postinst` (new)
9. `postreplace` (new)
10. `postupgrade` (new)
11. `start-stop-status` with `prestart` argument if it was started before being upgraded (new)
12. `start-stop-status` with `start` argument if it was started before being upgraded (new)

Uninstallation

1. `start-stop-status` with `prestop` argument if it has been started
2. `start-stop-status` with `stop` argument if it has been started
3. `preuninst`
4. `postuninst`

Start

1. `start-stop-status` with `prestart` argument
2. `start-stop-status` with `start` argument

Stop

1. start-stop-status with prestop argument
2. start-stop-status with stop argument

Script Environment Variables

Several variables are exported by Package Center and can be used in the scripts. Descriptions of the variables are given as below:

- **SYNOPKG_PKGNAME**: Package identify which is defined in **INFO**.
- **SYNOPKG_PKGVER**: Package version which is defined in **INFO**. The value will be new version of package when it is upgrading.
- **SYNOPKG_PKGDEST**: Target directory where the package is stored.
- **SYNOPKG_PKGDEST_VOL**: Target volume where the package is stored.
- **SYNOPKG_PKGPORT**: **adminport** port which is defined in **INFO**. This port will be occupied by this package with its management interface.
- **SYNOPKG_PKGINST_TEMP_DIR**: The temporary directory where the package are extracted when installing or upgrading it.
- **SYNOPKG_TEMP_LOGFILE**: A temporary file path for a script to log information or error messages.
- **SYNOPKG_TEMP_UPGRADE_FOLDER**: The temporary directory when the package is upgrading. You can move the files from the previous version of the package to it in **preupgrade** script and move them back in **postupgrade**.
- **SYNOPKG_DSM_LANGUAGE**: End user's DSM language.
- **SYNOPKG_DSM_VERSION_MAJOR**: End user's major number of DSM version which is formatted as [DSM major number].[DSM minor number].[DSM build number].
- **SYNOPKG_DSM_VERSION_MINOR**: End user's minor number of DSM version which is formatted as [DSM major number].[DSM minor number].[DSM build number].
- **SYNOPKG_DSM_VERSION_BUILD**: End user's DSM build number of DSM version which is formatted as [DSM major number].[DSM minor number].[DSM build number].
- **SYNOPKG_DSM_ARCH**: End user's DSM CPU architecture. Please refer [Appendix A: Platform and Arch Value Mapping Table](#) to more information
- **SYNOPKG_PKG_STATUS**: Package status presented by these values: INSTALL, UPGRADE, UNINSTALL, START, STOP or empty.
 1. INSTALL will be set as the status value in the **preinst** and **postinst** scripts while the package is installing. If the user chooses to “start after installation” at the last step of the installation wizard, the value will be set to INSTALL in the **start-stop-status** script when the package is started.
 2. UPGRADE will be set as the status value in the **preupgrade**, **preuninst**, **postunist**, **preinst**, **postinst** and **postupgrade** scripts sequentially while the package is upgrading. If the package has already started before upgrade, the value will be set to UPGRADE in the **start-stop-status** script when the package is started or stopped.
 3. UNINSTALL will be set as the status value in the **preuninst** and **postunist** scripts while the package is un-installing. If the package has already started before un-installation, the value will be set to UNINSTALL in the **start-stop-status** script when the package is stopped.
 4. If the user starts or stops a package in the Package Center, START or STOP will be set as the status value in the **start-stop-status** script.
 5. When the NAS is booting up or shutting down, its status value will be empty.
- **SYNOPKG_OLD_PKGVER**: Old package version which is defined in **INFO** during upgrading.
- **SYNOPKG_TEMP_SPKFILE**: The location of package spk file is temporarily stored in DS when the package is installing/upgrading.
- **SYNOPKG_USERNAME**: The user name who installs, upgrades, uninstalls, starts or stops the package. If the value is empty, the action is triggered by DSM, not by the end user.
- **SYNOPKG_PKG_PROGRESS_PATH**: A temporary file path for a script to showing the progress in installing and upgrading a package.

Note:

1. The progress value is between 0 and 1.
2. Example:

```
flock -x "$SYNOPKG_PKG_PROGRESS_PATH" -c echo 0.80 > "$SYNOPKG_PKG_PROGRESS_PATH"
```


Show Messages to Users

Show Message as Script Result

If you want to send a prompt users with **messages after they installed, upgraded, uninstalled, started, or stopped a package**, you can use the `$SYNOPKG_TEMP_LOGFILE` variable in related scripts. For example:

```
echo "Hello World!!" > $SYNOPKG_TEMP_LOGFILE
```

If you want to prompt users according to their language, you can use `$SYNOPKG_DSM_LANGUAGE` variable for language abbreviation as shown in the example below:

```
case $SYNOPKG_DSM_LANGUAGE in
    chs)
        echo "" > $SYNOPKG_TEMP_LOGFILE
        ;;
    cht)
        echo "" > $SYNOPKG_TEMP_LOGFILE
        ;;
    csy)
        echo "Český" > $SYNOPKG_TEMP_LOGFILE
        ;;
    dan)
        echo "Dansk" > $SYNOPKG_TEMP_LOGFILE
        ;;
    enu)
        echo "English" > $SYNOPKG_TEMP_LOGFILE
        ;;
    fre)
        echo "Français" > $SYNOPKG_TEMP_LOGFILE
        ;;
    ger)
        echo "Deutsch" > $SYNOPKG_TEMP_LOGFILE
        ;;
    hun)
        echo "Magyar" > $SYNOPKG_TEMP_LOGFILE
        ;;
    ita)
        echo "Italiano" > $SYNOPKG_TEMP_LOGFILE
        ;;
    jpn)
        echo "" > $SYNOPKG_TEMP_LOGFILE
        ;;
    krn)
        echo "" > $SYNOPKG_TEMP_LOGFILE
        ;;
    nld)
        echo "Nederlands" > $SYNOPKG_TEMP_LOGFILE
        ;;
    nor)
        echo "Norsk" > $SYNOPKG_TEMP_LOGFILE
        ;;
    plk)
        echo "Polski" > $SYNOPKG_TEMP_LOGFILE
        ;;
    ptb)
        echo "Português do Brasil" > $SYNOPKG_TEMP_LOGFILE
        ;;
    ptg)
        echo "Português Europeu" > $SYNOPKG_TEMP_LOGFILE
        ;;
    rus)
        echo "Русский" > $SYNOPKG_TEMP_LOGFILE
        ;;
    ;;
```

```
spn)
    echo "Español" > $SYNOPKG_TEMP_LOGFILE
;;
sve)
    echo "Svenska" > $SYNOPKG_TEMP_LOGFILE
;;
trk)
    echo "Türkçe" > $SYNOPKG_TEMP_LOGFILE
;;
*)
    echo "English" > $SYNOPKG_TEMP_LOGFILE
;;
esac
```

Please refer to ["scripts"](#) and ["Script Environment Variables"](#) sections for more information.

Show Message as Desktop Notification

It is possible to use `/usr/syno/bin/synodsmnotify` executable to **send desktop notifications to users**. The notification title / message must be an [I18N](#) string.

```
/usr/syno/bin/synodsmnotify -c [app_id] [user_or_group] [i18n_string_for_title] [i18n_string_for_msg]
```

```
/usr/syno/bin/synodsmnotify -c com.company.App1 admin MyPackage:app_tree:index_title MyPackage:app_tree:node_1
```

```
/usr/syno/bin/synodsmnotify -c com.company.App1 @administrators MyPackage:app_tree:index_title MyPackage:app_tree:node_1
```

Notification title and message here should be in the format of `[package_id]:[i18n_section]:[i18n_key]` where `package_id` is the `package` value in package `INFO` file. I18N string example can be found in [I18N](#) page. Remember to specify desktop notification strings to `preloadTexts` field in [application config](#).

conf

The **conf** folder contains the following files:

File/Folder Name	Required	Description	File/Folder Type	DSM Requirement
PKG_DEPS	X	Define dependency between packages with restrictions of DSM version.	File	4.2-3160
PKG_CONX	X	Define conflicts between packages with restrictions of DSM version.	File	4.2-3160
privilege	O	Define file privilege and execution privilege to secure the package.	File	6.2-5891
resource	X	Define system resources that can be used in the lifecycle of package.	File	6.2-5941

Since DSM 7.0, all packages are forced to lower the privilege explicitly. The `privilege` must be provided for package to work.

Privilege

DSM 7.0, packages are forced to lower the privilege by applying privilege mechanism explicitly.

To reduce security risks, package should run as an user rather than `root`. Package can apply such mechanism by providing a configuration file named `privilege`:

With the configuration, package developer is capable to

- Control default user / group name of process in `scripts`
- Control permission of files in `package.tgz`
- Control file capabilities in `package.tgz`
- Control if special system resources are accessible

To overcome the limitation that normal user cannot be used to do privileged operations, we provide a way for package to request system resources. Please refer to [Resource](#) for more information.

Setup privilege configuration

Just create a file at `conf/privilege` with [preferred configuration](#).

```
{
  "defaults": {
    "run-as": "package"
  }
}
```

Resource

Packages can obtain system resources even in lower privilege identity if they apply this mechanism.

Steps to setup resource config

1. Find out the resources you want from [Resource List](#)
2. Check if the corresponding [Timing](#) of selected resource is satisfied.
3. Create a file at `conf/resource` with [preferred configuration](#).

```
{
  "data-share": {
    "shares": [
      {
        "name": "MyShareFolderName",
        "permission": {
          "ro": ["MyUserName"]
        }
      }
    ]
  }
}
```

The instance handling the resource request is called `worker`.

PKG_DEPS

The PKG_DEPS is similar to **install_dep_packages** key in **INFO** file, but it additionally defines the restriction according to specific DSM versions.

priority of `PKG_DEPS` is higher than `install_dep_packages` in `INFO`

Each configuration file is defined in standard **.ini** file format with key/value pairs and sections. A section describes a unique name of dependent/conflicting package. Each section contains information about the requirements of package versions and the restriction of DSM versions.

Key	Description	Value
pkg_min_ver	Minimum version of dependent package.	Package version
pkg_max_ver	Maximum version of dependent package.	Package version
dsm_min_ver	Minimum required DSM version.	X.Y-Z DSM major number, DSM minor number, DSM build number
dsm_max_ver	Maximum required DSM version.	X.Y-Z DSM major number, DSM minor number, DSM build number

```
; Your package depends on Package A in any version
[Package A]

; Your package depends on Package B version 2 or newer
[Package B]
pkg_min_ver=2

; Your package depends on Package C with version 2 or older
[Package C]
pkg_max_ver=2

; Your package depends on Package D with version 2 or newer but it will be ignored when DSM version is smaller than 4.1-2668
[Package D]
dsm_min_ver=4.1-2668
pkg_min_ver=2

; Your package depends on Package E with version 2 or newer but it will be ignored when DSM version is bigger than 4.1-2668
[Package E]
dsm_max_ver=4.1-2668
pkg_min_ver=2
```

PKG_CONX

The PKG_CONX is similar to **install_conflict_packages** key in **INFO** file, but it additionally defines the restriction according to specific DSM versions.

priority of `PKG_CONX` is higher than `install_conflict_packages` in `INFO`

Each configuration file is defined in standard **.ini** file format with key/value pairs and sections. A section describes a unique name of dependent/conflicting package. Each section contains information about the requirements of package versions and the restriction of DSM versions.

Key	Description	Value
pkg_min_ver	Minimum version of conflicting package.	Package Version
pkg_max_ver	Maximum version of conflicting package.	Package Version
dsm_min_ver	Minimum required DSM version.	X.Y-Z DSM major number, DSM minor number, DSM build number
dsm_max_ver	Maximum required DSM version.	X.Y-Z DSM major number, DSM minor number, DSM build number

```
; Your package conflicts with Package A in any version
[Package A]

; Your package conflicts with Package B version 2 or newer
[Package B]
pkg_min_ver=2

; Your package conflicts with Package C version 2 or older
[Package C]
pkg_max_ver=2

; Your package conflicts with Package D version 2 or newer, but it will be ignored when DSM version is smaller than 4.1-2668
[Package D]
dsm_min_ver=4.1-2668
pkg_min_ver=2

; Your package conflict on Package E with version 2 or newer but it will be ignored when DSM version is bigger than 4.1-2668
[Package E]
dsm_max_ver=4.1-2668
pkg_min_ver=2
```

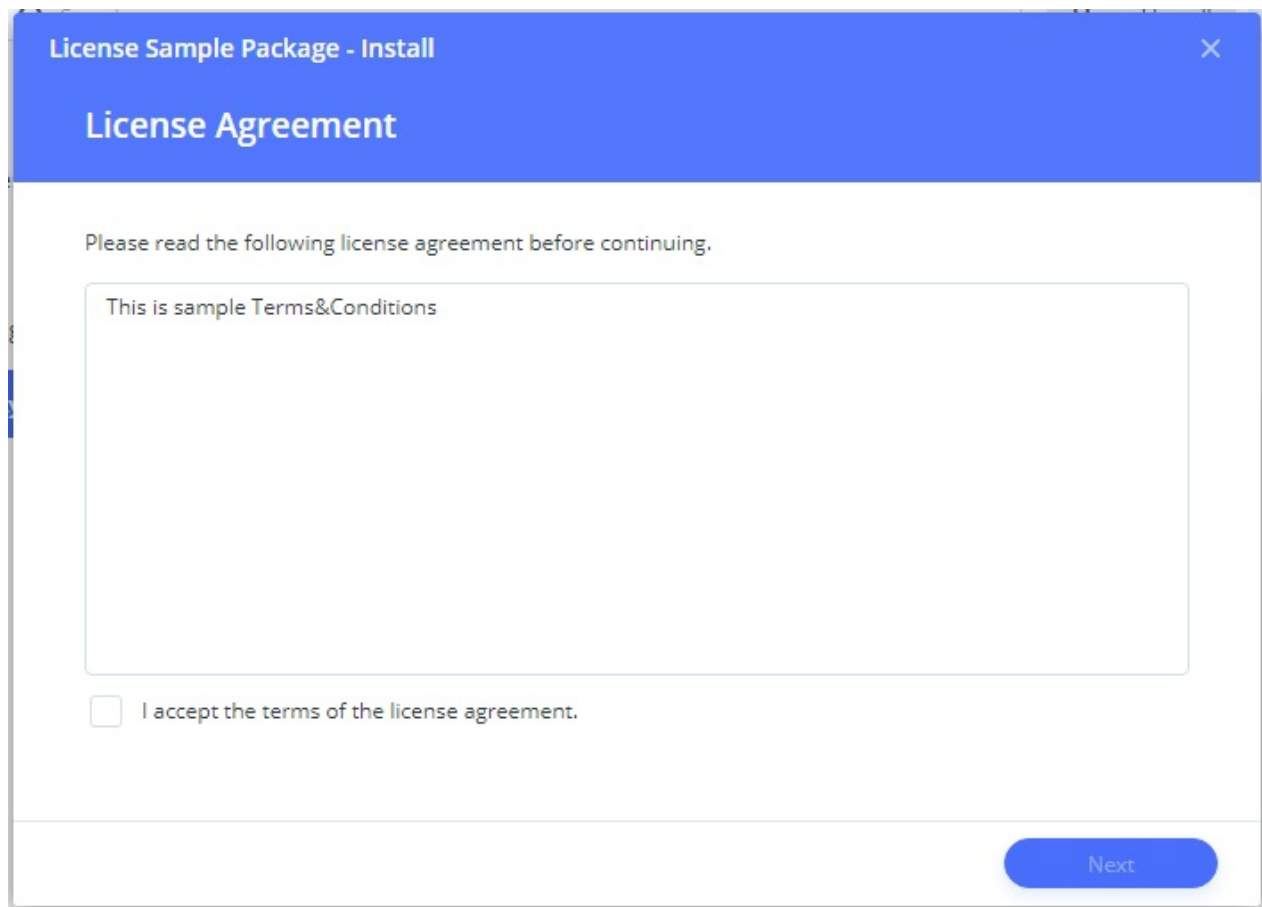

License

The *LICENSE* file contains the licenses / user terms & conditions / end user agreements to show on the package installation wizard. The package center would open up a dialog to show the content of *LICENSE* file and provide a checkbox for user to agree these terms on the dialog.

How to place LICENSE file

Prepare a file containing your terms & conditions in plain text format then put it to `/` of your package (the directory where the INFO is)

If the *LICENSE* file is properly put inside spk, the installation wizard would show your license file content like this:



Synology DSM Integration

Package Filesystem Hierarchy Standard

After the package installed, there will be some directories for package to put their data. There will be **different directories linked for packages who installed on volume partition / system partition.**

```
/var/packages/[package_name]
├─ etc      -> /volume[volume_number]/@appconf/[package_name] (move to volume since 7.0-41330, and old path still works)
├─ var      -> /volume[volume_number]/@appdata/[package_name]
├─ tmp      -> /volume[volume_number]/@apptemp/[package_name]
├─ home     -> /volume[volume_number]/@apphome/[package_name]
└─ target   -> /volume[volume_number]/@appstore/[package_name]
```

```
/var/packages/[package_name]
├─ etc      -> /usr/syno/etc/packages/[package_name]
├─ var      -> /usr/local/packages/@appdata/[package_name]
├─ tmp      -> /usr/local/packages/@apptemp/[package_name]
├─ home     -> /usr/local/packages/@apphome/[package_name]
└─ target   -> /usr/local/packages/@appstore/[package_name]
```

Please refer to [install_type](#) in [INFO](#) for more information about installation on volume / system partition.

Directory	Purpose	Mode	Creation Timing	Remove Timing	Script Variable
etc	permanant config storage	0755	installed / up graded	none	none
var (since 7.0-40314)	permanant data storage	0755	installed / up graded	none	SYNOPKG_PKGVAR
tmp (since 7.0-40356)	temporary data storage	0755	installed / up graded	uninstalled / up grading	SYNOPKG_PKTGMTMP
home (since 7.0-40759)	private storage	0700	installed / up graded	none	SYNOPKG_PKGHOME
target	data extracted from package.tgz	0755	installed / up graded	uninstalled / up grading	SYNOPKG_PKGDEST

Directory Owner Rules

- When defaults run-as is package, FHS directories are set to `[packageuser]:[packagegroup]`
- When defaults run-as is root, FHS directories are set to `root:[packagegroup]`

Please refer to [Privilege](#) section for more information about defaults run-as.

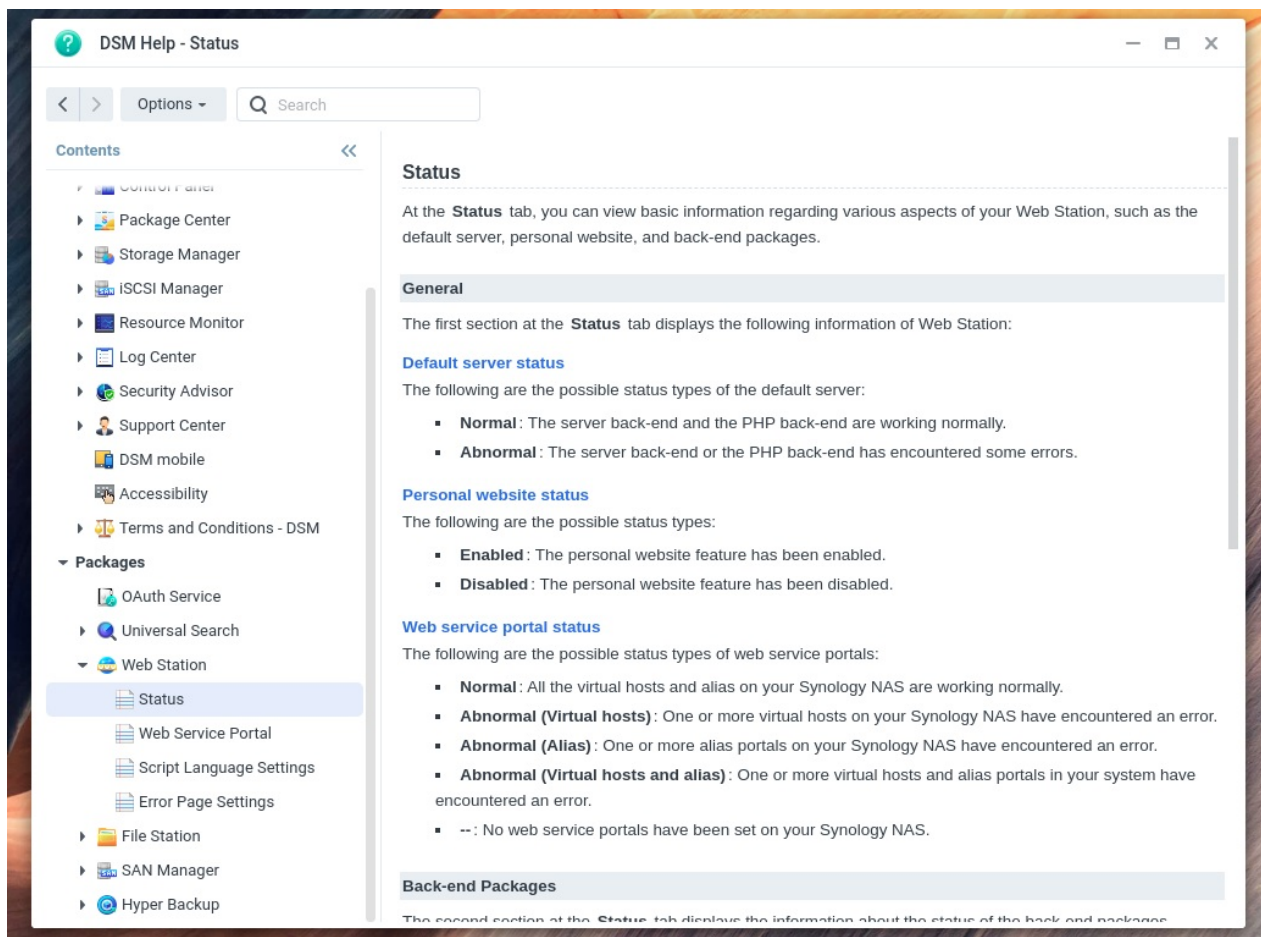
Desktop Application

You can provide a App Config for your package so that the configured application will show on the menu of desktop. It is possible to customize icon, application privilege and target url.

To distinguish different role of users, **one package can even provide more than one application** such as admin application for administrators and normal application for normal users.



In addition, any application can bring its own help documents into desktop by providing a Help Config.



Steps to setup desktop application

1. Create a directory inside `package.tgz`, this directory will be used to store desktop application configs. We name it as `ui` for example here.
2. Add `dsmuidir` key to your **INFO** or **INFO.sh** whose value is the relative path to the directory you just created on previous step.

```
dsmuidir="ui"
```

```
dsmuidir="MyApp1:appui1 MyApp2:appui2"
```

If you have multiple applications, the second form should be applied. In the example above, `MyApp1` represents an identifier and `appui1` represents a relative path.

Once the package is installed, DSM will create corresponding soft link at `/usr/syno/synoman/webman/3rdparty/[identifier]/` linking to the path where your relative path is. When the identifier is not presented in the first form, DSM will use package name as identifier by default.

3. Create your own **App Config** and **Help Config** under the directory specified by `dsmuidir` if necessary.
4. Add `dsmappname` key to your **INFO** or **INFO.sh** whose value is the unique application name inside App Config. This application will be the target application when open button of package is clicked in package center.

```
dsmappname="com.company.App1"
```


Application Config

To integrate desktop applications into DSM, you have to provide a `config` file in JSON format under the directory specified by `dsmuidir` in `INFO`.

```
{
  ".url": {
    "com.company.App1": {
      "type": "url",
      "icon": "images/app_{0}.png",
      "title": "Test App1",
      "desc": "Description",
      "url": "http://www.yahoo.com",
      "allUsers": true,
      "preloadTexts": [
        "app_tree:index_title",
        "app_tree:node_1"
      ]
    },
    "com.company.App2": {
      "type": "legacy",
      "icon": "images/app2_{0}.png",
      "title": "Test App2",
      "desc": "Description 2",
      "url": "http://www.synology.com",
      "allUsers": true
    }
  }
}
```

Property	Required	Description
com.company.App1 com.company.App2	O	In “.url” , each object should have a unique property name.
type	O	When you click the menu item, the address you use to connect to the DSM management UI will be shown in the right frame of the management UI. However, you can customize the address as you wish. The “type” value can be “url” or “legacy” . “url” means when you click the application icon, the URL will be opened in a pop-up window, while “legacy” implies that the URL will be opened in an iframe window application. You can follow the descriptions below to set up your customized URL.
icon	O	<p>“icon” indicates the icon for the application. It is a template string. The “{0}” can be replaced by “16”, “24”, “32”, “48”, “64”, “72”, “256” depending on the resolution of the icon.</p> <p>The icon must be saved under <code>/usr/syno/synoman/webman/3rdparty/xxx/</code> where <code>xxx</code> is the directory name of your package.</p> <p>For example, if you create a directory named <code>images</code> and put the icon image file “icon.png” in it, the full path for the icon would be:</p> <pre>/usr/syno/synoman/webman/3rdparty/xxx/images/icon_16.png /usr/syno/synoman/webman/3rdparty/xxx/images/icon_24.png /usr/syno/synoman/webman/3rdparty/xxx/images/icon_32.png /usr/syno/synoman/webman/3rdparty/xxx/images/icon_48.png /usr/syno/synoman/webman/3rdparty/xxx/images/icon_64.png /usr/syno/synoman/webman/3rdparty/xxx/images/icon_72.png /usr/syno/synoman/webman/3rdparty/xxx/images/icon_256.png</pre> <p>The icon value should also be set as “images/icon_{0}.png”</p>
title	O	“title” represents the application name that will be displayed in the main menu.
desc	X	“desc” displays more details about this application upon mouse-over.
url	O	<p>The following is an example of value setting for your URL of the application:</p> <p>“url”: <code>http://www.synology.com/</code> “url”: <code>“3rdparty/xxx/index.html”</code></p>

allUsers	X	<p>This key determines whether or not the menu items can be seen by users when they log in with an admin account. If you would like to have all users see the menu items, please set the key value as below:</p> <pre>"allUsers": true</pre> <p>The default setting is that only the admin can find the application.</p>
preloadTexts	X	<p>The specified i18n section:key strings will be loaded even when application ui is not opened. This is necessary when corresponding strings are used to send desktop notifications.</p>

Text fields support [i18n](#) value.

Application Help

To integrate help documents into DSM Help, please follow these steps:

1> Provide a `helptoc.conf` describing your help document structure and put it under the directory specified by `dsmuidir` in `INFO`.

```
{
  "app": "SYNO.App.TestAppInstance",
  "title": "app_tree:index_title",
  "content": "testapp_index.html",
  "toc": [
    {
      "title": "app_tree:node_1",
      "content": "testapp_node1.html",
      "nodes": [
        {
          "title": "app_tree:node_1_child",
          "content": "testapp_node1_child.html"
        }
      ]
    }, {
      "title": "app_tree:node_2",
      "content": "testapp_node2.html"
    }
  ]
}
```

Details of `helptoc.conf` are stated below:

Property	Description
app	the application instance.
title	the text being displayed.
content	the path to your help document.
toc	the child nodes of root. (use empty array if your application doesn't have one)
nodes	the child nodes of toc node.

Text fields support `i18n` value.

2> Create directories and files according to your `helptoc.conf`.

```
ui (specified by dsmuidir in INFO)
├─ helptoc.conf
├─ help
│   └─ enu
│       └─ testapp_index.html
│   └─ cht
│       └─ testapp_index.html
└─ texts
    └─ enu
        └─ strings
    └─ cht
        └─ strings
```

3> Write each help document in the following `HTML` format so that the UI style can be consistent with others.

```
<!DOCTYPE html>
<html class="img-no-display">
  <head>
```

```
<meta charset="UTF-8" />
<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">
<link href="../../help/help.css" rel="stylesheet" type="text/css">
<link href="../../help/scrollbar/flexcroll.css" rel="stylesheet" type="text/css">
<script type="text/javascript" src="../../help/scrollbar/flexcroll.js"></script>
<script type="text/javascript" src="../../help/scrollbar/initFlexcroll.js"></script>
</head>
<body>
  This is my help document content
</body>
</html>
```

Application Internationalization

The desktop application can have i18n text referenced by config, help, etc.

```
ui (specified by dsmuidir in INFO)
└─ texts
  └─ enu
    └─ strings
  └─ cht
    └─ strings
```

You have to create directories according to supported languages then create a file named `strings` inside each language directory.

- [dsmuidir]/texts/enu/strings

```
[app_tree]
index_title="This is a title"
node_1="This is node1"
[app_tab]
tab1="This is tab1"
tab2="This is tab2"
```

- [dsmuidir]/texts/cht/strings

```
[app_tree]
index_title=""
node_1="1"
[app_tab]
tab1="1"
tab2="2"
```

When you want to use these texts, just reference them in `section:key` format (one value can only be one i18n string)

```
"title": "app_tree:node_1"
```

I18N strings are loaded only when application opened on desktop after DSM 7.0. If the strings are used as desktop notifications, those strings should be specified in `preloadTexts` of [application config](#).

Application Authentication

After integrating your application into Synology DSM, you may want to perform an authentication check to ensure only logged-in users can access the page.

You can run `/usr/syno/synoman/webman/modules/authenticate.cgi` to check the user login status. However the `authenticate.cgi` must be run with some environment variables (`HTTP_COOKIE`, `REMOTE_ADDR`, `SERVER_ADDR`, etc.). So execute the `authenticate.cgi` directly from the package custom CGI is recommended since the environment variables needed are set automatically.

Sample Code *test.cgi*

The *authenticate.cgi* will output the user name if the user has logged in. There will be no output if the user has not been authenticated.

Here is the sample code for 3rd party CGI (Note. compile this with `-std=c99`)

```
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h>

/**
 * Check whether user is logged in.
 *
 * If user has logged in, put the username into "user".
 *
 * @param user    The buffer for get username
 * @param bufsize The buffer size of user
 *
 * @return 0: User not logged in or error
 *         1: User logged in. The user name is written to given "user"
 */

int IsUserLogin(char *user, int bufsize)
{
    FILE *fp = NULL;
    char buf[1024];
    int login = 0;

    bzero(user, bufsize);

    fp = popen("/usr/syno/synoman/webman/modules/authenticate.cgi", "r");
    if (!fp) {
        return 0;
    }
    bzero(buf, sizeof(buf));
    fread(buf, 1024, 1, fp);

    if (strlen(buf) > 0) {
        snprintf(user, bufsize, "%s", buf);
        login = 1;
    }
    pclose(fp);

    return login;
}

int main(int argc, char **argv)
{
    char user[256];

    printf("Content-Type: text/html\r\n\r\n");
    if (IsUserLogin(user, sizeof(user)) == 1) {
        printf("User is authenticated. Name: %s\n", user);
    }
}
```

```
    } else {  
        printf("User is not authenticated.\n");  
    }  
    return 0;  
}
```

How to run the *test.cgi*

DSM requires cookie to validate the DSM login session.

Login

Access the following cgi with your credential information, you will receive the session information in your cookie.

```
https://your-ip:5001/webapi/auth.cgi?api=SYNO.API.Auth&version=3&method=login&account=admin&passwd=your_admin_password&format=cookie
```

Note. If you're using the insecure http protocol, please alter the protocol and change the port number to 5000.

Access *test.cgi* with cookie

Access *test.cgi* with cookie information.

```
https://your-ip:5001/path/to/test.cgi
```

If you are having trouble accessing your *test.cgi*, please try to access any other webapi with your cookie. This would help you to clarify if your cookie information is valid or not.

```
https://your-ip:5001/webapi/entry.cgi?api=SYNO.Core.System&version=3&method=info
```

Logout

By accessing the following webapi, you will be logged out.

```
https://your-ip:5001/webapi/auth.cgi?api=SYNO.API.Auth&version=1&method=logout
```

Privilege Config

To make your package work, there must exist `conf/privilege` inside your package. It controls [security related behaviours](#) in entire package lifecycle.

```
{
  "defaults":{
    "run-as": "package"
  },
  "username": "myusername",
  "groupname": "mygroupname",
  "tool": [{
    "relpath": "bin/mytool",
    "user": "package",
    "group": "package",
    "permission": "0700"
  }]
}
```

defaults (required)

Controls default settings for entire privilege file. It can only be set as value below.

run-as	behaviour on file	behaviour on script
package	<code>chown -hR "\${package}:\${package}"</code>	set resid as [username]

run-as	behaviour on file	behaviour on script
root	<code>chown -hR "root:root"</code>	set resid as root

username / groupname (optional) (since 6.0-5940)

Specify which name will be the user name and group name. If not specified, the package name will be the default value.

ctrl-script (optional)

Control the identity to run scripts.

```
"ctrl-script": [{
  "action": "start",
  "run-as": "package"
}]
```

Member	Since	Description
action	6.0-5891	one of <code>preinst</code> , <code>postinst</code> , <code>preuninst</code> , <code>postuninst</code> , <code>preupgrade</code> , <code>postupgrade</code> , <code>start</code> , <code>stop</code> , <code>status</code> , <code>prestart</code> , <code>prestop</code>
run-as	6.0-5891	see the description above

executable (optional)

Specify the identity to **chown** on installed for specific file.

```
"executable": [{
  "relpath": "bin/mybin",
  "run-as": "package"
}]
```

```
}}
```

Member	Since	Description
relpath	6.0-5891	relative path under <code>/var/packages/[package_name]/target</code>
run-as	6.0-5891	see the description above

tool (optional)

Specify the identity to **chown** and **chmod** on installed for specific file.

If you want, you can even set file capabilities.

```
"tool": [{
  "relpath": "bin/mytool",
  "user": "package",
  "group": "package",
  "permission": "0700"
}]
```

Member	Since	Description
relpath	6.0-5891	String, the file's relative path under <code>/var/packages/\${package}/target/</code> .
user	6.0-5891	String, file's owner user, must be "package".
group	6.0-5891	String, file's owner group, must be "package"
permission	6.0-5891	4 digit number to set file permission, for example: 4750

```
"tool": [{
  "relpath": "bin/mytool",
  "user": "package",
  "group": "package",
  "capabilities": "cap_chown,cap_net_raw",
  "permission": "0700"
}]
```

Member	Since	Description
capabilities	7.0-40656	capabilities string without any <code>+-=eip</code> symbol. the value can be viewed HERE

Package User / Group Visibility On UI

Package users and groups will not appear on most UI settings, but there are some exceptions:

- [x] Application privilege permission viewer
- [x] FTP chroot user selector
- [x] File Station
 - [x] Change owner
 - [x] Shared Links Manager -> Enable secure sharing
- [o] Control Panel > Shared Folder > Edit > Permission > System internal user
- [o] ACL editor

Resource Config

It defines the system resource that is necessary for this package to work.

```
{
  "<resource-id>": {
    <specification>
  }
}
```

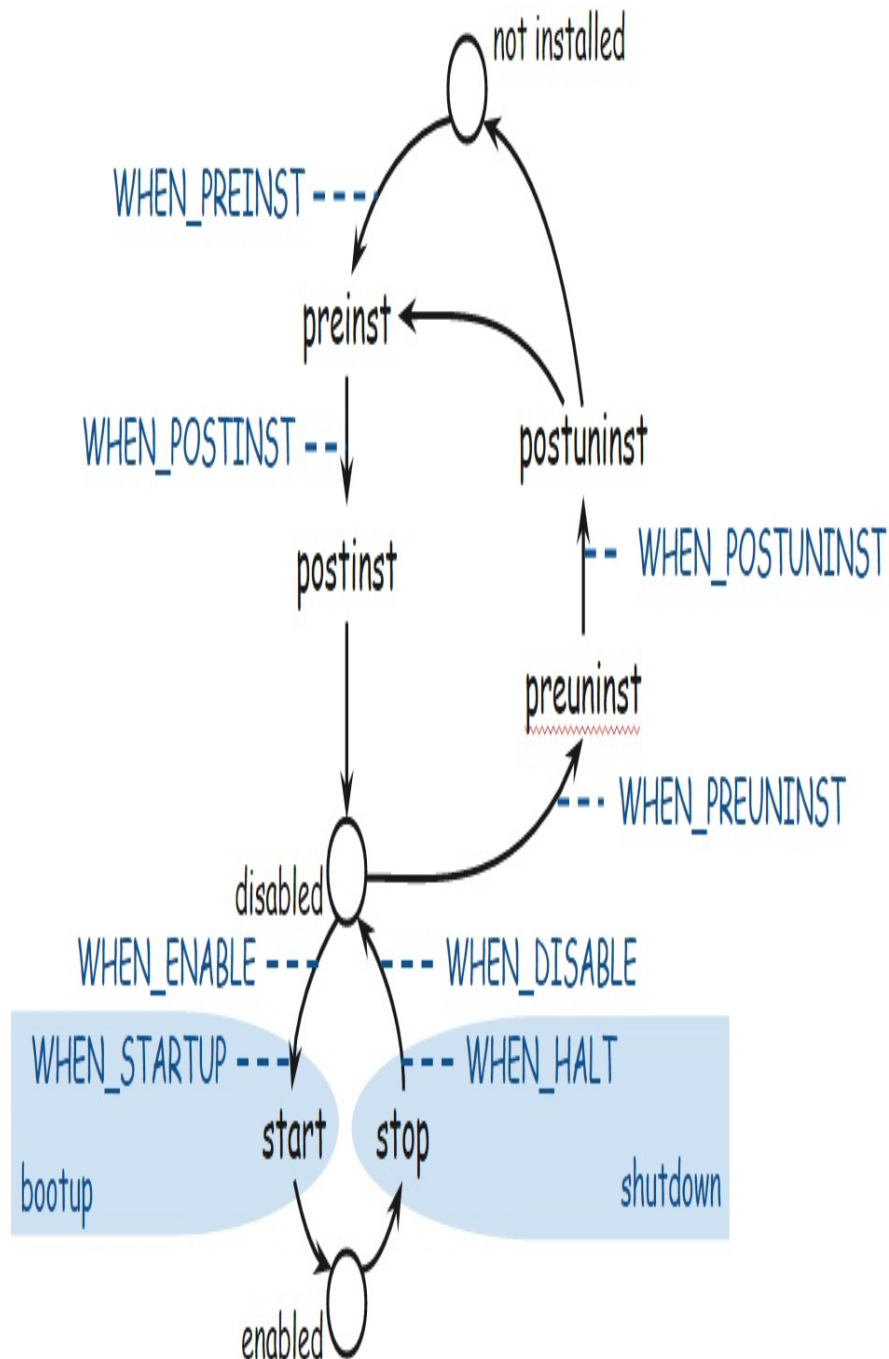
For example, you can apply */usr/local linker*:

```
{
  "usr-local-linker": {
    "lib": ["lib/foo"],
    "bin": ["bin/foo"],
    "etc": ["etc/foo"],
  }
}
```

From this example, the `usr-local-linker` represents the resource id and its value represents the file to be linked.

Resource Timing

Every worker acquires resources at certain timings and holds it during an interval. For example, `/usr/local linker` holds the resource during the interval `FROM_ENABLE_TO_DISABLE`, which means it acquires resource at `WHEN_ENABLE` and releases it at `WHEN_DISABLE`. The timings are listed and explained below:



timing	description	when Failure
WHEN_PREINST	before <i>preinst</i>	abort installation, rollback, show alert message on UI
WHEN_POSTINST	before <i>postinst</i>	finish installation, show alert message on UI
WHEN_ENABLE	before WHEN_STARTUP , won't process during bootup	abort startup, rollback, show alert message on UI
WHEN_STARTUP	before <i>start</i>	abort startup, rollback, show alert message on UI

WHEN_PREUNINST	after <i>preuninst</i>	finish uninstallation, show alert message on UI
WHEN_POSTUNINST	before <i>postuninst</i>	finish uninstallation, show alert message on UI
WHEN_DISABLE	after WHEN_HALT , won't process during shutdown	ignore
WHEN_HALT	after <i>stop</i>	ignore

NOTE To let the package itself decide whether uninstallation should continue or not, WHEN_PREUNINST is processed after the preuninst script.

Resource Update

Some workers support update operation outside of worker timings. `/usr/syno/sbin/synopkg-helper` should be used to accomplish this job. Below are the steps to update the resource:

1. Update the file at `/var/packages/[package_name]/conf/resource`
2. Execute the command `/usr/syno/sbin/synopkg-helper update [package_name] [resource_id]` to trigger updating procedure.

For example, suppose a package allows the user to edit its listening port and needs to update corresponding network settings:

1. User submits new port to the application
2. The application updates the file at `/var/packages/[package_name]/conf/resource`
3. The application executes the command `/usr/syno/sbin/synopkg-helper update ${package} port-config`, then the `port-config` worker will read the config and reload network settings.

NOTE Not all resource support update operation, please refer to the *Updatable* section of each resource.

Available Workers

As mentioned in the section [Resource](#), a worker is needed for resource management.

Given a [Resource Config](#) file, the resource worker will acquire / release the resource at certain time. This section describes the available resource workers on the DSM.

/usr/local linker

Description

Package's executables and library files should be installed to */usr/local*. This worker link / unlink files to */usr/local/{bin,lib,etc}* during package start / stop.

- Acquire() : Create symbolic links under */usr/local/{bin,lib,etc}/* that points to files in */var/packages/\${package}/target/*.
 - Files not found under */var/packages/\${package}/target/* will be ignored.
 - If the target file already exists in */usr/local/{bin,lib,etc}*, it will be `unlink()` first.
 - Failure on any file link results in this worker to abort and triggers rollback.
- Release() : Delete the links under */usr/local/{bin,lib,etc}/*.
 - Ignore files that are not found.
 - Ignore `unlink()` failure.

Provider

DSM

Timing

FROM_ENABLE_TO_DISABLE

Environment Variables

None

Updatable

No

Syntax

```
"usr-local-linker": {
  "bin" ["<relpath>", ...],
  "lib" ["<relpath>", ...],
  "etc" ["<relpath>", ...]
}
```

Member	Since	Description
bin	6.0-5941	String array, list of files to be linked under <i>/usr/local/bin/</i> .
lib	6.0-5941	String array, list of files to be linked under <i>/usr/local/lib/</i> .
etc	6.0-5941	String array, list of files to be linked under <i>/usr/local/etc/</i> .
relpath	6.0-5941	String, target file's relative path under <i>/var/packages/\${package}/target/</i> .

Example

```
"usr-local-linker": {
  "bin": ["usr/bin/a2p", "usr/bin/perl"],
  "lib": ["lib/perl5"]
}
```

The above specifications generates the following symbolic links for the Perl package:

```
root@DS $ ls -l /usr/local/{bin,lib,etc}
/usr/local/bin/:
total 0
lrwxrwxrwx 1 root root 30 Aug 13 06:32 a2p -> /var/packages/Perl/target/usr/bin/a2p
lrwxrwxrwx 1 root root 31 Aug 13 06:32 perl -> /var/packages/Perl/target/usr/bin/perl

/usr/local/lib/:
total 0
lrwxrwxrwx 1 root root 28 Aug 13 06:32 perl5 -> /var/packages/Perl/target/lib/perl5

/usr/local/etc/:
total 0
```

Apache 2.2 Config

Description

Packages can carry sites-enabled/*.conf files for Apache HTTP Server 2.2. This worker installs / uninstalls these config files during package start / stop.

- `Acquire()` : Copy the conf files to `/usr/local/etc/httpd/sites-enabled/`. Then reload Apache 2.2.
 - The files should have .conf extension, otherwise it will be ignored
 - Files will be prefixed by `${package}`.
 - Existing files will be `unlink()` first.
 - Failure on any file copy results in this worker to abort and triggers rollback.
- `Release()` : Delete previously created links
 - Ignore files that are not found.
 - Ignore `unlink()` failure.

Provider

WebStation

Timing

FROM_ENABLE_TO_DISABLE

Environment Variables

None

Updatable

No

Syntax

```
"apache22": {
  "sites-enabled": [{
    "relpath": "<conf-relpath>",
    }, ...]
}
```

Member	Since	Description
sites-enabled	WebStation-1.0-0049	Object array, list of conf files to install.
relpath	WebStation-1.0-0049	Target file's relative path under <code>/var/packages/\${package}/target/</code> .

Example

```
{
  "apache22": {
    "sites-enabled": [{
      "relpath": "synology_added/test_1.conf"
    }, {
      "relpath": "synology_added/test_2.conf"
    }, {
      "relpath": "synology_added/test_3.conf"
    }
  ]
}
```


Data Share

Description

This worker creates shared folder and set its permission during package startup. The share name can be hard-coded in the specification. The shared folder will not be removed after package uninstallation, since it might delete the user's personal data as well.

- `Acquire()` : Create shared folder and set its permission.
 - If the shared folder already exists, skip share creation and set the permission.
- `Release()` : Does nothing

Provider

DSM

Timing

`FROM_ENABLE_TO_POSTUNINST`

Environment Variables

None

Updatable

No

Syntax

```
"data-share": {
  "shares": [{
    "name": "<share-name>",
    "permission": {
      "ro": ["<user-name>", ...],
      "rw": ["<user-name>", ...]
    },
    "once": "<once>"
  }, ...]
}
```

Member	Since	Description
<code>shares</code>	6.0-5914	Object array, array of shares to create
<code>name</code>	6.0-5914	String, name of the share
<code>permission</code>	6.0-5914	Json object, permission of the share. (optional)
<code>ro</code>	6.0-5914	String array users to be assigned with read-only permission.
<code>rw</code>	6.0-5914	String array users to be assigned with read / write permission.
<code>once</code>	6.0-5914	Boolean, only try to create share on package's first start. (optional, default = <code>false</code>)

Example

The following specification creates a share *music*, and gives the user *AudioStation* read-only permission. Since `once` defaults to `false`, the above procedure is ran every time the package starts.

```
"data-share": {  
  "shares": [{  
    "name": "music",  
    "permission": {  
      "ro": ["AudioStation"]  
    }  
  }  
}]  
}
```

since 7.0-41201, package center will create a symlink under `/var/packages/[package_id]/shares/` named by share folder pointing to share folder path.

Docker (since DSM7.0)

Description

Docker worker is made for docker package to help them easily deploy their containers without calling docker command by themselves. Docker worker use docker-compose framework, it will generate docker-compose.yaml according to user's docker worker configuration and create containers during installation.

When in install/remove package stage, worker will create/remove docker-compose.yaml, volume on host directory, images and containers.

When in start/stop package stage, worker will start/stop containers by calling docker-compose start/stop.

FROM_POSTINST_TO_PREUNINST

- `Acquire()` : Create docker-compose.yaml and prepare host volume for container to mount. Worker will also create containers in this stage.
- `Release()` : Remove docker-compose.yaml, host volume, containers and images. Note that worker will not remove host volume during upgrade docker package.

FROM_STARTUP_TO_HALT

- `Acquire()` : Start containers.
- `Release()` : Stop containers.

Provider

Docker

Timing

`FROM_POSTINST_TO_PREUNINST` `FROM_STARTUP_TO_HALT`

Environment Variables

None

Updatable

No

Syntax

```
"docker":{
  "services": [{
    service setting 1
  },{
    service setting 2
  }...],
}
```

Key	Since	Type	Required	Nullable	Default Value	Description
services	18.09.0-1018	Array	true	false	N/A	List of docker services information to create docker-compose.yaml

services

`services` specify service configurations such as service name, image name and tag, container name, volume ... etc. Docker worker will create docker-compose.yml according to given service configurations.

Key	Since	type	Required	Nullable	Default Value	Description
service	18.09.0-1018	string	true	false	N/A	Service name.
image	18.09.0-1018	string	true	false	N/A	Image name.
tag	18.09.0-1018	string	true	false	N/A	Image tag.
build	18.09.0-1018	string	true	false	N/A	Relative path to Dockerfile directory that package carries. More detail will be elaborated in <code>build</code> section.
container_name	18.09.0-1018	string	false	true	N/A	Container name.
shares	18.09.0-1018	array of objects	false	true	N/A	Container mount volume specifications especially for persistant data perpose. More detail will be elaborated in <code>volumes</code> section.
volumes	18.09.0-1018	array of objects	false	true	N/A	Container mount volume specifications especially for mounting config file perpose. More detail will be elaborated in <code>volumes</code> section.
ports	18.09.0-1018	array of objects	false	true	N/A	Container ports specification.
environment	18.09.0-1018	array of objects	false	false	N/A	Container environment variables specification.
depends	18.09.0-1018	array of objects	false	false	N/A	Specify dependent service.

build

`build` attribute is for building image with given Dockerfile path. The path will be relative path based on package target path (`/var/packages/PKG_NAME/target/`).

- Syntax:

```
{
  "build": "[Dockerfile directory]"
}
```

Transform to docker-compose.yml:

```
build: /var/packages/PKG_NAME/target/[Dockerfile directory]
```

- Example: Let odoo_docker directory contains Dockerfile and is under `/var/packages/Odoo/target/`

```
{
  "build": "odoo_docker"
}
```

Transform to docker-compose.yaml:

```
build: /var/packages/0doo/target/odoo_docker
```

volumes

Volumes contains two categories - `shares` and `volumes` which are for different purposes. Although those two categories will all be transform into docker-compose's "volumes" section, we separate them for different usage. That is, `shares` attribute is for persistent data volumes while `volumes` is for configuration files or any other configurations relative files.

shares: The `shares` attribute is for containers to persistent data. User only need to fill in the a directory name in `shares` and the worker will first create directory under docker share directory for user and, then, generate `SOURCE:TARGET` pair under `volumes` section in docker-compose.yaml.

- Syntax:

```
{
  "shares": [{
    "host_dir": "[host directory]",
    "mount_point": "[mount point]"
  }, ... {
    ...
  }]
}
```

Transform to docker-compose.yaml:

```
volumes:
- /volumeX/docker/PKG_NAME/[host directory]:[mount point]
```

- Example:

```
{
  "shares": [{
    "host_dir": "odoo_data",
    "mount_point": "/var/lib/odoo"
  }]
}
```

Transform to docker-compose.yaml:

```
volumes:
- /volume1/docker/0doo/odoo_data:/var/lib/odoo
```

volumes: The `volumes` attribute is similar to `shares` attribute but is design for configuration files or directory that user would like to mount into container. User can specify relative path of host configuration file or directory based on package target path (/var/packages/PKG_NAME/target/) and the worker will generate `SOURCE:TARGET` pair under `volumes` section in docker-compose.yaml.

- Syntax:

```
{
  "volumes": [{
    "host_dir": "[host config or directory]",
    "mount_point": "[mount point]"
  }, ... {
    ...
  }]
}
```

Transform to docker-compose.yml:

```
volumes:
  - /var/packages/PKG_NAME/target/[host config or directory]:[mount point]
```

- Example:

```
{
  "volumes": [{
    "host_dir": "odoo_docker/config",
    "mount_point": "/etc/odoo"
  }]
}
```

Transform to docker-compose.yml:

```
volumes:
  - /var/packages/Odoo/target/odoo_docker:/etc/odoo
```

ports

`ports` attribute is for creating ports binding for container.

- Restriction: Host port needs to be at between 1025 to 65535.
- Syntax:

```
{
  "ports": [{
    "host_port": "[port on host]",
    "container_port": "[port in container]",
    "protocol": "[tcp or udp]"
  }, ... {
    ...
  }]
}
```

Transform to docker-compose.yml:

```
ports:
  - "[port on host]:[port in container]/[tcp or udp]"
```

- Example:

```
{
  "ports": [{
    "host_port": "30076",
    "container_port": "80",
    "protocol": "tcp"
  }, {
    "host_port": "30078",
    "container_port": "443",
    "protocol": "tcp"
  }]
}
```

Transform to docker-compose.yml:

```
ports:
  - "30076:80/tcp"
  - "30078:443/tcp"
```

environment

`environment` attribute is for creating environment variables and values for containers.

- Syntax:

```
{
  "environment": [{
    "env_var": "[variable name]",
    "env_value": "[value]"
  }, ... {
    ...
  }]
}
```

Transform to docker-compose.yml:

```
environment:
- "[variable name]:[value]"
```

- Example:

```
{
  "environment": [{
    "env_var": "HOST",
    "env_value": "odoo_db"
  }, {
    "env_var": "USER",
    "env_value": "odoo"
  }, {
    "env_var": "PASSWORD",
    "env_value": "odoo"
  }]
}
```

Transform to docker-compose.yml:

```
environment:
- HOST=odoo_db
- USER=odoo
- PASSWORD=odoo
```

depends

`depends` attribute is for specifying dependent services, in the same way as docker-compose.

- Syntax:

```
{
  "depends": [{
    "dep_service": "[service name]"
  }, ... {
    ...
  }]
}
```

Transform to docker-compose.yml:

```
depends_on:
- [service name]
```

- Example:

```
{
```

```

    "depends": [{
      "dep_service": "odoo_db"
    }]
  }
}

```

Transform to docker-compose.yml:

```

depends_on:
  - odoo_db

```

docker-compose generation example

conf/resource:

```

{
  "docker": {
    "services": [{
      "service": "odoo",
      "build": "odoo_docker",
      "image": "odoo",
      "container_name": "Odoo",
      "tag": "12.0",
      "environment": [{
        "env_var": "HOST",
        "env_value": "odoo_db"
      }], {
        "env_var": "USER",
        "env_value": "odoo"
      }, {
        "env_var": "PASSWORD",
        "env_value": "odoo"
      }],
      "shares": [{
        "host_dir": "odoo_data",
        "mount_point": "/var/lib/odoo"
      }],
      "ports": [{
        "host_port": "{{wizard_http_port}}",
        "container_port": "8069",
        "protocol": "tcp"
      }],
      "depends": [{
        "dep_service": "odoo_db"
      }]
    }, {
      "service": "odoo_db",
      "image": "postgres",
      "tag": "10",
      "container_name": "Odoo_db",
      "shares": [{
        "host_dir": "db",
        "mount_point": "/var/lib/postgresql/data/pgdata"
      }],
      "environment": [{
        "env_var": "POSTGRES_DB",
        "env_value": "postgres"
      }], {
        "env_var": "POSTGRES_PASSWORD",
        "env_value": "odoo"
      }, {
        "env_var": "POSTGRES_USER",
        "env_value": "odoo"
      }, {
        "env_var": "PGDATA",
        "env_value": "/var/lib/postgresql/data/pgdata"
      }],
    }]
  }
}

```



```
}
```

Transform to docker-compose.yaml:

```
version: '3'
services:
  odoo:
    build: /var/packages/Docker_Odoo_SynoCommunity/target/odoo_docker
    image: odoo:12.0
    container_name: Odoo
    environment:
      - HOST=odoo_db
      - USER=odoo
      - PASSWORD=odoo
    volumes:
      - /volume1/docker/Docker_Odoo_SynoCommunity//odoo_data:/var/lib/odoo
    ports:
      - "30076:8069/tcp"
    depends_on:
      - odoo_db
    networks:
      - Docker_Odoo_SynoCommunity
  odoo_db:
    image: postgres:10
    container_name: Odoo_db
    environment:
      - POSTGRES_DB=postgres
      - POSTGRES_PASSWORD=odoo
      - POSTGRES_USER=odoo
      - PGDATA=/var/lib/postgresql/data/pgdata
    volumes:
      - /volume1/docker/Docker_Odoo_SynoCommunity//db:/var/lib/postgresql/data/pgdata
    networks:
      - Docker_Odoo_SynoCommunity
networks:
  Docker_Odoo_SynoCommunity:
    driver: bridge
```

Index DB

Description

Index / unindex package help and app index during package start / stop.

For detailed description on package app index and help index, please refer to [Integrate Help Document into DSM Help](#).

- `Acquire()` : Index package help and app content.
- `Release()` : Un-index package help and app content.

Provider

DSM

Timing

`FROM_ENABLE_TO_DISABLE`

Environment Variables

None

Updatable

No

Syntax

```
"indexdb": {
  "app-index" : {
    "conf-relpath": "<conf relpath>",
    "db-relpath": "<app db relpath>"
  },
  "help-index": {
    "conf-relpath": "<conf relpath>",
    "db-relpath": "<help db relpath>"
  }
}
```

Member	Since	Description
<code>app-index</code>	6.0-5924	Object, app index info.
<code>help-index</code>	6.0-5924	Object, help index info.
<code>conf-relpath</code>	6.0-5924	String, config file's relative path under <code>/var/packages/\${package}/target/</code> .
<code>db-relpath</code>	6.0-5924	String, db folder's relative path under <code>/var/packages/\${package}/target/</code> .

Example

```
"indexdb": {
  "app-index" : {
    "conf-relpath": "app/index.conf",
    "db-relpath": "indexdb/appindexdb"
  },
  "help-index": {
```

```
    "conf-reldpath": "app/helpdoc.conf",  
    "db-reldpath": "indexdb/helpindexdb"  
  }  
}
```

Maria DB 10

Description

This worker register on the following timings:

- `PREINST/PREUNINST` It checks the resource specification from user / wizard to avoid failure on another timing.
- `POSTINST/POSTUNINST` It performs several stages when package intents to do so:
 - `POSTINST` (install & upgrade)
 - **migrate-db**: migrate db from mariadb5 to mariadb10, usually used on upgrade
 - **create-db**: create database
 - **grant-user**: create user
 - **drop-db-inst**: drop old database, usually used on db migration
 - `POSTUNINST` (uninstall) (do not run during update)
 - **drop-db-uninst**: drop database
 - **drop-user-uninst**: delete user, if multiple packages share same user, this option should not be applied

If worker fails on any stage, worker framework would rollback the performed operations.

Provider

MariaDB10 package

Timing

`FROM_PREINST_TO_PREUNINST`

`FROM_POSTINST_TO_POSTUNINST`

Environment Variables

None

Updatable

No

Syntax

```
"mariadb10-db": {
  "admin-account-m10": "<db account>",
  "admin-pw-m10": "<db password>",
  "admin-account-m5": "<m5 db account>",
  "admin-pw-m5": "<m5 db password>",
  "migrate-db": {
    "flag": true | false,
    "m5-db-name": "<db name>",
    "m10-db-name": "<db name>",
    "db-collision": "replace" | "error"
  },
  "create-db": {
    "flag": true | false,
    "db-name": "<db name>",
    "db-collision": "replace" | "skip" | "error"
  },
  "grant-user": {
    "flag": true | false,
```

```

    "db-name": "<db name>",
    "user-name": "<db username>",
    "host": "<db host>",
    "user-pw": "<db password>"
  },
  "drop-db-inst": {
    "flag": true | false,
    "ver": "m5" | "m10",
    "db-name": "<db name>"
  },
  "drop-db-uninst": true | false,
  "drop-user-uninst": true | false
}

```

All fields are not necessary, but if you enable some stages then you have to fillup some fields. (e.g., enable `create-db` stage, then you have to provide `admin-account-m10` and `admin-pw-m10`)

Member(L1)	Member(L2)	Since	Description
<code>admin-account-m10</code>	-	10.0.30-0005	MariaDB10 account id which has full access permission(root)
<code>admin-pw-m10</code>	-	10.0.30-0005	MariaDB10 account password which has full access permission(root)
<code>admin-account-m5</code>	-	10.0.30-0005	MariaDB account id which has full access permission(root)
<code>admin-pw-m5</code>	-	10.0.30-0005	MariaDB account password which has full access permission(root)
<code>migrate-db</code>	<code>flag</code>	10.0.30-0005	whether to run the stage
	<code>m5-db-name</code>	10.0.30-0005	migration source db name of MariaDB
	<code>m10-db-name</code>	10.0.30-0005	migration destination db name of MariaDB10
	<code>db-collision</code>	10.0.30-0005	DB Collision Strategy on import do not provide skip
<code>create-db</code>	<code>flag</code>	10.0.30-0005	whether to run the stage
	<code>db-name</code>	10.0.30-0005	db name to create on MariaDB10
	<code>db-collision</code>	10.0.30-0005	DB Collision Strategy on create db
<code>grant-user</code>	<code>flag</code>	10.0.30-0005	whether to run the stage
	<code>db-name</code>	10.0.30-0005	target db name to grant for user
	<code>user-name</code>	10.0.30-0005	create and grant user name
	<code>host</code>	10.0.30-0005	user host (default = localhost)
	<code>user-pw</code>	10.0.30-0005	user password
<code>drop-db-inst</code>	<code>flag</code>	10.0.30-0005	whether to run the stage
	<code>ver</code>	10.0.30-0005	target mariadb version (m5 / m10)

	db-name	10.0.30-0005	db name to drop
drop-db-uninst	-	10.0.30-0005	whether to run the stage
drop-user-uninst	-	10.0.30-0005	whether to run the stage

DB Collision Strategy : when there are same db names as provided from `migrate-db` & `create-db` , it can solve the conflict by one of the following strategies:

1. **replace**
drop exist db and replace it using new db
2. **error**
do not do anything and just report error, which might cause installation failure
3. **skip**
do not do anything and continue to execute as usual

Example

```
"mariadb10-db": {
  "admin-account-m10": "root",
  "admin-pw-m10": "password!@#123432",
  "admin-account-m5": "",
  "admin-pw-m5": "",
  "migrate-db": {
    "flag": false,
    "m5-db-name": "",
    "m10-db-name": "",
    "db-collision": ""
  },
  "create-db": {
    "flag": true,
    "db-name": "myservice",
    "db-collision": "error"
  },
  "grant-user": {
    "flag": true,
    "db-name": "myservice",
    "user-name": "myservice_dbuser",
    "host": "localhost",
    "user-pw": "password!@#123432asd123123"
  },
  "drop-db-inst": {
    "flag": false,
    "ver": "",
    "db-name": ""
  },
  "drop-db-uninst": true,
  "drop-user-uninst": false
}
```

PHP INI

Description

Packages can carry custom php.ini and fpm.conf files. This worker installs / uninstalls these config files during package start / stop.

- `Acquire()` : Copy the php.ini and fpm.conf files to `/usr/local/etc/php56/conf.d/` and `/usr/local/etc/php56/fpm.d/`. Then reload php56-fpm.
 - `php.ini` / `fpm.conf` files should have `.ini` / `.conf` extension, otherwise it will be ignored
 - Files will be prefixed by `${package}`.
 - Existing files will be `unlink()` first.
 - Failure on any file copy results in this worker to abort and triggers rollback.
- `Release()` : Delete previously created links
 - Ignore files that are not found.
 - Ignore `unlink()` failure.

Provider

PHP5.6

Timing

FROM_ENABLE_TO_DISABLE

Environment Variables

None

Updatable

No

Syntax

```
"php": {
  "php-ini": [{
    "relpath": "<ini-relpath>",
  }, ...],
  "fpm-conf": [{
    "relpath": "<conf-relpath>",
  }, ...]
}
```

Member	Since	Description
php-ini	PHP5.6-5.6.17-0020	Object array, list of php.ini files to install.
fpm-conf	PHP5.6-5.6.17-0020	Object array, list of fpm.conf files to install.
relpath	PHP5.6-5.6.17-0020	Target file's relative path under <code>/var/packages/\${package}/target/</code> .

Example

```
{
  "php": {
    "php-ini": [{
```

```
        "relpath": "synology_added/etc/php/conf.d/test_1.ini"
    }, {
        "relpath": "synology_added/etc/php/conf.d/test_2.ini"
    }, {
        "relpath": "synology_added/etc/php/conf.d/test_3.ini"
    }],
    "fpm-conf": [{
        "relpath": "synology_added/etc/php/fpm.d/test_1.conf"
    }, {
        "relpath": "synology_added/etc/php/fpm.d/test_2.conf"
    }, {
        "relpath": "synology_added/etc/php/fpm.d/test_3.conf"
    }]
}
}
```


Port Config

Description

Install / uninstall service port config file during package install / uninstall.

For detailed description on what is and how to write a port config file, please refer to [Install Package Related Ports Information into DSM](#).

- `Acquire()` : copy the .sc file to `/usr/local/etc/service.d/`
 - If the destination file exists, skip file copy.
- `Release()` : remove the .sc file and reload the firewall and portforward.
- `Update()` : update the .sc file and reload firewall and portforward.

Timing

FROM_POSTINST_TO_POSTUNINST

Environment Variables

None

Updatable

Yes, please refer to [Config Update](#) on how to trigger update.

Syntax

```
"port-config": {  
  "protocol-file": <protocol_file>  
}
```

Member	Since	Description
protocol_file	6.0-5936	.sc file's relative path under <code>/var/package/{\$package}/target/</code>

Example

```
"port-config": {  
  "protocol-file": "port_conf/xxdns.sc"  
}
```

Systemd User Unit

Description

The package framework would copy files at `conf/systemd/pkguser-[customname]` to `home/.config/systemd/user/` on acquired and remove them on released.

note that user unit cannot be related with normal systemd unit. If you need your package to be related with system service, please refer to [start_dep_services](#)

The package should use `synosystemctl start` and `synosystemctl stop` to control user units inside scripts.

Extra

If you want to have [systemd unit](#) inside the system, you may just put your units at `conf/systemd/pkg-[customname]` **without** the need to use this `systemd-user-unit` worker.

The package framework would copy systemd units to `/usr/local/lib/systemd/system` on acquired and remove them on released.

Provider

DSM

Since

7.0-40761

Timing

`FROM_POSTINST_TO_POSTUNINST`

Syntax

```
"systemd-user-unit": {}
```

Syslog Config

Description

Install / uninstall the syslog-ng and logrotate config file during package start / stop.

Please refer to [syslog-ng](#) on how to write the syslog-ng's config file.

- `Acquire()` : Copy `patterndb / logrotate` to `/usr/local/etc/syslog-ng/patterndb.d/ /usr/local/etc/logrotate.d/`. Then reload syslog-ng.
 - If file exists, `unlink()` it first.
 - Failure on any file copy results in this worker to abort and triggers rollback.
- `Release()` : Delete the config files and reload syslog-ng.
 - Ignore `unlink()` failure.

Provider

DSM

Timing

`FROM_ENABLE_TO_DISABLE`

Environment Variables

None

Updatable

No

Syntax

```
"syslog-config": {
  "patterndb-relpath": "<relpath>",
  "logrotate-relpath": "<relpath>"
}
```

Member	Since	Description
<code>patterndb-relpath</code>	6.0-7145	String, syslog-ng's config file's relative path under <code>/var/packages/\${package}/target/</code> , ignore this if the log is not generated by syslog-ng (optional)
<code>logrotate-relpath</code>	6.0-5911	String, logrotate's config file's relative path under <code>/var/packages/\${package}/target/</code> , ignore this if log is saved to database (optional)

Example

```
"syslog-config": {
  "patterndb-relpath": "etc/syslog-ng.conf",
  "logrotate-relpath": "etc/logrotate.conf"
}
```


Web Service (since DSM7.0)

Description

When in install/remove package stage, worker will update/remove service and default portal setting.

When in start/stop package stage, worker will start/stop service setting.

FROM_PREINST_TO_PREUNINST

- `Acquire()` : sync information in user specified `/var/package/${package}/target/*.json` into user setting, do **migrate** and setup **portal** and **service** which user specify in **resource** file
- `Release()` : remove user's setting

FROM_ENABLE_TO_DISABLE

- `Acquire()` : copy `*.json` and `.mustache` under `/var/packages/${package}/target/` into `/usr/syno/etc/www/app.d/` and enable **service** setting.
- `Release()` : remove files which copied into `/usr/syno/etc/www/app.d/` and disable **service** setting

Provider

WebStation

Timing

`FROM_PREINST_TO_PREUNINST` `FROM_ENABLE_TO_DISABLE`

Lower privilege

According to package center [privilege policy](#), web package will get a confined privilege during installation and run time. In order to setup environment for web package, webservice worker provide a mechanism called `pkg_dir_prepare` to assist web package creating website root directory and setting corresponding owner, group. The detail of `pkg_dir_prepare` will be elaborate in [pkg_dir_prepare](#) section.

Environment Variables

None

Updatable

No

Syntax

```
"webservice":{
  "services": [{
    service setting 1
  },{
    service setting 2
  }...],
  "portals": [{
    default portal setting 1
  },{
    default portal setting 2
  }],
  "migrate": {
    Migration data
  },
}
```

```

    "pkg_dir_prepare": [{
        package directory prepare settings
    }]
}

```

Key	Since	Type	Required	Nullable	Default Value	Description
services	3.0.0-0214	Array	true	false	N/A	List services which are wanted to be registered
portals	3.0.0-0214	Array	false	true	Empty array	List default portal for services (Unnecessary)
migrate	3.0.0-0214	Object	false	true	Empty object	Migrate information (Unnecessary)
pkg_dir_prepare	3.0.0-0256	Array	true	false	Empty array	Setting specification of website root under web_package

Framework will use default value when field is not **required** and doesn't exist or is null.

services

Web services which are going to register, allow multiple web services to register. For more detail please see [Web Service](#)

portals

Default portal which are going to register for access portal of services and will create UI Shortcut. Divided into `server portal` and `alias portal`.

Important: Default server portal is not allowed registered as name base portal, since you may not be able to lookup FQDN's correct IP from client side.

Example:

```

Alias Portal
{
    "service": "wordpress",
    "name": "wordpress",
    "app": "SYNO.SDS.WordPress",
    "type": "alias",
    "alias": "wordpress"
}
Server Portal
{
    "service": "wordpress",
    "name": "wordpress",
    "app": "SYNO.SDS.WordPress",
    "type": "server",
    "http_port": [9000],
    "https_port": [9001]
}

```

Key	Since	type	Required	Nullable	Default Value	Description
service	3.0.0-0214	string	true	false	N/A	portal service name that portal link, corresponding to <code>service</code> field in service that is about to register
name	3.0.0-0214	string	true	false	N/A	portal name
display_name	3.0.0-0302	string	false	true	same as <i>name</i>	the title of web UI portal shortcut

app	3.0.0-0214	string	false	true	empty string	pacakge's UI App name
type	3.0.0-0214	string	true	false	N/A	portal's type, could be alias or server
alias	3.0.0-0214	string	true (if type is alias)	false	N/A	alias name
http_port	3.0.0-0214	int array	false	false	empty array (if type is server)	Http port setting for server portal, only 1 port allowed. There should be at least http_port or https_port or both.
https_port	3.0.0-0214	int array	false	false	empty array (if type is server)	Https port setting for server portal, only 1 port allowed. There should be at least http_port or https_port or both.

migrate

Migrate assist package migration from older version (< DSM 7.0) to newer version. Supporting two kinds of migrate setting - `root` and `vhost` .

root

```
"root": [{
  "old": "wordpress",
  "new": "wordpress"
}]
```

Key	Since	type	Required	Nullable	Default Value	Description
root	3.0.0-0214	array	false	true	empty array	Migrate web package from web share folder to web_packages share folder.
old	3.0.0-0214	string	true	false	N/A	name of old package which in web share folder.
new	3.0.0-0214	string	true	false	N/A	name of new package which in web_packages share folder.

vhost

```
"vhost": [{
  "root": "wordpress",
  "service": "wordpress"
}]
```

Key	Since	type	Required	Nullable	Default Value	Description
vhost	3.0.0-0222	array	false	true	empty array	Migrate virtualhost, which pointing to old package, to service portal.
root	3.0.0-0222	string	true	false	N/A	name of old pacakge which in web share folder.
service	3.0.0-0222	string	true	false	N/A	new package's service name

pkg_dir_prepare

Webservice worker will set up website root directory under `web_packages` according to the information web package specified in worker config. The worker will remove the `target` directory under `web_package` between `preuninst` and `postuninst`. Make sure to **backup** your website root in `preuninst` script during upgrade.

`pkg_dir_prepare` example:

```
"pkg_dir_prepare": [{
  "source": "/var/package/WordPress/target/src",
  "target": "wordpress",
  "mode": "0755",
  "group": "http",
  "user": "WordPress"
}]
```

Key	Since	type	Required	Nullable	Default Value	Description
source	3.0.0-0256	string	false	true	N/A	Your web package source code directory. Mostly it will be under package target path (<code>/var/package/\$PKG_NAME/target/</code>). Webservice worker will move your <code>source</code> directory to <code>target</code> directory and set owner group according to your <code>user:group</code> specification. Note that you should specify a full path in <code>source</code> field.
target	3.0.0-0256	string	true	false	N/A	Your website root directory. <code>target</code> directory will be created under <code>web_packages</code> directory. Webservice worker will move <code>source</code> directory to <code>target</code> and set with corresponding owner group according to your <code>user:group</code> specification. You should only specify a relative path based on <code>web_packages</code> . Note that when <code>source</code> field is not specified, webservice worker will only create <code>target</code> directory and set owner group for <code>target</code> directory.
mode	3.0.0-0256	string	true	false	N/A	<code>target</code> directory access mode e.g. "0755", "0644" ... etc.
group	3.0.0-0256	string	true	false	N/A	Name of <code>target</code> directory group ownership.
user	3.0.0-0256	string	true	false	N/A	Name of <code>target</code> directory user ownership.

Web Service

Package could register to WebStation via WebStation `webapi` or Package Worker.

- Web Service support following types
 - static service
 - static web pages web services
 - `nginx_php` service
 - web service that using Nginx as HTTP server and PHP as scripts, e.g. phpMyAdmin
 - Will generate PHP Profile after service registered. You can modify it in WebStation -> Script Language Settings -> PHP.
 - `apache_php` service
 - web service that using Apache as HTTP server and PHP as scripts, e.g. WordPress
 - Will generate PHP Profile after service registered. You can modify it in WebStation -> Script Language Settings -> PHP.
 - `reverse_proxy` service
 - web service depending on reverse proxy, e.g. Docker-GitLab

common field

--	--	--	--	--	--	--

Key	Since	type	Required	Nullable	Default Value	Description
service	3.0.0-0214	string	true	false	N/A	service name
display_name	3.0.0-0214	string	true	false	N/A	service display name
display_name_i18n	3.0.0-0214	string	false	true	null	service displayed in different language (optional)
support_alias	3.0.0-0214	bool	false	false	true	Whether support alias portal, downgrade is not allowed
support_server	3.0.0-0214	bool	false	false	true	Whether suport server portal, downgrade is not allowed
icon	3.0.0-0214	string	false	true	null	icon path, relative path from package's target . Resolution should replaced in {0} . For now, we only support png format. Will use default icon if this field is empty.
type	3.0.0-0214	string	true	false	N/A	service type
php	3.0.0-0214	object	true	false	N/A	php-fpm setting including profile_name , backend , open_basedir , extensions , ... etc. The detail will be shown as following section.

Detail of php profile

Key	Since	type	Required	Nullable	Default Value	Description
profile_name	3.0.0-0214	string	true	false	N/A	Name of default php profile, user may not modify this field.
profile_desc	3.0.0-0214	string	true	false	N/A	Description of php profile
backend	3.0.0-0214	int	true	false	N/A	php version, 3 for PHP5.6, 4 for PHP7.0, 5 for PHP7.1, 6 for PHP7.2 and 7 for PHP7.3, user may not modify this field
open_basedir	3.0.0-0214	string	false	true	empty string	default php open_basediruser may modify this field.
extensions	3.0.0-0214	string array	false	true	empty array	default switched on php extension, user may not switch of these php extension; however, they may switch on others
php_settings	3.0.0-0214	object	false	true	empty object	key value pairs, define php ini setting, user may modify this field.
user	3.0.0-0256	string	true	false	N/A	Name of user with privilege while php-fpm accessing your website. Note that the value of user should be the same as pkg_dir_prepare user in order to access your website correctly.
group	3.0.0-0256	string	true	false	N/A	Name of group with privilege while php-fpm accessing your website. Note that the value of group should be the same as pkg_dir_prepare group in order to access your website correctly.

static service

When type is static, system will serve your package with nginx.

Key	Since	type	Required	Nullable	Default Value	Description
root	3.0.0-0214	string	true	false	N/A	service working directory, will be treated as absolute path if start with <code>/</code> , otherwise, relative path to <code>web_packages</code>
index	3.0.0-0214	string array	false	true	["index.html", "index.html"]	static service's index file. note use default value if <code>null</code> in this field
custom_rule	3.0.0-0214	object	false	true	empty object	Support customized routing rule. For more detail, please see Custom rule

static service worker setting example:

```
{
  "service": "static",
  "display_name": "static service",
  "support_alias": true,
  "support_server": true,
  "type": "static",
  "root": "static_dir",
  "icon": "ui/wordpress_{0}.png"
}
```

nginx_php service

When type is `nginx_php`, system will serve your package with nginx. The `php` file will be executed by `php-fpm`. `Php-fpm` default behavior can be defined in field `php`

Key	Since	type	Required	Nullable	Default Value	Description
root	3.0.0-0214	string	true	false	N/A	service working directory, will be treated as absolute path if start with <code>/</code> , otherwise, relative path to <code>web_packages</code>
index	3.0.0-0214	string array	false	true	["index.htm", "index.html", "index.php"]	nginx service's index file. note use default value if <code>null</code> in this field.
custom_rule	3.0.0-0214	object	false	true	empty object	Support customized routing rule. For more detail, please see Custom rule
connect_timeout	3.0.0-0214	int	false	false	60	timeout setting for connecting <code>php-fpm</code> , in units of second
read_timeout	3.0.0-0214	int	false	false	60	timeout setting for getting response from <code>php-fpm</code> , in units of second
send_timeout	3.0.0-0214	int	false	false	60	timeout setting for sending request to <code>php-fpm</code> , in units of second
php	3.0.0-0214	object	true	false	N/A	define default <code>php</code> profile

nginx_php service worker setting example:

```
{
  "service": "wordpress",
```

```
    "display_name": "WordPress",
    "support_alias": true,
    "support_server": true,
    "type": "nginx_php",
    "root": "wordpress",
    "icon": "ui/Wordpress_{0}.png",
    "php": {
      "profile_name": "WordPress Profile",
      "profile_desc": "PHP Profile for WordPress",
      "backend": 6,
      "open_basedir": "/var/services/web_packages/wordpress:/tmp:/var/services/tmp",
      "extensions": [
        "mysql",
        "mysqli",
        "pdo_mysql",
        "curl",
        "gd",
        "iconv"
      ],
      "php_settings": {
        "mysql.default_socket": "/run/mysqld/mysqld10.sock",
        "mysqli.default_socket": "mysqli.default_socket",
        "pdo_mysql.default_socket": "/run/mysqld/mysqld10.sock",
        "mysql.default_port": "3307",
        "mysqli.default_port": "3307"
      }
    },
    "connect_timeout": 60,
    "read_timeout": 3600,
    "send_timeout": 60
  }
}
```

apache_php service

When type is apache_php, nginx will pass request to apache server. The php file will be executed by php-fpm. Php-fpm default behavior can be defined in field php . Compare to nginx_php, apache_php with additional filed backend to specify apache version

Key	Since	type	Required	Nullable	Default Value	Description
backend	3.0.0-0214	int	true	false	N/A	1 (Apache2.2) or 2 (Apache2.4)
intercept_errors	3.0.0-0284	bool	false	false	true	true (on) or false (off)

apache_php service worker setting example:

```
{
  "service": "wordpress",
  "display_name": "WordPress",
  "support_alias": true,
  "support_server": true,
  "type": "apache_php",
  "root": "wordpress",
  "backend": 2,
  "icon": "ui/Wordpress_{0}.png",
  "php": {
    "profile_name": "WordPress Profile",
    "profile_desc": "PHP Profile for WordPress",
    "backend": 6,
    "open_basedir": "/var/services/web_packages/wordpress:/tmp:/var/services/tmp",
    "extensions": [
      "mysql",
      "mysqli",
      "pdo_mysql",
      "curl",
      "gd",
      "iconv"
    ]
  }
}
```

```

    ],
    "php_settings": {
        "mysql.default_socket": "/run/mysqld/mysqld10.sock",
        "mysqli.default_socket": "mysqli.default_socket",
        "pdo_mysql.default_socket": "/run/mysqld/mysqld10.sock",
        "mysql.default_port": "3307",
        "mysqli.default_port": "3307"
    },
    "intercept_errors": false,
    "connect_timeout": 60,
    "read_timeout": 3600,
    "send_timeout": 60
}

```

reverse_proxy service

When type is reverse_proxy, nginx will proxy request to target services

Key	Since	type	Required	Nullable	Default Value	Description
proxy_target	3.0.0-0214	string	true	false	N/A	Proxy target, support http, https, and unix. This value will be filled in nginx proxy_pass URL. For more detail please see proxy_pass
proxy_headers	3.0.0-0214	array	false	true	empty array	define proxy relay header value pair list
proxy_intercept_errors	3.0.0-0284	bool	false	false	false	specify whether letting nginx return error page for your packages if there's an error occur. Default is setting to <code>false</code>
proxy_http_version	3.0.0-0214	int	false	false	1	proxy http version, support 1.0 (0), 1.1 (1)
custom_rule	3.0.0-0214	object	false	true	empty object	define specific routing rule, should be compatible with support_alias and support_server setting. For more detail please see custom rule
connect_timeout	3.0.0-0214	int	false	false	60	timeout setting for connecting proxy target, in units of second
read_timeout	3.0.0-0214	int	false	false	60	timeout setting for getting response from proxy target, in units of second
send_timeout	3.0.0-0214	int	false	false	60	timeout setting for sending request to php-fpm, in units of second

You could define proxy header to modify proxy behavior, e.g. modify host or turn on websocket. If need support of websocket, you should specify Upgrade and Connection header as shown below:

Key	Since	type	Required	Nullable	Default Value	Description
name	3.0.0-0214	string	true	false	N/A	header name
value	3.0.0-0214	string	true	false	N/A	header value

reverse_proxy service worker setting example:

```

{
    "service": "gitlab",

```

```

    "display_name": "Git Lab",
    "support_alias": true,
    "support_server": true,
    "type": "reverse_proxy",
    "icon": "ui/gitlab_{0}.png",
    "proxy_target": "http://gitlab:30000",
    "proxy_headers": [{
      "name": "host",
      "value": "gitlab"
    }], {
      "name": "Upgrade",
      "value": "$http_upgrade"
    }, {
      "name": "Connection",
      "value": "$connection_upgrade"
    }
  ]
  "connect_timeout": 60,
  "read_timeout": 3600,
  "send_timeout": 60
}

```

Custom Rule

- You could modify config via `custom_rule` field in json key value format. Json key is target name, json value is target config's mustache file path
- You can reference `nginx_service_template.mustache`, `apache22_service_template.mustache` and `apache24_service_template.mustache` under `/var/packages/WebStation/target/misc` for routing rule that you can modify.
- Field `{{ @json key@ }}` in mustache template will be replaced by files specified in `custom_rule`
- You should consider the compatibility between `server` and `alias`, and could use `{{ #alias }}` to separate these two different routing rules.

Custom rule example:

```

"custom_rule": {
  "global_rule": "/var/packages/WordPress/target/misc/nginx_global.mustache",
  "fastcgi_rule": "/var/packages/WordPress/target/misc/nginx_fastcgi.mustache",
  "proxy_rule": "/var/packages/WordPress/target/misc/nginx_proxy.mustache",
  "apache_rule": "/var/packages/WordPress/target/misc/apache.mustache"
}

```

Custom rule type

key	affect target	affect service type	effect
global_rule	Nginx	all	modify service's request behavior
fastcgi_rule	Nginx	nginx_php	modify behavior of request passed to php-fpm
proxy_rule	Nginx	reverse_proxy	modify behavior of request passed to proxy target
apache_rule	Apache2.2 or Apache2.4 (depends on apache backend)	apache_php	modify apache behavior

Port

If your package service uses specific ports for communication (e.g. Surveillance Station uses ports 19997/udp for source port and 19998/udp for destination port), you should prepare a service configuration file for this package to describe which ports will be used. After that, once the user creates firewall rules or port forwarding rules from the built-in application, your package service will also be listed for selection.

Service Configure File Name

The file name should follow the naming convention **[package_name].sc** (ex: **SurveillanceStation.sc**). **[package_name]** should be the package name that is specified by the key "**package**" in the **INFO** file, and **sc** means Service Configure file.

Configure Format Template

Please see the following example:

```
[service_name]
title="English title"
desc="English description"
port_forward="yes" or "no"
src.ports="ports/protocols"
dst.ports="ports/protocols"

[service_name2]
...
```

Section/Key Descriptions

Please see the following statements for the strings and keys:

Section/Key	Description	Value	Default Value	DSM Requirement
service_name	<p><i>Required</i></p> <p>Usually a package only has one unique service name. If your package needs more than one port description, you can define <i>service_name2</i>, <i>service_name3</i>, ...</p> <p><i>Note: service_name cannot be empty and can only include characters "a~z", "A~Z", "0~9", "-", "\", "."</i></p>	Unique service name	N/A	4.0-2206
title	<p><i>Required</i></p> <p>English title which will be shown on field Protocol at firewall build-in selection menu.</p>	English title	N/A	4.0-2206
desc	<p><i>Required</i></p> <p>English description which will be shown on field Applications at firewall build-in selection menu.</p>	English description	N/A	4.0-2206
port_forward	<p><i>Optional</i></p> <p>If set to "yes," your package service related ports will be listed when users set port forwarding rule from build-in applications. Otherwise they will not</p>	"yes" or "no"	"no"	4.0-2206

	be listed.			
src.ports	<p><i>Optional</i></p> <p>If your package service has specified source ports, you can set them in this key. The value should contain at least the port numbers, and a default protocol that is tcp + udp.</p> <p>Ex: 6000,7000:8000/tcp,udp means source ports are 6000, 7000 to 8000, all ports are tcp + udp.</p>	<p>ports/protocols ports: 1~65535 (separated by ‘,’ and use ‘:’ to represent port range) protocols: tcp,udp (separated by ‘,’)</p>	<p>ports: N/A protocols: tcp,udp</p>	4.0-2206
dst.ports	<p><i>Required</i></p> <p>Each service should have destination ports. The value should contain at least the port numbers, and a default protocol that is tcp + udp.</p> <p>Ex: 6000,7000:8000/tcp,udp means destination ports are 6000, 7000 to 8000, all ports are tcp + udp.</p>	<p>ports/protocols ports: 1~65535 (separated by ‘,’ and use ‘:’ to represent port range) protocols: tcp,udp (separated by ‘,’)</p>	<p>ports: N/A protocols: tcp,udp</p>	4.0-2206

Please see the following example (SurveillanceStation.sc):

```
[ss_findhostd_port]
title="Search Surveillance Station"
desc="Surveillance Station"
port_forward="yes"
src.ports="19997/udp"
dst.ports="19998/udp"
```

After the service configuration file is ready, add the following content to the resource specification file. Please refer to [Port Config](#) for more detail.

```
"port-config": {
  "protocol-file": "port_conf/xxdns.sc"
}
```

Check port conflict

Before trying to change a port number, you would need to check if the port number was already in use.

How to check if the port number was in use

Assume the package named Dhcp Server and the port-config DhcpServer.sc contains:

```
[dhcp_udp]
title="DHCP Server"
title_key="DHCP Server"
desc="DHCP Server"
desc_key="DHCP Server"
port_forward="no"
dst.ports="67,68/udp"
```

Please run the following instructions to check if the port is in use while you are trying to change the port number from 67 to 667

```
servicetool --conf-port-conflict-check --tcp 667
```

The output would look like this:

```
root@dev:~# servicetool --conf-port-conflict-check --tcp 667
IsConflict: false      Port: 667      Protocol: tcp  ServiceName: (null)
root@dev:~#
```

The return code does not indicate port occupation, you need to parse the standard output to extract the *IsConflict* value.

If the *IsConflict* value is *false*, you can use that port number safely.

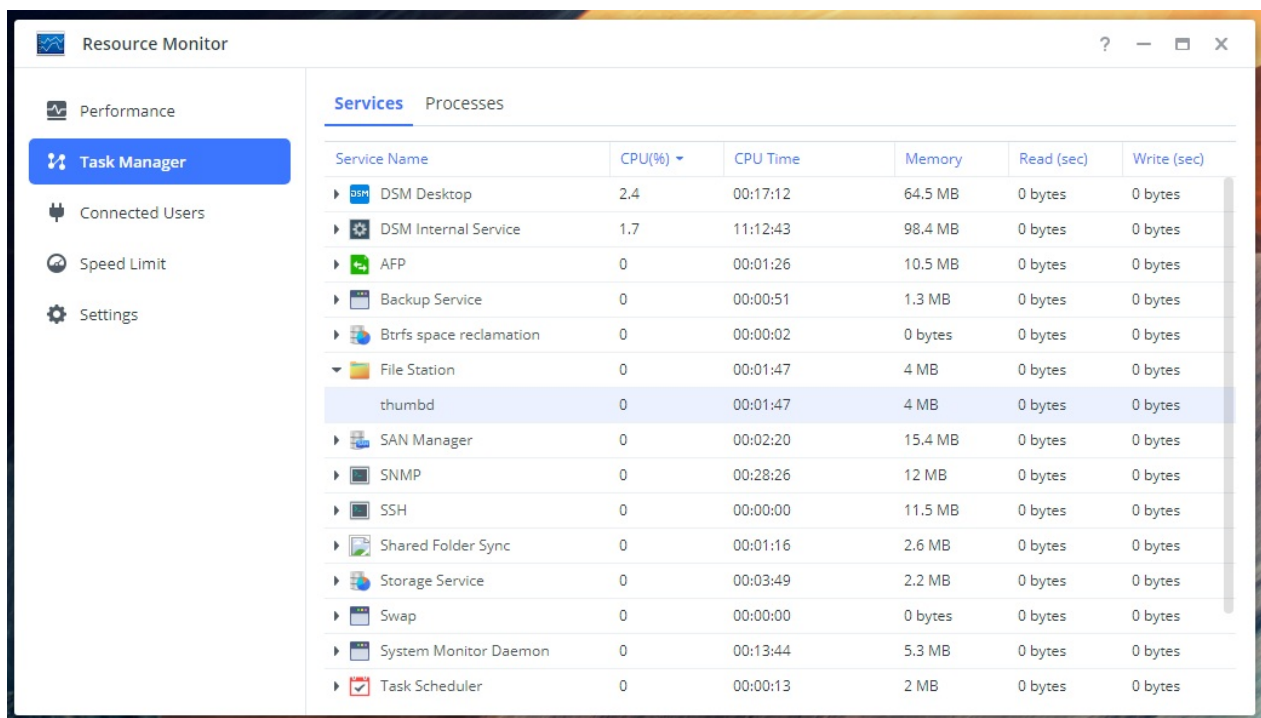
Monitor

The DSM manages resource by slices or processes. It requires the information "who owns this process". For packages, they should tell DSM which daemon belongs to them.

All you have to do is to fill the `slice` field in your systemd unit with `[package_name].slice`. Here is an example field from units for MyPackage:

```
...
[Service]
Slice=MyPackage.slice
...
```

If the field is properly set, you should be able to see your package shown on the resource monitor.



Service Name	CPU(%)	CPU Time	Memory	Read (sec)	Write (sec)
DSM Desktop	2.4	00:17:12	64.5 MB	0 bytes	0 bytes
DSM Internal Service	1.7	11:12:43	98.4 MB	0 bytes	0 bytes
AFP	0	00:01:26	10.5 MB	0 bytes	0 bytes
Backup Service	0	00:00:51	1.3 MB	0 bytes	0 bytes
Btrfs space reclamation	0	00:00:02	0 bytes	0 bytes	0 bytes
File Station	0	00:01:47	4 MB	0 bytes	0 bytes
thumbd	0	00:01:47	4 MB	0 bytes	0 bytes
SAN Manager	0	00:02:20	15.4 MB	0 bytes	0 bytes
SNMP	0	00:28:26	12 MB	0 bytes	0 bytes
SSH	0	00:00:00	11.5 MB	0 bytes	0 bytes
Shared Folder Sync	0	00:01:16	2.6 MB	0 bytes	0 bytes
Storage Service	0	00:03:49	2.2 MB	0 bytes	0 bytes
Swap	0	00:00:00	0 bytes	0 bytes	0 bytes
System Monitor Daemon	0	00:13:44	5.3 MB	0 bytes	0 bytes
Task Scheduler	0	00:00:13	2 MB	0 bytes	0 bytes

Package Examples

Compile Open Source Project

This chapter will show you how to build an open source project for your DSM system using Package Toolkit. If you wish to compile the open source project manually, please refer to [Appendix B: Compile Open Source Project Manually](#).

You have to create SynoBuildConf/build, SynoBuildConf/install, and SynoBuildConf/depends before using Package Toolkit.

Unlike the previous example, compiling an application on most open source projects may require executing the following three steps:

1. `configure`
2. `make`
3. `make install`

The configure script consists of many lines which are used to check some details about the machine where the software is going to be installed. This script will also check a lot of dependencies on your system. When you run the configure script, you will see a lot of output on the screen, each being some sort of question with a respective yes/no as a reply. If any of the major requirements are missing on your system, the configure script will exit and you will not be able to proceed with the installation until you meet the required conditions. In most cases, compile applications on some particular target machines will require you to modify the configure script manually to provide the correct values.

When running the configure script to configure software packages for cross-compiling, you will need to specify the `CC`, `LD`, `RANLIB`, `CFLAGS`, `LDFLAGS`, `host`, `target`, and `build`.

In this chapter, we will use **platform avoton** as our example.

Preparation:

First download the tmux source code from the official [github site](#) or you can download **example tmux package project** from this [link](#).

Note: The archive file you've downloaded from the above links is different from the official tmux source code. We have added the necessary build scripts.

Project Layout:

```
tmux/
├── tmux related source code
├── SynoBuildConf/
│   ├── build
│   ├── depends
│   └── install
├── synology
│   ├── conf/
│   ├── scripts/
│   └── INFO.sh
```

SynoBuildConf/depends:

The following is the **depends** file for this example. There is nothing special about the **depends** file.

```
[default]
all="7.0"
```

SynoBuildConf/build:

The build script is slightly different from the previous one. Here you will have to pass the following environment variables to configure:

- CC
- AR
- CFLAGS
- LDFLAGS

In addition, since tmux is dependent on ncurses, you will need to use `pkg-config` to resolve the necessary header files and libraries for tmux.

The following is an example of SynoBuildConf/build:

```
#!/bin/sh
# Copyright (c) 2000-2021 Synology Inc. All rights reserved.

case ${MakeClean} in
    [Yy][Ee][Ss])
        make distclean
        ;;
    esac

NCURSES_INCS="$(pkg-config ncurses --cflags)"
NCURSES_LIBS="$(pkg-config ncurses --libs)"

CFLAGS="${CFLAGS} ${NCURSES_INCS}"
LDFLAGS="${LDFLAGS} ${NCURSES_LIBS}"

autoreconf -if

env CC="${CC}" AR="${AR}" CFLAGS="${CFLAGS}" LDFLAGS="${LDFLAGS}" \
./configure ${ConfigOpt}

make ${MAKE_FLAGS}
```

SynoBuildConf/install

Instead of copying the binary to the destination folder, most big projects will use `make install` to install the binaries and libraries. You can pass the DESTDIR environment variable to specify where you want to install the binaries and libraries.

```
#!/bin/bash
# Copyright (c) 2000-2021 Synology Inc. All rights reserved.

PKG_NAME="tmux"
INST_DIR="/tmp/${PKG_NAME}"
PKG_DIR="/tmp/${PKG_NAME}_pkg"
PKG_DEST="/image/packages"

PrepareDirs() {
    for dir in $INST_DIR $PKG_DIR; do
        rm -rf "$dir"
    done
    for dir in $INST_DIR $PKG_DIR $PKG_DEST; do
        mkdir -p "$dir"
    done
}

InstallTmux() {
    DESTDIR="${INST_DIR}" make install
}

GenerateINFO() {
    synology/INFO.sh > INFO
    cp INFO "${PKG_DIR}"
}

InstallSynologyConfig(){
```

```

cp -r synology/scripts/ "${PKG_DIR}"
cp -r synology/conf/ "${PKG_DIR}"
cp synology/PACKAGE_ICON{,_256}.PNG "${PKG_DIR}"
}

MakePackage() {
    source /pkgscripts/include/pkg_util.sh
    pkg_make_package $INST_DIR $PKG_DIR
    pkg_make_spk $PKG_DIR $PKG_DEST
}

main() {
    PrepareDirs
    InstallTmux
    GenerateINFO
    InstallSynologyConfig
    MakePackage
}

main "$@"

```

INFO.sh

As mentioned before, we will use INFO.sh to generate the INFO file.

```

#!/bin/sh
# Copyright (c) 2000-2021 Synology Inc. All rights reserved.
. /pkgscripts/include/pkg_util.sh
package="tmux"
version="1.9.1-1001"
os_min_ver="7.0-40850"
displayname="tmux"
arch="$(pkg_get_platform) "
maintainer="Synology Inc."
description="Tmux package for Synology DSM."
support_url="https://github.com/tmux/tmux"
thirdparty="yes"
startable="no"
silent_install="yes"
silent_upgrade="yes"
silent_uninstall="yes"
[ "$(caller)" != "0 NULL" ] && return 0
pkg_dump_info

```

Note: Remember to set the executable bit of INFO.sh file.

Build and Create Package:

Run the following commands to compile the source code and build the package.

```
/toolkit/pkgscripts-ng/PkgCreate.py -p avoton -c tmux
```

After the build process, you can check the result in `/toolkit/result_spk`.

Verify the Result

If the building process was successful, you will see that the .spk file has been placed under result_spk folder. To test the spk file, You can use manual install from Package Center then connect to DSM via ssh to try `tmux` command.

If you failed to install the package, it is possible to find out the error logs at `/var/log/messages`.

References

- [Toolkit](#)
- [Package Format](#)
- [Privilege](#)
- [Resource](#)

Compile Open Source Project: nmap

This chapter will show you how to build an open source project for your DSM system using Package Toolkit.

The open source project that we are going to build in this example is **nmap**, a network scanning program. We will use **avoton** as our build environment platform.

If you wish to compile an open source project manually, please refer to [Appendix B: Compile Open Source Project Manually](#).

You have to create the SynoBuildConf/build, SynoBuildConf/install, and SynoBuildConf/depends before using Package Toolkit.

Unlike the previous example, compiling an application on most open source projects may require executing the following three steps:

1. `configure`
2. `make`
3. `make install`

The configure script consists of many lines which are used to check some details about the machine where the software is going to be installed. This script will also check a lot of dependencies on your system. When you run the configure script, you will see a lot of output on the screen, each being some sort of question with a respective yes/no as a reply. If any of the major requirements are missing on your system, the configure script will exit and you will not be able to proceed with the installation until you meet the required conditions. In most cases, compile applications on some particular target machines will require you to modify the configure script manually to provide the correct values.

When running the configure script to configure software packages for cross-compiling, you will need to specify the `CC`, `LD`, `RANLIB`, `CFLAGS`, `LDFLAGS`, `host`, `target`, and `build`.

Preparation:

You can download the projects by following commands:

```
git clone https://github.com/SynologyOpenSource/ExamplePackages.git
cp -a ExamplePackages/libpcap /toolkit/source
cp -a ExamplePackages/nmap /toolkit/source
```

Our nmap & libpcap source code come from here:

```
wget https://nmap.org/dist/nmap-7.91.tar.bz2
wget http://www.tcpdump.org/release/libpcap-1.9.1.tar.gz
```

Project Layout:

After you download the source code, your toolkit layout should look like the following figure.

```
/toolkit/
├─ build_env/
│   └─ ds.${platform}-${version}/
│       └─ /usr/syno/
│           ├── bin
│           ├── include
│           └─ lib
├─ pkgscripts-ng/
└─ source/
    └─ nmap/
        ├── nmap related source code
        ├── SynoBuildConf/
        │   ├── build
        │   └─ depends
```

```

| | └─ install
| └─ synology
|     └─ PACKAGE_ICON.PNG
|     └─ PACKAGE_ICON_256.PNG
|     └─ INFO.sh
|     └─ conf/
|         └─ privilege
|         └─ resource
|     └─ scripts/
└─ libpcap/
    └─ libpcap related source code
    └─ Makefile
    └─ SynoBuildConf/
        └─ build
        └─ depends
        └─ install-dev
        └─ install

```

The file, **install-dev**, is a special file which we will be covered in the following section.

SynoBuildConf/depends:

The SynoBuildConf/depends for nmap is slightly different from the previous example. Since nmap depends on libpcap, we have to add the value to the BuildDependent field, so that the PkgCreate.py can resolve the dependency and compile the project in the correct order.

The **depends** file for nmap is as follows.

```

[BuildDependent]
libpcap

[default]
all="7.0"

```

However, the SynoBuildConf/depends for libpcap is the same as the Hello World Example.

```

[BuildDependent]

[default]
all="7.0"

```

SynoBuildConf/build:

The SynoBuildConf/build script is also different from the previous one.

Here you will have to pass several environment variables to configure, so that nmap can be compiled properly

- CC
- CXX
- LD
- AR
- STRIP
- RANLIB
- NM
- CFLAGS
- CXXFLAGS
- LDFLAGS

Since nmap will be compiled with many features by default, we will need to disable some of them to make it clean. The following list contains the features that will be disabled:

- ndiff
- zenmap
- nping
- ncat
- nmap-update
- liblua

Note: If you are interested in some of the above features and you want to enable them, just change the `--without-${feature}` into `--with-${feature}` .

The following is the SynoBuildConf/build for nmap

```
#!/bin/sh
# Copyright (c) 2000-2021 Synology Inc. All rights reserved.

PKG_NAME=nmap
INST_DIR=/tmp/_${PKG_NAME}

case ${MakeClean} in
    [Yy][Ee][Ss])
        make distclean
        ;;
    esac

LDFLAGS+=$(shell pkg-config --libs libnl libnl-genl)

env CC="${CC}" CXX="${CXX}" LD="${LD}" AR=${AR} STRIP=${STRIP} RANLIB=${RANLIB} NM=${NM} \
    CFLAGS="${CFLAGS}" CXXFLAGS="${CXXFLAGS} $CFLAGS" \
    LDFLAGS="${LDFLAGS} -ldbus-1" \
    ./configure ${ConfigOpt} \
    --prefix=${INST_DIR} \
    --without-ndiff \
    --without-zenmap \
    --without-nping \
    --without-ncat \
    --without-nmap-update \
    --without-liblua \
    --with-libpcap=/usr/local

make ${MAKE_FLAGS}
```

In this example, `--with-libpcap` is assigned with value `/usr/local` . We need to install libpcap's cross compiled product into `"usr/local"` so that nmap's configure can retrieve libpcap correctly.

The following is the SynoBuildConf/build for libpcap.

```
#!/bin/bash
# Copyright (c) 2000-2021 Synology Inc. All rights reserved.

case ${MakeClean} in
    [Yy][Ee][Ss])
        make distclean
        ;;
    esac

case ${CleanOnly} in
    [Yy][Ee][Ss])
        return
        ;;
    esac

# prefix with /usr/local, all files will be installed into /usr/local
env CC="${CC}" CXX="${CXX}" LD="${LD}" AR=${AR} STRIP=${STRIP} RANLIB=${RANLIB} NM=${NM} \
    CFLAGS="${CFLAGS} -Os" CXXFLAGS="${CXXFLAGS}" LDFLAGS="${LDFLAGS}" \
    ./configure ${ConfigOpt} \
    --with-pcap=linux --prefix=/usr/local
```

```
make ${MAKE_FLAGS}

make install
```

The above script will install libpcap related files into `/usr/local/` in chroot environment. After installing libpcap, nmap can find libpcap's cross compiled products in `/usr/local`.

Synology toolkit provides `libpcap` in chroot.

```
> dpkg -l | grep libpcap
ii  libpcap-avoton-dev          7.0-7274      all          Synology build-time library
```

nmap can use chroot's libpcap by using `${SysRootPrefix}` variable.

```
--with-libpcap=${SysRootPrefix}
```

SynoBuildConf/install

Instead of copying the binary to the destination folder, most big projects will use `make install` to install the binaries and libraries. Since we have used the `--prefix` flag when configuring the nmap project, we can just execute **make install** and it will install the nmap related files to the folder specified by `--prefix`.

```
#!/bin/bash
# Copyright (c) 2000-2021 Synology Inc. All rights reserved.

PKG_NAME="nmap"
INST_DIR="/tmp/${PKG_NAME}"
PKG_DIR="/tmp/${PKG_NAME}_pkg"
PKG_DEST="/image/packages"

PrepareDirs() {
    for dir in $INST_DIR $PKG_DIR; do
        rm -rf "$dir"
    done
    for dir in $INST_DIR $PKG_DIR $PKG_DEST; do
        mkdir -p "$dir"
    done
}

SetupPackageFiles() {
    make install
    synology/INFO.sh > INFO
    cp INFO "$PKG_DIR"
    cp -r synology/conf/ "$PKG_DIR"
    cp -r synology/scripts/ "$PKG_DIR"
    cp synology/PACKAGE_ICON{,_256}.PNG "$PKG_DIR"
}

MakePackage() {
    source /pkgscripts-ng/include/pkg_util.sh
    pkg_make_package $INST_DIR $PKG_DIR
    pkg_make_spk $PKG_DIR $PKG_DEST
}

main() {
    PrepareDirs
    SetupPackageFiles
    MakePackage
}

main "$@"
```

conf/resource

```
{
  "usr-local-linker": {
    "bin": ["bin/nmap"]
  }
}
```

conf/privilege

```
{
  "defaults": {
    "run-as": "package"
  }
}
```

INFO.sh

As mentioned before, we will use INFO.sh to generate the INFO file.

```
#!/bin/sh
# Copyright (c) 2000-2021 Synology Inc. All rights reserved.

. /pkgscripts-ng/include/pkg_util.sh
package="nmap"
version="7.91-1001"
os_min_ver="7.0-40850"
displayname="nmap"
arch="$(pkg_get_platform) "
maintainer="Synology Inc."
description="This package will install nmap in your DSM system."
[ "${caller}" != "0 NULL" ] && return 0
pkg_dump_info
```

Note: Remember to set the executable bit of INFO.sh file.

Build and Create Package:

Lastly, run the following commands to compile the source code and build the package.

```
/toolkit/pkgscripts-ng/PkgCreate.py -p avoton -x0 -c nmap
```

After the build process, you can check the result in `/toolkit/result_spk` .

Verify the Result

If the packing process was successful, you will see an spk file placed in the result_spk folder. To test the spk file, You can use manual install from Package Center then connect to DSM via ssh to try `nmap -v -A localhost` command.

If you failed to install the package, it is possible to find out the error logs at `/var/log/messages` .

References

- [Toolkit](#)
- [Package Format](#)
- [Privilege](#)
- [Resource](#)

Compile Docker Package - Gitlab

This chapter will show how to compile a docker package by using a well known version control opensource - Gitlab.

To create a Gitlab docker container, you only need to depends on Docker package and fill in docker worker configuration and the worker will do the reset for you.

As mentioned before, you have to create **SynoBuildConf/build**, **SynoBuildConf/install** and **SynoBuildConf/depends** for packing spk. However, since docker package will pull images or build image on the DSM, We don't need to build any code while packing the spk.

Project Layout:

```
docker-gitlab
├── conf
│   ├── privilege
│   └── resource
├── INFO.sh
├── scripts
│   ├── postinst
│   ├── postuninst
│   ├── postupgrade
│   ├── preinst
│   ├── preuninst
│   ├── preupgrade
│   ├── script_customized
│   └── start-stop-status
├── SynoBuildConf
│   ├── build
│   ├── depends
│   └── install
└── ui
    ├── config.png
    ├── Gitlab_120.png
    ├── Gitlab_16.png
    ├── Gitlab_24.png
    ├── Gitlab_256.png
    ├── Gitlab_32.png
    ├── Gitlab_48.png
    ├── Gitlab_64.png
    └── Gitlab_72.png
```

INFO.sh:

We will use INFO.sh to generate the INFO file. The following is the **INFO.sh** file for this example. For more details of each key's purpose, please see [INFO](#).

```
#!/bin/bash
# Copyright (c) 2000-2021 Synology Inc. All rights reserved.

package="wordpress_sample"

. "/pkgscripts-ng/include/pkg_util.sh"
version="12.9.0-1"
os_min_ver="7.0-40337"
install_dep_packages="Docker>=18.09.0-1017"
maintainer="Gitlab"
thirdparty="yes"
arch="avoton"
reloadui="yes"
adminurl="wordpress"
dsmuidir="ui"
```

```
displayname="Gitlab"
package_icon="`/pkgscripts-ng/include/base64.php ${ICON_PATH}`"

[ "${caller}" != "0 NULL" ] && return 0
pkg_dump_info
```

SynoBuildConf/depends:

The following is the **depends** file for this example.

```
[default]
all="7.0"
```

SynoBuildConf/build:

The following is the **build** file for this example. Since WordPress is depends on PHP, there is nothing to do in **build**.

```
#!/bin/bash
# Copyright (c) 2000-2021 Synology Inc. All rights reserved.

case ${MakeClean} in
    [Yy][Ee][Ss])
        make clean
        ;;
    esac

case ${CleanOnly} in
    [Yy][Ee][Ss])
        return
        ;;
    esac

make ${MAKE_FLAGS}
```

SynoBuildConf/install:

The following is the **install** file for this example.

```
#!/bin/bash
# Copyright (c) 2000-2021 Synology Inc. All rights reserved.

# set include projects to install into this package
INST_DIR="/tmp/_Gitlab"      # temp folder for dsm files
PKG_DIR="/tmp/_Gitlab_pkg"   # temp folder for package files
PKG_DEST="/image/packages"

# prepare install and package dir
for dir in $INST_DIR $PKG_DIR; do
    rm -rf "$dir"
done
for dir in $INST_DIR $PKG_DIR $PKG_DEST; do
    mkdir -p "$dir" # use default mask
done

[ -d $INST_DIR/ui ] || install -d $INST_DIR/ui
cp -a ui/* $INST_DIR/ui

[ -d $PKG_DIR ] || install -d $PKG_DIR
[ -d $PKG_DIR/scripts ] || install -d $PKG_DIR/scripts
cp -a conf $PKG_DIR
cp -a scripts/* $PKG_DIR/scripts
chmod 755 $PKG_DIR/scripts/*
```

```
./INFO.sh > INFO

install -c -m 644 INFO $PKG_DIR

. "/pkgscripts-ng/include/pkg_util.sh"
pkg_make_package $INST_DIR $PKG_DIR
pkg_make_spk $PKG_DIR $PKG_DEST
```

UI config:

UI config is placed in `ui` folder.

```
{
  ".url": {
    "SYNO.SDS.GitLab": {
      "allUsers": true,
      "desc": "Docker-GitLab",
      "icon": "images/Docker_GitLab_SynoCommunity-{0}.png",
      "port": "@PORT@",
      "protocol": "http",
      "texts": "texts",
      "title": "GitLab",
      "type": "url",
      "url": "/"
    }
  }
}
```

Scripts:

The following are spk scripts for installing docker Gitlab spk into DSM.

- preinst: There is nothing to do for `preinst` in this example. You can customize your own `preinst` script to fit your circumstances.

```
#!/bin/sh

exit 0
```

- postinst: In `postinst` stage, we set up port in ui/config after user specify in install wizard.

```
#!/bin/sh
PKG_NAME="Gitlab"
PORT_CONFIG_FILE="/var/packages/$PKG_NAME/etc/port_config"

port=""
if [ ! -z "$wizard_http_port" ]; then
  # new install
  port="$wizard_http_port"
elif [ -f "$PORT_CONFIG_FILE" ]; then
  # upgrade
  port=$(get_key_value "$PORT_CONFIG_FILE" port)
fi

echo "port=$port" > $PORT_CONFIG_FILE

if [ -f "$SYNOPKG_PKGDEST/app/config" ]; then
  sed -i "s/@PORT@/$port/g" "$SYNOPKG_PKGDEST/ui/config"
fi

exit 0
```

- `preuninst`: There is nothing to do in `preuninst` in this example. You can customize your own `preuninst` script to fit your circumstances.

```
#!/bin/sh

exit 0
```

- `postuninst`: In `postuninst` stage, we remove gitlab port configuration file.

```
#!/bin/sh
PKG_NAME="Gitlab"
PORT_CONFIG_FILE="/var/packages/$PKG_NAME/etc/port_config"

if [ "$SYNOPKG_STATUS" = "UNINSTALL" ]; then
    rm -f "$PORT_CONFIG_FILE"
fi

exit 0
```

- `preupgrade`: There is nothing to do in `preupgrade` in this example. You can customize your own `preupgrade` script for upgrade purpose.

```
#!/bin/sh

exit 0
```

- `postupgrade`: There is nothing to do in `postupgrade` in this example. You can customize your own `postupgrade` script for upgrade purpose.

```
#!/bin/sh

exit 0
```

- `start-stop-status`: For `start-stop-status` in this example. You could call `docker_inspect` to see if your container is running.

```
#!/bin/bash
GITLAB_NAME="GitLab"
DOCKER_INSPECT="/usr/local/bin/docker_inspect"

case "$1" in
    start)
        ;;
    stop)
        ;;
    status)
        "$DOCKER_INSPECT" "$GITLAB_NAME" | grep -q "\"Status\": \"running\"," || exit 1
        ;;
    log)
        echo ""
        ;;
    *)
        echo "Usage: $0 {start|stop|status}" >&2
        exit 1
        ;;
esac
exit 0
```

Privilege:

The following is the **privilege** file under `conf` directory. The **privilege** file is configuration for specifying the installation and run time privilege. The detail of privilege will be elaborated under [privilege](#) section.


```
{
  "defaults": {
    "run-as": "package"
  },
  "username": "Gitlab"
}
```

Worker:

The following is the **resource** file under `conf` directory. The **resource** file are configurations for calling workers. In this example, since docker package only need docker worker to prepare container for them, we write docker worker configuration for setting up Gitlab container. For more details, please see [docker worker](#).

```
{
  "docker": {
    "services": [{
      "service": "gitlab",
      "image": "gitlab/gitlab-ce",
      "container_name": "GitLab",
      "tag": "12.9.0-ce.0",
      "restart": "always",
      "shares": [{
        "host_dir": "gitlab/data",
        "mount_point": "/var/opt/gitlab"
      }, {
        "host_dir": "gitlab/logs",
        "mount_point": "/var/log/gitlab"
      }, {
        "host_dir": "gitlab/config",
        "mount_point": "/etc/gitlab"
      }],
      "ports": [{
        "host_port": "{{wizard_http_port}}",
        "container_port": "80",
        "protocol": "tcp"
      }, {
        "host_port": "{{wizard_https_port}}",
        "container_port": "443",
        "protocol": "tcp"
      }, {
        "host_port": "{{wizard_ssh_port}}",
        "container_port": "22",
        "protocol": "tcp"
      }
    ]
  }
}
```

Build and Create Package

Run the following command to build your source code into package.

```
/toolkit/pkgscripts-ng/PkgCreate.py -p avoton -c docker-gitlab
```

After the build process, you can check the result in `/toolkit/result_spk` .

Verify the Result

If the building process was successful, you will see that the .spk file has been placed under result_spk folder. To test the spk file, you can use manual install in Package Center to install your package.

Compile Web Package - WordPress

This chapter will use well known open source project - WordPress as an example to show you how to build a php based web package integrating with DSM Packages -- WebStation, MariaDB and Apache server.

WordPress is the largest self-hosted blogging Open Source Project that have been used by millions of websites. All it need is a PHP web server and a database, then you can build your own blogging website. In this example, we will use WebStation and Apache as web server to host WordPress, and use MariaDB as database. Once the website was setted up, you could modify web server configurations for WordPress via WebStation UI.

As mentioned before, you have to create **SynoBuildConf/build**, **SynoBuildConf/install**, **SynoBuildConf/depends** and WordPress source project before creating spk. However, since WordPress depends on PHP, we don't have to compile any source code.

Preparation:

First you need to download WordPress from official website and unarchive it into your spk source project. In this example, we put it under **src** as shown in [Project Layout](#).

Secondly, before installing your WordPress spk, you need to download the dependant packages such as WebStation, MariaDB, PHP7.2 and Apache2.2 in DSM from Package Center. Noted that we use PHP7.2 and Apache2.2 in this example, you can choose whatever you want in considering your circumstances.

Third, according to instructions from WordPress official website, you have to setup DB information for WordPress. For more details, please see [WordPress - how ot install wordpress](#).

Project Layout:

```
/toolkit/source/wordpress_sample
├─ PACKAGE_ICON.PNG
├─ PACKAGE_ICON_256.PNG
├─ conf
│   └─ privilege
│   └─ resource
├─ INFO.sh
├─ Makefile
├─ scripts
│   └─ postinst
│   └─ postuninst
│   └─ postupgrade
│   └─ preinst
│   └─ preuninst
│   └─ preupgrade
│   └─ script_customized
│   └─ start-stop-status
├─ src
│   └─ wordpress
│       └─ wp-admin
├─ SynoBuildConf
│   └─ build
│   └─ depends
│   └─ install
└─ ui
    └─ wordpress_120.png
    └─ wordpress_16.png
    └─ wordpress_24.png
    └─ wordpress_256.png
    └─ wordpress_32.png
    └─ wordpress_48.png
    └─ wordpress_64.png
    └─ wordpress_72.png
```

INFO.sh (this file should have executable permission: `chmod +x INFO.sh`):

As mentioned before, we will use `INFO.sh` to generate the `INFO` file. The following is the **INFO.sh** file for this example. For more details of each key's purpose, please see [INFO](#).

```
#!/bin/bash
# Copyright (c) 2000-2021 Synology Inc. All rights reserved.

package="wordpress_sample"

. "/pkgscripts-ng/include/pkg_util.sh"
version="5.5.1-1001"
os_min_ver="7.0-40337"
startstop_restart_services="nginx.service"
instuninst_restart_services="nginx.service"
install_dep_packages="WebStation>=3.0.0-0226:MariaDB10:PHP7.3>=7.3.16-0150:Apache2.2>=2.2.34-0104"
install_provide_packages="WEBSTATION_SERVICE"
maintainer="WordPress"
thirdparty="yes"
silent_upgrade="yes"
arch="noarch"
reloadui="yes"
adminprotocol="http"
adminport="80"
adminurl="wordpress"
dsmuidir="ui"

[ "${caller}" != "0 NULL" ] && return 0
pkg_dump_info
```

SynoBuildConf/depends:

The following is the **depends** file for this example.

```
[default]
all="7.0"
```

SynoBuildConf/build:

The following is the **build** file for this example. Since WordPress depends on PHP, there is nothing to do in **build**.

```
#!/bin/bash
# Copyright (c) 2000-2021 Synology Inc. All rights reserved.

case ${MakeClean} in
    [Yy][Ee][Ss])
        make clean
        ;;
    esac

case ${CleanOnly} in
    [Yy][Ee][Ss])
        return
        ;;
    esac

make ${MAKE_FLAGS}
```

SynoBuildConf/install:

The following is the **install** file for this example. In this example, we install our package with the help of **Makefile**.

```
#!/bin/bash
# Copyright (c) 2000-2021 Synology Inc. All rights reserved.

# set include projects to install into this package
INST_DIR="/tmp/_WordPress"      # temp folder for dsm files
PKG_DIR="/tmp/_WordPress_pkg"   # temp folder for package files
PKG_DEST="/image/packages"

# prepare install and package dir
for dir in $INST_DIR $PKG_DIR; do
    rm -rf "$dir"
done
for dir in $INST_DIR $PKG_DIR $PKG_DEST; do
    mkdir -p "$dir" # use default mask
done

make INSTALLDIR=$INST_DIR install
make PACKAGEDIR=$PKG_DIR package

. "/pkgscripts-ng/include/pkg_util.sh"
pkg_make_package $INST_DIR $PKG_DIR
pkg_make_spk $PKG_DIR $PKG_DEST
```

Makefile:

The following is the **Makefile** file for this example. **Watch out the indent must be tab instead of space.**

```
WORDPRESSDIR=src
WORDPRESS_INSTALL_DIR=$(INSTALLDIR)/$(WORDPRESSDIR)

all clean:

.PHONY:

install:
    [ -d $(INSTALLDIR) ] || install -d $(INSTALLDIR)
    [ -d $(WORDPRESS_INSTALL_DIR) ] || install -d $(WORDPRESS_INSTALL_DIR)
    cp -a $(WORDPRESSDIR)/* $(WORDPRESS_INSTALL_DIR)

    [ -d $(INSTALLDIR)/ui ] || install -d $(INSTALLDIR)/ui
    cp -a ui/* $(INSTALLDIR)/ui

    # change owner to nobody user/group on DS
    chown -R http:http $(WORDPRESS_INSTALL_DIR)

INFO: INFO.sh
    env UISTRING_PATH=$(STRING_DIR) ./INFO.sh > INFO

package: INFO
    [ -d $(PACKAGEDIR) ] || install -d $(PACKAGEDIR)
    [ -d $(PACKAGEDIR)/scripts ] || install -d $(PACKAGEDIR)/scripts
    cp -a scripts/* $(PACKAGEDIR)/scripts
    chmod 755 $(PACKAGEDIR)/scripts/*

    cp -a PACKAGE_ICON.PNG $(PACKAGEDIR)
    cp -a PACKAGE_ICON_256.PNG $(PACKAGEDIR)
    cp -a conf $(PACKAGEDIR)
    install -c -m 644 INFO $(PACKAGEDIR)

clean:
```

Scripts (these files should have executable permission):

The following are spk scripts for installing WordPress spk into DSM.

- `preinst`: There is nothing to do for `preinst` in this example. You can customize your own `preinst` script to fit your circumstances.

```
#!/bin/sh

exit 0
```

- `postinst`: In `postinst` stage, we move the source project into `/var/services/web_packages` since it's Web Station's working directory.

```
#!/bin/sh
WEBSITE_ROOT="/var/services/web_packages/wordpress"

chown -R WordPress:http "$WEBSITE_ROOT/*"

exit 0
```

- `preuninst`: There is nothing to do in `preuninst` in this example. You can customize your own `preuninst` script to fit your circumstances.

```
#!/bin/sh

exit 0
```

- `postuninst`: In `postuninst` stage, we remove source project from `/var/services/web_packages`.

```
#!/bin/sh

exit 0
```

- `preupgrade`: There is nothing to do in `preupgrade` in this example. You can customize your own `preupgrade` script for upgrade purpose.

```
#!/bin/sh

exit 0
```

- `postupgrade`: There is nothing to do in `postupgrade` in this example. You can customize your own `postupgrade` script for upgrade purpose.

```
#!/bin/sh

exit 0
```

- `start-stop-status`: There is nothing to do in `start-stop-status` in this example. You can customize your own `start-stop-status` script by following the template.

```
#!/bin/sh

case "$1" in
    start)
        exit 0
    ;;
```

```

stop)
    exit 0
    ;;

status)
    exit 0
    ;;

*)
    exit 1
    ;;

esac

```

Privilege:

The following is the **privilege** file under `conf` directory. The **privilege** file is configuration for specifying the installation and run time privilege. The detail of privilege will be elaborated under [privilege](#) section.

```

{
  "defaults": {
    "run-as": "package"
  },
  "username": "WordPress",
  "join-groupname": "http"
}

```

Worker:

The following is the **resource** file under `conf` directory. The **resource** file are configurations for calling workers. In this example, since we would like to integrate WordPress with WebStation, we will call WebStation's worker to run specific setup during installation. For more details, please see [webservice](#).

```

{
  "webservice": {
    "services": [{
      "service": "wordpress",
      "display_name": "WordPress",
      "support_alias": true,
      "support_server": true,
      "type": "apache_php",
      "root": "wordpress",
      "backend": 1,
      "icon": "ui/Wordpress_{0}.png",
      "php": {
        "profile_name": "WordPress Profile",
        "profile_desc": "PHP Profile for WordPress",
        "backend": 7,
        "open_basedir": "/var/services/web_packages/wordpress:/tmp:/var/services/tmp",
        "extensions": [
          "mysql",
          "mysqli",
          "pdo_mysql",
          "curl",
          "gd",
          "iconv"
        ],
        "php_settings": {
          "mysql.default_socket": "/run/mysqld/mysqld10.sock",
          "mysqli.default_socket": "mysqli.default_socket",
          "pdo_mysql.default_socket": "/run/mysqld/mysqld10.sock",
          "display_errors": "1",
          "error_reporting": "E_ALL",
          "log_errors": "true"
        }
      }
    ]
  }
}

```

```

        },
        "user": "WordPress",
        "group": "http"
    },
    "connect_timeout": 60,
    "read_timeout": 3600,
    "send_timeout": 60
}],
"portals": [{
    "service": "wordpress",
    "type": "alias",
    "name": "wordpress",
    "alias": "wordpress",
    "app": "SYNO.SDS.WordPress"
}],
"pkg_dir_prepare": [{
    "source": "/var/packages/WordPress/target/src/wordpress",
    "target": "wordpress",
    "mode": "0755",
    "user": "WordPress",
    "group": "http"
}]
}
}

```

Build and Create Package

Run the following command to build your source code into package.

```
/toolkit/pkgscripts-ng/PkgCreate.py -p avoton -c wordpress_sample
```

After the build process, you can check the result in `/toolkit/result_spk`.

Verify the Result

If the building process was successful, you will see that the .spk file has been placed under result_spk folder. To test the spk file, you can use manual install in Package Center to install your package.

WordPress Installation Note

- user need to create database manually first (by using phpmyadmin or something else)
- database address should be set to `localhost:/run/mysqld/mysqld10.sock` if you are using db root user
- if you see error pages from nginx, you might need to disable nginx error intercept manually:

1. find out the nginx conf of wordpress

```

root@nas:/etc/nginx/conf.d# grep -R 'wordpress' .
./service.6522c657-36cf-4165-84ab-f9e271a712eb.60d1dcff-5b7f-4908-8890-fcfc19b333c8.conf:location ^~ /wordpress/ {
./service.6522c657-36cf-4165-84ab-f9e271a712eb.60d1dcff-5b7f-4908-8890-fcfc19b333c8.conf:    location ^~ /wordpress
/ {
./www.webservice_portal_6522c657-36cf-4165-84ab-f9e271a712eb.conf:location = /wordpress {
./www.webservice_portal_6522c657-36cf-4165-84ab-f9e271a712eb.conf:location ~ ^/wordpress/ {

```

```

root@nas:/etc/nginx/conf.d# cat service.6522c657-36cf-4165-84ab-f9e271a712eb.60d1dcff-5b7f-4908-8890-fcfc19b333c8.conf
location ^~ /wordpress/ {
    include conf.d/.webstation.error_page.default.conf*;
    location ^~ /wordpress/ {
        proxy_connect_timeout 60s;
        proxy_read_timeout 3600s;
    }
}

```



```
proxy_send_timeout 60s;
proxy_pass http://localhost:914;
proxy_set_header X-Forwarded-By $server_addr;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-Proto $scheme;
proxy_set_header X-Forwarded-Port $server_port;
proxy_set_header Host $http_host;
proxy_set_header Upgrade $http_upgrade;
proxy_http_version 1.1;
proxy_intercept_errors on;
}
}
```

2. modify `proxy_intercept_errors` from on to off in wordpress nginx conf
3. run `systemctl reload nginx` then you can see original error page from wordpress now

Publish Synology Packages

Get Started with Publishing

To publish in Synology Package Center requires a few simple steps. Here is how to do it:

1. Apply on Synology website (<https://www.synology.com/en-global/support/developer#apply>).
2. Read and accept the Developer Distribution Agreement and Package Developer Guideline. Note that packages that you publish on Package Center must comply with the Terms of Service in Package Center.

Please note that the package quality directly influences the long-term success of your package in terms of installation, online reviews, engagement, and user retention.

Submitting the Package for Approval

Before you publish your package in Package Center and distribute it to users, you need to get the package (the SPK file) ready, make sure you have test it internally, and prepare your promotion materials if needed. Please see the below before submitting your package to us.

Confirm Package Behaviour

- It should meet our package review items. Please refer to [Package review](#).

Free or Paid Package

In Package Center, you can publish free or paid packages. Free packages can be downloaded by any user in Package Center. Paid apps can be downloaded only by users who have a registered Synology Account.

Deciding whether your package will be free or paid is important because **free packages must remain free**.

- Once your package is published as a free one, you cannot change it to a paid package.
- If you publish your package as a paid one, you can change it to free at any time (but cannot be changed back to paid).

Prepare Screenshots

When you publish in Package Center, you must supply a variety of high-quality screen-shots to showcase your package or brand. After you publish, they will appear on your package details page, or elsewhere. These screen-shots are a key part of a successful package details page that will attract and engage users. Therefore, you may also consider hiring a professional to produce them for you.

Submit Your Package

When you are ready to publish, go to Synology website (<https://www.synology.com/en-global/support/developer#apply>) to apply your package.

Make sure that:

- Your package is the right version.
- You provide a download link for your package.
- You provide a package description with what it does.
- You provide a change log with what was updated in this version.
- The link to your website and the support email address is correct.
- You have acknowledged that your package meets the Developer Distribution Agreement and also the [Terms of Service](#) from Package Center.

We will have a completed and rigorous internal process to make sure the quality of the published package. There are four major processes in short:

1. Receive your package and release note
2. Check the scripts of the package
3. Verify the functions of the package on different major versions of DSM and different models.(Checklist)
4. Release the package in Package Center. In the verification stage, we will ask you to provide a brief operation manual and test scenario for testing. If there are any issues, we will feedback to your teams and provide the related information. In order to expedite the verification, We strongly recommend your QC should verify the package before submitting it.

Responding to User Issues

After you publish a package, it is crucial for you to offer support to your customers. Prompt and courteous support can provide a better experience for users, which can result in more downloads and more positive online reviews for your packages. Users are more likely to be more engaged with your package and recommend it if you are responsive to their needs and feedback.

There are many ways that you can keep in touch with users and offer them support. The most common way is to provide a support email address in your package details page. You can also provide support in other ways, such as a forum or a mailing list. The Synology technical support team provides user support for downloading, installing and payments issues, but issues that fall outside of these topics will fall under your domain. Examples of issues you can support include: feature requests, questions about using the app and questions about compatibility settings.

After publishing, please plan to:

- Provide a link to your support resources and set up any other support outlets such as a forum.
- Provide an appropriate support email address on your package detail page and respond to users when they email you.
- Acknowledge and fix issues with your package. It helps to be transparent and list known issues on your package details page regularly.
- Publish updates frequently, without sacrificing quality or annoying users with too-frequent updates.
- With each update, make sure you provide a summary of what is new. Users will read it and appreciate that you are serious about improving the quality of your package.

Appendix A: Platform and Arch Value Mapping Table

The architecture of the NAS is developed upon various platforms on which your package is designed and needs to be addressed in the **INFO** file in the package.

In the below table, you will find the string value corresponding to the platform in question. For example, if the platform of your NAS is Marvell ARMADA 370, armada370, the value that should to be provided as a pair of the arch key is `armada370`.

Please check the platforms of the NAS to be supported and refer to the table below for their corresponding string values:

Arch Family	Member platforms
noarch	(all platforms)
x86_64	apollo lake, avoton, braswell, broadwell, broadwellnk, broadwellntb, broadwellntbap, bromolow, cedarview, coffeelake, denverton, geminilake, grantley, kvmx64, purley, skylaked, v1000
i686	evansport
armv7	alpine, alpine4k
armv5	628x
armv8	rtd1296, armada37xx

Supported platform value list:

- alpine
- alpine4k
- apollo lake
- armada370
- armada375
- armada37xx
- armada38x
- armadaxp
- avoton
- braswell
- broadwell
- broadwellnk
- broadwellntb
- broadwellntbap
- bromolow
- cedarview
- coffeelake
- comcerto2k
- denverton
- evansport
- geminilake
- grantley
- kvmx64
- monaco
- purley
- rtd1296
- rtd1619
- skylaked
- v1000

You can check the "Package Arch" field in the [CPU list](#) to find out which arch does your NAS belong to.

Compile Applications

The Synology NAS employs embedded SoC or x86-based CPUs, implementing several platforms -- such as ARM and x86 -- on a variety of Synology NAS models. In order to run 3rd-party applications on the Synology NAS, it is necessary to compile applications into an executable format for the corresponding platform.

This information will help you determine which DSM tool chain (please refer to the “Download DSM Tool Chain” section) to download for each model.

Please refer to [What kind of CPU does my NAS have](#) for a complete model list.

To compile an application for the Synology NAS, a compiler that runs on Linux PC is required in order to generate an executable file for the Synology NAS. This compiling procedure is called *cross-compiling*, and the set of compiling tools (compiler, linker, etc) used to compile the application is called a *tool chain*.

Download DSM Tool Chain

To download the DSM tool chain, please go to [Synology Archive](#).

You would need to know what your target platform is to download the corresponding tool chain. Here is the [platform list](#)

If you are not sure about which tool chain you need, please execute the following command on your Synology NAS.

```
DiskStation> uname -a
Linux DiskStation 4.4.59+ #24922 SMP PREEMPT Mon Aug 19 12:13:37 CST 2019 x86_64 GNU/Linux synology_apollolake_718+
DiskStation>
```

The output "synology_apollolake_718+" tells you which tool chain is appropriate. For example, apollolake means you need the tool chain for "Intel x86 Linux 4.4.59 (Apollolake)" on the [Synology Archive](#).

After you download the DSM tool chain, extract it to where you want it on your computer. For the following instructions we will extract to **/usr/local/** as an example. You can extract the tool chain by using the following command:

```
# tar xJf apollolake-gcc493_glibc220_linaro_x86_64-GPL.txz -C /usr/local/
```

Please make sure the tool chain is located in the directory **/usr/local** on your computer to ensure proper integration.

Compile

You can start compiling an application called `examplePkg.c`, for example, that looks like this:

```
#include <sys/sysinfo.h>

int main()
{
    struct sysinfo info;
    int ret;
    ret = sysinfo(&info);
    if (ret != 0) {
        printf("Failed to get system information.\n");
        return -1;
    }
    printf("Total RAM: %u\n", info.totalram);
    printf("Free RAM: %u\n", info.freeram);
    return 0;
}
```

To compile the application, run the following command:

```
/usr/local/arm-marvell-linux-gnueabi/bin/arm-marvell-linux-gnueabigcc examplePkg.c -o sysinfo
```

You can also write a Makefile for it:

```
EXEC= sysinfo
OBS= sysinfo.o

CC= /usr/local/arm-marvell-linux-gnueabi/bin/arm-marvell-linuxgnueabi-gcc
LD= /usr/local/arm-marvell-linux-gnueabi/bin/arm-marvell-linuxgnueabi-ld
CFLAGS += -I/usr/local/arm-marvell-linux-gnueabi/arm-marvell-linuxgnueabi/libc/include
LDFLAGS += -L/usr/local/arm-marvell-linux-gnueabi/arm-marvell-linuxgnueabi/libc/lib

all: $(EXEC)

$(EXEC): $(OBS)
    $(CC) $(CFLAGS) $(OBS) -o $@ $(LDFLAGS)

clean:
    rm -rf *.o $(PROG) *.core
```

Compile Open Source Projects

To compile an application on most open source projects, you will be asked to execute the following three steps:

1. `configure`
2. `make`
3. `make install`

The configure script basically consists of many lines which are used to check details about the machine on where the software is going to be installed. The script will check for a lot of dependencies on your system. When you run the configure script, you will see a lot of output on the screen, each being some sort of question with a respective yes/no reply. If there are any major requirements missing on your system, the configure script will exit and you will not be able to proceed with the installation until you meet all the requirements. In most cases, compile applications on some particular target machines will require you to modify the configure script manually to provide the correct values.

When running the configure script to configure software packages for cross-compiling, you will need to specify the `CC` , `LD` , `RANLIB` , `CFLAGS` , `LDFLAGS` , `host` , `target` , and `build` , etc. All these values can be found in `/env32.mak` or `/env64.mak` in your chroot environment. Some examples are given below.

For Intel X86-compatible platform in DSM 7.0:

```
env CC=/usr/local/x86_64-pc-linux-gnu/bin/x86_64-pc-linux-gnu-wrap-gcc \  
LD=/usr/local/x86_64-pc-linux-gnu/bin/x86_64-pc-linux-gnu-ld \  
RANLIB=/usr/local/x86_64-pc-linux-gnu/bin/x86_64-pc-linux-gnu-ranlib \  
CFLAGS="-DSYNOPLAT_F_X86_64 -O2 -include /usr/syno/include/platformconfig.h -DSYNO_ENVIRONMENT -DBUILD_ARCH=64 -D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64 -g -DSDK_VER_MIN_REQUIRED=600" \  
./configure \  
--host=i686-pc-linux-gnu \  
--target=i686-pc-linux-gnu \  
--build=i686-pc-linux \  
--prefix=/usr/local
```

Package Review

We are excited that you are creating packages for the Synology DSM and want to help you understand our guidelines so you can be confident your package will get through the review process quickly.

Review Item	Review Guideline
INFO: required field	Ensure required fields in INFO exists
INFO: deprecated field	Ensure deprecated fields in INFO does not exist (from DSM 7.0)
Lower privileged	The package should be run with non-privileged user (from DSM 7.0)
Package installation	The package should be installed successfully
Package start	The package should be started successfully
Package stop	The package should be stopped successfully
Package upgrade	The package should be upgraded successfully
Package uninstall	The package should be uninstalled successfully
Offline installation	The package should be able to be installed offline
Network activity during installation	There should not be any abnormal connection during installation
Security advisor scan	The package should not cause any security advisor issue
Antivirus essential scan	The package should pass the virus scanning
Clean up/file leftover	Files belong to package should be removed after uninstallation
Clean up/process leftover	Process belong to package should be stopped after uninstallation
Port-config	Register port numbers used by services of package
Port conflict	Registered port should not conflict with other services
Error log	There should not be any error log left on system
Apparmor log	There should not be any deny log from apparmor
Coredump file	There should not be any coredump file left on system
Ad-hoc test	Check any other abnormal behavior