Synology DiskStation Manager

3rd-Party Apps Developer Guide

Synology

2013-03-11

Synology Inc.

Synology Inc.

3F-3, No. 106, Chang-An W. Rd. Taipei 103, Taiwan

Table of Contents

Getting Started

Synology offers this developer guide to provide our users and system integrators with instructions regarding how to develop and install 3rd-party applications on the Synology DiskStation products, a line of network attached storage devices developed on the Linux kernel. With this guide, you can familiarize yourself with the following procedures:

- Compile programs to run on the Synology DiskStation.
- Integrate applications with the DiskStation's operation system -- Synology DiskStation Manager (DSM).
- Install application files to the recommended path in order to keep them intact when DSM is upgraded.
- Integrate applications with the Synology web authentication interface.
- Create a package file for manual or one-click installation in Synology's Package Center.

Overview

This document is written for Synology users and system integrators interested in adding their applications to the Synology DiskStation. You are advised to have some basic understanding of Linux programming before reading this document.

System Requirements

DSM 4.0 or later is required on Synology DiskStation.

Compile Applications

The Synology DiskStation employs embedded SoC or x86-based CPUs, implementing several platforms -- such as ARM and PowerPC -- on a variety of Synology DiskStation models. In order to run 3rd-party applications on the Synology DiskStation, it is necessary to compile applications into an executable format for the corresponding platform.

The table below lists the CPU, architecture, Endianness, and Linux kernel version of each Synology DiskStation model. This information will help you determine which DSM tool chain (please refer to the **"**Download DSM Tool Chain" section on page 7) to download for each model.

Please refer to **What kind of CPU does my NAS have** for a complete model list.

Model (To name a few)	CPU	Arch	Endianness	Linux
DS112j, DS112, DS112+, DS411slim, DS213, DS212j	Marvell 6281 Marvell 6282	ARM	Little Endian	2.6.32
DS213j	Marvell armada 370	ARM	Little Endian	3.2.30
N/A	Marvell armada xp	ARM	Little Endian	3.2.30
DS712+, DS2412+, RS2212+, DS1512+, DS1812+, DS412+, RS812+	Intel Atom	Intel x86	Little Endian	3.2.11
DS3612xs, RS3412xs, RS3412RPxs	Intel Core i3	Intel x86	Little Endian	3.2.11
N/A	Intel SoC	Intel x86	Little Endian	3.2.30
DS213+, DS413	Freescale QorIQ P1022	PowerPC	Big Endian	2.6.32
DS110+, DS210+, DS410	Freescale 8533 Freescale 8533 E	PowerPC	Big Endian	2.6.32

To compile an application for the Synology DiskStation, a compiler that runs on Linux PC is required in order to generate an executable file for the Synology DiskStation. 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 **http://sourceforge.net/projects/dsgpl/files**. The table below shows the filename of tool chains for DiskStation with different CPUs:

CPU	Tool Chain	Linux
Marvell 6281	**Marvell 88F628x Linux 2.6.32**	2.6.32
Marvell armada 370	**Marvell armada 370 Linux 3.2.30**	3.2.30
Marvell armada xp	**Marvell armada xp Linux 3.2.30**	3.2.30
Freescale 8533 Freescale 8533 E	**PowerPC 853x Linux 2.6.32**	2.6.32
Freescale QorIQ (P1022)	**PowerPC QorIQ Linux 2.6.32**	2.6.32
Intel Atom	**Intel x86 Linux 3.2.11 (Pineview)** **Intel x86 Linux 3.2.11 (Cedarview)**	3.2.11
Intel Core i3	**Intel x86 Linux 3.2.11 (Bromolow)**	3.2.11
Intel SoC	**Intel x86 Linux 3.2.30 (Evansport)**	3.2.30

If you're not sure about which tool chain you should use, please execute the following command on your NAS.

```
# uname -a

Linux myds 3.2.40 #3503 SMP Thu Mar 21 15:17:31 CST 2013 x86_64
GNU/Linux synology_x86_712+
```

The last "synology_x64_712+" tells you which tool chain is appropriate. For examples, x86 means you need tool chain for Pineview.

After you download the DSM tool chain, untar it into **/usr/local/** on your computer using the following command:

```
# tar zxpf gcc343_glibc232_88f5281.tgz -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 to compile an application. For example, the content of an application called "**sysinfo.c**" 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-gnu/bin/arm-marvell-linux-gnu-gcc
sysinfo.c –o sysinfo
```

You can also write a Make file for it:

```
EXEC= sysinfo
OBJS= sysinfo.o

CC=   /usr/local/arm-marvell-linux-gnu/bin/arm-marvell-linux-gnu-gcc
LD=   /usr/local/arm-marvell-linux-gnu/bin/arm-marvell-linux-gnu-ld
CFLAGS  += -I/usr/local/arm-marvell-linux-gnu/include
LDFLAGS+=-L/usr/local/arm-marvell-linux-gnu/lib

all: $(EXEC)

$(EXEC): $(OBJS)
```

```
	$(CC) $(CFLAGS) $(OBJS) -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 some details about the machine on which the software is going to be installed. This script checks for lots of dependencies on your system. When you run the configure script, you would see a lot of output on the screen, each being some sort of question and a respective yes/no as the reply. If any of the major requirements are missing on your system, the configure script will exit and you won't be able to proceed with the installation, until you get those required things. In most cases, to compile applications on some particular target machines requires you to modify the configure script manually so as 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, CFLAGS, host, target, and build, etc. Examples are given as below.

For PowerPC 8544/8533 platform:

```
# env CC=/usr/local/powerpc-linux-gnuspe/bin/powerpc-linux-gnuspe-gcc \

 LD=/usr/local/powerpc-linux-gnuspe/bin/powerpc-linux-gnuspe-ld \

 RANLIB=/usr/local/powerpc-linux-gnuspe/bin/powerpc-linux-gnuspe-
ranlib \

 CFLAGS="-I/usr/local/powerpc-linux-gnuspe/include -mcpu=8548 -mhard-
float -mfloat-gprs=double" \

 LDFLAGS="-L/usr/local/powerpc-linux-gnuspe/lib" \

 ./configure \

 --host=powerpc-unknown-linux \

 --target=powerpc-unknown-linux \

 --build=i686-pc-linux \

 --prefix=/usr/local
```

For PowerPC QorIQ platform:

```
# env CC=/usr/local/powerpc-none-linux-gnuspe/bin/powerpc-none-linux-gnuspe-gcc \
 LD=/usr/local/powerpc-none-linux-gnuspe/bin/powerpc-none-linux-gnuspe-ld \
 RANLIB=/usr/local/powerpc-none-linux-gnuspe/bin/powerpc-none-linux-gnuspe-ranlib \
 CFLAGS-I/usr/local/powerpc-none-linux-gnuspe/include -mcpu=8548 -mhard-float -mfloat-gprs=double" \
 LDFLAGS="-L/usr/local/powerpc-none-linux-gnuspe/lib" \
 ./configure \
 --host=powerpc-unknown-linux \
 --target=powerpc-unknown-linux \
 --build=i686-pc-linux \
 --prefix=/usr/local
```

For Marvell 6281 platform:

```
# env CC=/usr/local/arm-none-linux-gnueabi/bin/arm-none-linux-gnueabi-gcc \
 LD=/usr/local/arm-none-linux-gnueabi/bin/arm-none-linux-gnueabi-ld \
 RANLIB=/usr/local/arm-none-linux-gnueabi/bin/arm-none-linux-gnueabi-ranlib \
 CFLAGS="-I/usr/local/arm-none-linux-gnueabi/include" \
 LDFLAGS="-L/usr/local/arm-none-linux-gnueabi/lib" \
 ./configure \
 --host=armle-unknown-linux \
 --target=armle-unknown-linux \
 --build=i686-pc-linux \
 --prefix=/usr/local
```

For Marvell armada 370 platform:

```
# env CC= /usr/local/arm-marvell-linux-gnueabi/bin/arm-marvell-linux-
gnueabi-gcc \
 LD= /usr/local/arm-marvell-linux-gnueabi/bin/arm-marvell-linux-
gnueabi-ld \
 RANLIB= /usr/local/arm-marvell-linux-gnueabi/bin/arm-marvell-linux-
gnueabi-ranlib \
 CFLAGS=" -I/usr/local/arm-marvell-linux-gnueabi/arm-marvell-linux-
gnueabi/libc/include -mhard-float -mfpu=vfpv3-d16" \
 LDFLAGS=" -L/usr/local/arm-marvell-linux-gnueabi/arm-marvell-linux-
gnueabi/libc/lib" \
 ./configure \
 --host=armle-unknown-linux \
 --target=armle-unknown-linux \
 --build=i686-pc-linux" \
 --prefix=/usr/local
```

For Marvell armada xp platform:

```
# env CC= /usr/local/arm-marvell-linux-gnueabi/bin/arm-marvell-linux-
gnueabi-gcc \
 LD= /usr/local/arm-marvell-linux-gnueabi/bin/arm-marvell-linux-
gnueabi-ld \
 RANLIB= /usr/local/arm-marvell-linux-gnueabi/bin/arm-marvell-linux-
gnueabi-ranlib \
 CFLAGS=" -I/usr/local/arm-marvell-linux-gnueabi/arm-marvell-linux-
gnueabi/libc/include -mhard-float -mfpu=vfpv3-d16" \
 LDFLAGS=" -L/usr/local/arm-marvell-linux-gnueabi/arm-marvell-linux-
gnueabi/libc/lib" \
 ./configure \
 --host=armle-unknown-linux \
 --target=armle-unknown-linux \
 --build=i686-pc-linux" \
```

```
  --prefix=/usr/local
```

For Intel X86 platform:

```
# env CC=/usr/local/i686-linux-gnu/bin/i686-linux-gnu-gcc \
  LD=/usr/local/i686-linux-gnu/bin/i686-linux-gnu-ld \
  RANLIB=/usr/local/i686-linux-gnu/bin/i686-linux-gnu-ranlib \
  CFLAGS="-I/usr/local/i686-linux-gnu/include" \
  LDFLAGS="-L/usr/local/i686-linux-gnu/lib" \
  ./configure \
  --host=i686-linux-gnu \
  --target=i686-linux-gnu \
  --build=i686-pc-linux \
  --prefix=/usr/local
```

For Intel Atom Evansport platform:

```
# env CC=/usr/local/i686-pc-linux-gnu/bin/i686-pc-linux-gnu-gcc \
  LD=/usr/local/i686-pc-linux-gnu/bin/i686-pc-linux-gnu-ld \
  RANLIB=/usr/local/i686-pc-linux-gnu/bin/i686-pc-linux-gnu-ranlib \
  CFLAGS="-I/usr/local/i686-pc-linux-gnu/i686-pc-linux-gnu/sys-
root/usr/include" \
  LDFLAGS="-L/usr/local/i686-pc-linux-gnu/i686-pc-linux-gnu/sys-
root/lib" \
  ./configure \
  --host=i686-pc-linux-gnu \
  --target=i686-pc-linux-gnu \
  --build=i686-pc-linux \
  --prefix=/usr/local
```

Compile Kernel Modules

If you would like to compile kernel modules, you will need to obtain a Synology GPL to access the kernel source code. Please refer to **http://www.synology.com/enu/gpl/** for details.

In the kernel source code, there are different configuration files for different platforms. The configuration files are listed below:

CPU	Configuration File	Arch	Linux
Marvell 6281 Marvell 6282	synoconfigs/88f6281	ARM	2.6.32
Marvell armada 370	synoconfigs/armada370	ARM	3.2.30
Marvell armada xp	synoconfigs/armadaxp	ARM	3.2.30
Freescale 8533	synoconfigs/ppc8533	PowerPC	2.6.32
Freescale QorIQ (P1022)	synoconfigs/ppcQorIQ	PowerPC	2.6.32
Intel Atom D525, D510, D410, D425	synoconfigs/x86_64	x86	3.2.11
Intel Atom D2700	synoconfigs/cedarview	x86	3.2.11
Intel SoC CE5335	synoconfigs/evansport	x86	3.2.30
Intel Core i3	synoconfigs/bromolow	x86	3.2.11

Please copy the proper configuration file to "**.config**", and run "**make oldconfig**" and "**make menuconfig**" to select your kernel modules. Depending on the platform you would like to compile, you have to set the proper ARCH and CROSS_COMPILE into environment variables.

```
For example, to compile 2.6 kernel modules for 5281 platform:

# cd linux-2.6.15

# cp 88f5182-config .config

# make ARCH=arm \ CROSS_COMPILE=/usr/local/arm-marvell-linux-
gnu/bin/arm-marvell-linux-gnu- oldconfig

# make ARCH=arm \
CROSS_COMPILE=/usr/local/arm-marvell-linux-gnu/bin/arm-marvell-linux-
gnu- menuconfig

# make ARCH=arm \
CROSS_COMPILE=/usr/local/arm-marvell-linux-gnu/bin/arm-marvell-linux-
gnu- modules
```

Synology Package

Package Introduction

Synology Package Center in DSM automates the process of installing, upgrading, configuring, and uninstalling packages. In Synology Package Center, the developer defines some scripts and metadata to control the installation, uninstalling, and upgrading processes as well as how to communicate with DSM.

Package Structure

The Synology package is a SPK file in **tar** format, containing metadata and files as the following:

File/Folder Name	Description	File/Folder Type	DSM Requirement
INFO	This file contains the information displayed in Package Center or to control the flow of installation. (Please refer to "INFO" on page 16 for more information)	File	2.0-0731
WIZARD_UIFILES	Optional. This folder contains files in which descriptions of UI components are shown during the installation, uninstalling, and upgrading processes. (Please refer to "WIZARD_UIFILES" on page 25 for more information)	Folder (Contains install_uifile, upgrade, uifile, uninstall_uifile)	3.2-1922
package.tgz	This is a compressed file in .tgz format containing all the files that are required, such as executable binary, library, or UI files. (Please refer to "package.tgz" on page 29 for more information)	.tgz File	2.0-0731
scripts	This folder contains shell scripts which are executed during the installation, uninstalling, upgrading, start, and stop processes. (Please refer to "scripts" on page 29 for more information)	Folder (Contains preinst, postinst, preuninst, postunist, preupgrade, postupgrade, start-stop-status)	2.0-0731

File/Folder Name	Description	File/Folder Type	DSM Requirement
conf	Optional. The folder contains configures. Note: If you want to configure files within it, **support_conf_folder** key in **INFO** file must be set to "yes". It's not compatible with all DSM versions because the "conf" folder won't be installed in DSM 4.1 or older. Please install it yourself if your package can be installed in older DSM. (Please refer to "conf" on page 21 for more information)	Folder (contains PKG_DEPS, PKG_CONX)	4.2-3160
LICENSE	Optional. This file is shown in the installation process, and must be less than 1 MB.	File	3.2-1922
PACKAGE_ICON.PNG	72 x 72 .png image is shown in Package Center	.png file	3.2-1922
PACKAGE_ICON_120.PNG	120 x 120 .png image is shown in Package Center Note: It's not compatible with all DSM versions because the icon won't be installed in DSM 4.1 or older. If your package can be installed in DSM 4.1 or older, please refer next section to define **package_icon_120** in INFO file to instead of taking PACKAGE_ICON_120.PNG.	.png file	4.2-3160

Note:

- All words are case sensitive.
- You can use **tar -cvf packge.spk [files]** to create the package.

INFO

The "**INFO**" file is used to describe the information of the package. Package Center will search for information to control the flow of installation, upgrading, uninstalling, start, stop processes and listing in Package Center. For example, if you would like the installation package to be dependent on some services, you can define the key as **install_dep_services**; if you would like to restart some services after the installation, you can define the key as **instuninst_restart_services**.

Each piece of information in the **INFO** file is defined in key/value pairs separated by an equals sign, e.g. **key="value"**. The following are some keys configured in **INFO**:

Key	Description	Value	Default Value	DSM Requirement
package	Package name. No more than one version of a package can exist at the same time; therefore, the name is unique. If **displayname** key is empty, Package Center will show "**package='xxx'**" where xxx is the name of the package. For example, **package="Time Backup"**. Note: This key cannot contain any of these special characters :, /, >, < or =.	String	(Empty)	2.0-0731
version	Package version. End users can identify the package version, e.g. **version="7.2.1"**.	String	(Empty)	2.0-0731
description	Package Center shows a short description of the package, e.g. description = "Time Backup is an innovative solution that backs up DiskStation data in multiple versions. You could intuitively browse among versions and easily restore data to any specific time."	String	(Empty)	2.3-1118
support_url	Package Center shows a support link to allow users seek technical support when needed, e.g. support_url ="**https://MyDS.synology.com/support/support_form.php**".	String	(Empty)	4.2-3160
description_[DSM language]	Optional. Package Center shows a short description in the DSM language set by the end-user.	String	description	2.3-1118
displayname	Optional. Package Center shows the name of the package.	String	package name	2.3-1118
displayname _[DSM language]	Optional. Package Center shows the name in the DSM language set by the end-user.	String	package name	2.3-1118
maintainer	Package Center shows the developer of the package, e.g. **maintainer="Synology Inc."**	String	(Empty)	2.0-0731
maintainer_url	Optional. If a package has a "**maintainer**" webpage, Package Center will show a link to let user open it. e.g. **maintainer_url="http://www.synology.com"**	String	(Empty)	4.2-3160
distributor	Optional. Package Center shows the publisher of the package, e.g. **distributor="Synology Inc."**.	String	(Empty)	4.2-3160

Key	Description	Value	Default Value	DSM Requirement
distributor_url	Optional. If a package is installed and has a "help" webpage, Package Center will show a link to let user open it, e.g. **distributor_url ="**http://www.synology.com/enu/apps/3rd-party_application_integration.php**"**.	String	(Empty)	4.2-3160
arch	List the CPU architectures which can be used to install the package, e.g. **arch="noarch" or arch = "x86 ppc853x"**. Reference: http://forum.synology.com/wiki/index.php/What_kind_of_CPU_does_my_NAS_have	ppc853x, 88f6281, 88f6282, x86, cedarview, bromolow, qoriq, armada370, armadaxp, evansport, noarch (Separated with a space)	noarch	2.0-0731
model	Optional. List the models on which packages can be installed. It is composed of Synology string, architecture and model name.	String (Separated with a space, e.g. synology_88f6281_209, synology_cedarview_rs812rp+, synology_x86_411+II, synology_bromolow_3612xs, synology_cedarview_rs812rp+, …)	(Empty)	4.0-2219
checksum	Optional. Contain MD5 string to verify the package.tgz.	String	(Empty)	3.2-1922
adminport	Optional. A package listens on a specific port to display its own management UI. If the package is defined by a port, e.g. **adminport="9002"**, a link will be opened when the package is activated, e.g. **adminprotoco**l://ip:**adminport/adminurl**.	0~65536	80	2.0-0731
adminurl	Optional. If a package is installed and has an "**administration**" webpage, a link will be opened when the package is activated, e.g. **adminprotoco**l://ip:**adminport/adminurl**.	String	(Empty)	2.3-1118
adminprotocol	Optional. For example: **adminprotocol**://ip:**adminport/adminurl**	http / https (Separated with a space)	http	3.2-1922
firmware	Optional. Minimum version of DSM firmware that is required to run the package.	X.Y-Z DSM major number, DSM minor number, DSM build number	(Empty)	2.3-1118

Key	Description	Value	Default Value	DSM Requirement
dsmuidir	Optional. DSM UI folder name in package.tgz. The folder in **/volumeX/@appstore/[packge name] /**[**dsmuidir**] will be automatically linked to **/usr/syno/synoman/webman/3rdparty/[package name]** after the **start-stop-status** script with the start argument is run and the file mode bits is changed to **777**. To remove the link, run the **start-stop-status** script with the stop argument. Note: This key cannot contain **:** or **/.**	String	(Empty)	3.2-1922
checkport	Optional. 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).	"yes"/"no"	"yes"	3.2-1922
startable	Optional. 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 “startable” is set to “no”, the start-stop-status script is still required.	"yes"/"no"	"yes"	3.2-1922
helpurl	Optional. If a package is installed and has a "help" webpage, Package Center will show a link to let user open it, e.g. helpurl ="**http://www.synology.com/enu/apps/3rd-party_application_integration.php**".	String	(Empty)	3.2-1922
report_url	Optional. If a package is a beta version and has a "report" webpage, Package Center will show you the link. This package is considered the beta version, and the beta information will be also shown in Package Center.	String	(Empty)	3.2-1922
package_icon	72x72 png image data is encoded by Base64. Note: This value will be replaced when a **PACKAGE_ICON.PNG** file is stored in the **[package name].spk**. If the value is not defined and no **PACKAGE_ICON.PNG** file is in the **[package name].spk**, the package icon is a default one.		a default icon data	3.2-1922
package_icon_120	120x120 png image data is encoded by Base64. Note: This value will be replaced when a **PACKAGE_ICON_120.PNG** file is stored in the **[package name].spk**. If the value is not defined and no **PACKAGE_ICON_120.PNG** file is in the **[package name].spk**, 72x72 icon is a default one.		package_icon (72x72 png image)	4.2-3160

Key	Description	Value	Default Value	DSM Requirement
install_reboot	Optional. Reboot DS after installing or upgrading the package. Note: If the value is set to "yes", the value of **instuninst_restart_services** is ignored.	"yes"/"no"	"no"	3.2-1922
install_dep_packages	Optional. Before a package is installed or upgraded, these packages must be installed first. Besides, the order of start or stop packages is also depended on it. The format consists of a package name, e.g. **install_dep_packages="packageA"**. If more than one dependent packages are required, the package 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 is 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 pckage can be installed in DSM 4.1 or older because it can't be compared correctly. Instead, the package version should be set lower or higher.	Package names (Separated with a colon)	(Empty)	3.2-1922
install_conflict_packages	Optional. Before your package is installed or upgraded, these conflict packages can't be installed. The format consists of a package name, e.g. **install_conflict_packages="packageA"**. If more than one conflict package are required of the format, the package 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 is 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. Don't use **<=** and **>=** if a pckage can be installed in DSM 4.1 because it can't be compared correctly. Instead, the package version should be set lower or higher.	Package names (Separated with a colon)	(Empty)	4.1-2851

Key	Description	Value	Default Value	DSM Requirement
instuninst_restart_services	Optional. Restart services after installing, upgrading and uninstalling the package. Note: If the service is not enabled or started by the end-user, it cannot be restarted. If the **install_reboot** is set to “yes”, this value is ignored in the installation process.	apache-sys, apache-web, mdns, samba, db, applenetwork, cron, nfs, firewall (Separated with a space; see notes below)	(Empty)	3.2-1922
startstop_restart_services	Optional. Restart services after starting and stopping the package. Note: If the service is not enabled or started by the end-user, it cannot be restarted. If startable is set to “no”, the value is ignored.	apache-sys, apache-web, mdns , samba, db , applenetwork, cron, nfs, firewall (Separated with a space; see notes below)	(Empty)	3.2-1922
install_dep_services	Optional. Before the package is installed or upgraded, these services must be started or enabled by the end-user.	apache-web, mysql, php_disable_safe_exec_dir (Separated with a space)	(Empty)	3.2-1922
start_dep_services	Optional. 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.	apache-web, mysql, php_disable_safe_exec_dir (Separated with a space)	(Empty)	3.2-1922
dsmappname	Optional. 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.	(Separated with a space)	(Empty)	3.2-1922
extractsize	Optional. The value means the minimum space to install a package. It will be used to prompt the user if there is enough free space to install it.	Size number in bytes	The byte size of SPK file of package	4.0-2166
support_conf_folder	Optional. If you want to use to some special configure files within a "**conf**" folder, this value must be set to "**yes**". More details are given in the "**conf**" section.	"yes"/"no"	"no"	4.2-3160

Please note the following points:

- [DSM language]: 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).
- All words are case insensitive.

- The **apache-sys** is an apache daemon listening on DSM ports (e.g. 5000 or 5001), while **apache-web** is an apache daemon listening on Web Station ports (e.g. 80 or 443).
- Code words of value:
  - **mdns** = Multicast DNS Service Discovery
  - **db** = MySQL and PostgreSQL
  - **apple network** = Apple Network
  - **nfs** = NFS
- The version of DSM requirement means key/value pairs in **INFO** works correctly in the minimum version of DSM.

conf

The "**conf**" folder contains special configurations including some information which cannot be described in **INFO** file with key/pair format. Package Center will control the flow of installation, upgrading, uninstalling, start, stop processes according to these configurations.

In DSM 4.2, there are two configurations, **PKG_DEPS** and **PKG_CONX**, stored within this folder that are used to define dependency or conflict between packages. However, dependency or conflict will be checked according to the end-user's DSM version. For example, Perl is built in DSM 4.1, but does not exist in DSM 4.2. Therefore, if your package depends on Perl, the Perl package must be installed in DSM 4.2 before your package can be installed. You can set **PKG_DEPS** configuration to specify that the dependency rule only works on DSM 4.2 or newer.

Dependency or conflict is similar to **install_dep_packages** and **install_conflict_packages** keys in **INFO** file, but they cannot define the restriction according to specific DSM versions. In addition, if you define dependence in **PKG_DEPS** file, then the **install_dep_packages** key in the **INFO** file will be ignored in DSM 4.2 or newer. If you define conflict in the **PKG_CONX** file, then the **install_conflict_packages** key in **INFO** file will be ignored in DSM 4.2 or newer.

Finally, if you want to set some special configurations in files stored within it, the **support_conf_folder** key in **INFO** file must be set to "yes." Otherwise, because the **conf** folder will not be installed in DSM 4.1 or older, if your package can be installed in older DSM, please manually install it to **/var/packages/[package name]/conf** in package scripts, for the purpose of making them work after DSM is upgraded to 4.2 or newer.

The **conf** folder contains files as follows:

File/Folder Name	Description	File/Folder Type	DSM Requirement
PKG_DEPS	Define dependency between packages with restrictions on DSM firmware. Before your package is installed or upgraded, these packages must be installed first. Package Center controls the order of start or stop packages according to the dependency.	File	4.2-3160
PKG_CONX	Define conflicts between packages with restrictions on DSM firmware. Before your package is installed or upgraded, these conflicting packages can't be installed.	File	4.2-3160

Note:

- All words are case sensitive.

Each configure file is defined in standard .ini file format in key/value pairs with sessions, for example:

```
[session]
```

A session describes a unique name of dependent/conflict package. Each session contains some information about the requirement of package versions and the restriction of DSM versions.

Keys configured in **PKG_DEPS** file each dependent package (session) contains:

Key	Description	Value
pkg_min_ver	Minimum version of dependent package. End user must install this dependent package with the version or newer before installing your package.	Package version
pkg_max_ver	Maximum version of dependent package. End user must install this dependent package with the version or older before installing your package.	Package version
dsm_min_ver	Required minimum version of DSM. If end user has the version or newer of DSM, this dependency will be considered, but it will be ignored in older DSM.	X.Y-Z DSM major number, DSM minor number, DSM build number
dsm_max_ver	Required maximum version of DSM. If end user has the version or older of DSM, this dependency will be considered, but it will be ignored in newer DSM.	X.Y-Z DSM major number, DSM minor number, DSM build number

Example:

```
# Your package depends on Package A in any version

[Package A]



# Your package depends on Package B version 2 or newer

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 older but it will
be ignored when DSM version is older 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 newer than 4.1-2668
[Package E]
dsm_max_ver =4.1-2668
pkg_min_ver=2
```

Keys configured in **PKG_CONX** file each conflicting package (session) contain:

Key	Description	Value
pkg_min_ver	Minimum version of conflicting package. If end user installs this conflicting package with the specified version or newer, he will not be able to install your package.	Package Version
pkg_max_ver	Maximum version of conflicting package. If end user installs this conflicting package with the specified version or older, he will not be able to install your package.	Package Version
dsm_min_ver	Required minimum version of DSM. If end user has the specified version or newer of DSM, this conflict will be considered, but it will be ignored in older versions of DSM.	X.Y-Z DSM major number, DSM minor number, DSM build number
dsm_max_ver	Required maximum version of DSM. If the end user has the specified version or older of DSM, this conflict will be considered, but it will be ignored in newer DSM.	X.Y-Z DSM major number, DSM minor number, DSM build number

Example:

```
# 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 older, but it will
be ignored when DSM version is older 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 newer than 4.1-2668

[Package E]

dsm_max_ver =4.1-2668

pkg_min_ver=2
```

WIZARD_UIFILES

install_uifile, upgrade_uifile, and **uninstall_uifile** are the files which describe UI components in JSON format, and are stored in the "**WIZARD_UIFILES**" folder. During the installation, upgrading, and uninstalling processes, these UI components will appear in the wizard. Once these components are selected, their keys will be set in the script environment variables, and their values are "true," "false," or text values.

These files can be regarded as user settings or used to control the flow of script execution.

- **install_uifile**: Describes some UI components for the installation process. During the running of the **preinst** and **postinst** scripts, these component keys and values can be found in the environment variables.
- **upgrade_uifile**: Describes some UI components for the upgrading process. During the running of the **preupgrade**, **postupgrade**, **preinst** and **postinst** scripts, these component keys and values can be found in the environment variables.
- **uninstall_uifile**: Describe some UI components for the uninstalling process. During the running of the **preuninstall** and **postuninstall** scripts, these component keys and values can be found in the environment variables.

Note:

If you would like to localize the descriptions of UI components, you can add a language abbreviation suffix to the file "**install_uifile_[DSM language],**" "**upgrade_uifile_[DSM language]**" or "**uninstall_uifile_[DSM language]**" in this folder. For example, in order to perform installation in Traditional Chinese, **[DSM language]** should be replaced with "**cht**" like "**install_uifile_cht**".

Example of the file in JSON format:

```
[{
    "step_title": "Step1",
    "items": [{
        "type": "singleselect",
        "desc": "a radio group",
        "subitems": [{
            "key": "radio1",
            "desc": "Radio button 1",
            "defaultVaule": false
```

```
        }, {
            "key": "radio2",
            "desc": "Radio button 2",
            "defaultVaule": true
        }]
    }]
}, {
    "step_title": "Step2",
    "items": [{
        "type": "multiselect",
        "desc": "a check group",
        "subitems": [{
            "key": "check1",
            "desc": "Check button 1"
        }, {
            "key": "check2",
            "desc": "Check button 2",
            "defaultVaule": true,
            "validator": {
                "fn": "{var v=arguments[0]; if (!v) return 'Check
this';return true;}"
             }
        }]
    }, {
        "type": "textfield",
        "desc": "textfield",
        "subitems": [{
            "key": "textfield1",
```

```
            "desc": "textfield 1",
            "defaultVaule": "default",
            "validator": {
                "allowBlank": false,
                "minLength": 2,
                "maxLength": 10
            }
        },{
            "key": "textfield2",
            "desc": "textfield 2",
            "emptyText": "abc1@cde.com",
            "validator": {
                "vtype": "email",
                "regex": {
                    "expr": "/[0-9]/i",
                    "errorText": "Error"
                }
            }
        }]
    }]
}]
```

Here are the details of file content in JSON format:

Property	Description	DSM Requirement
step_title	Optional. Describe the title of the step currently performed in wizard.	3.2-1922
items	Describe an array containing the components of “**singleselect**”, “**multiselect**”, “**textfield**”, or “**password**” type.	3.2-1922

Property	Description	DSM Requirement
type	Must be "**singleselect**", "**multiselect**", "**textfield**" or "**password**". "**singleselect**" type represents the components in the **subitems** which are all radio buttons. End-users can select only one radio box with a unique key. "**multiselect**" type represents the components in the **subitems** which are all checkboxes. End-users can check more than one checkboxes. "**textfield**" type represents the components in the **subitems** which are all text fields. End-users can type texts. "**password**" type represents the components in the **subitems** which are all password fields. End-users can type password.	3.2-1922
desc	Optional. Describe a component in the label text.	3.2-1922
subitems	Describe an array containing radio buttons, checkboxes, text fields or password components.	3.2-1922
key	A unique component key value represents a UI component. If a component is selected by the end-user, this key will be set in the script environment variables (the string value of the selected checkbox or radio button is always "**true**".).	3.2-1922
defaultVaule	Optional. true/false value to initialize "**singleselect**" or "**multiselect**" component, or a string value to initialize "**textfield**" or "**password**" component.	4.2-3160
emptyText	Optional. The prompt text to place into an empty "**textfield**" or "**password**" component to prompt the user how to fill out it if **defaultVaule** isn't set.	4.2-3160
validator	JSON-style object to describe validation functions. If it's failed to valudate it by these functions, the user can't go to the next step in the wizard. More detailed properties of **validator** are given in the next table.	4.2-3160

Note:

1. All words are case sensitive.
2. In DSM 4.0 or above, if both the **type** and **subitems** properties are empty, texts in the **desc** property will be displayed as one of the steps of wizard.

Here are the properties of **validator**:

Property	Description	Value
allowBlank	Specify false to validate that the value's length of "**textfield**" or "**password**" component is > 0	true/false
minLength	Minimum length of "**textfield**" or "**password**" component	Number
maxLength	Maximum length of "**textfield**" or "**password**" component	Number

Property	Description	Value
vtype	Specify predefined validation function, "**alpha**": validate alpha value "**alphanum**": validate alphanumeric value "**email**": validate email address "**url**": validate URL	"alpha", "alphanum", "email", "url"
regex	Describe validation function in regular expression and invalid message. Properties contains: "**expr**": Javascript Regular Expression "**errorText**": invalid string	JSON-style object
fn	Describe Javascript function which is encoded by JSON-style string with curly brackets. In this function, you can use arguments[0] to get the value of the component. In addition, this function must return true if the value is valid or an invalid string if the value is invalid.	String

Note:

- All words are case sensitive.

package.tgz

This is a compressed file containing all files such as executable files, library, and UI files, and will be saved in the directory "**@appstore**" in the installed volume once uncompressed. A package can be created with the command "**tar czf package.tgz [files]**".

scripts

This folder contains shell scripts which are executed during the installation, uninstalling, upgrading, start, and stop packages. There are seven script files stored in the "**scripts**" folder.

1 **start-stop-status**: This script is used to start and stop a package, detect running status, and generate the log file. Parameters used by the script are listed below:

   a **start**: When the user clicks the button "**Run**" to run the package, after the package is installed, or when the DS is turned on, the Package Center program will call this script with "**start**" parameter, and a returned value will be acquired along the way.

   For nonzero returned values, you can compile error messages in the **SYNOPKG_TEMP_LOGFILE** file to prompt the user, ex:echo "Start failed" > $SYNOPKG TEMP LOGFILE. You can also write messages in the **SYNOPKG_TEMP_LOGFILE** file for zero returned values which represent that the process was successful.

   b **stop**: When the user clicks the button "**Stop**" to stop the running package, before the package is uninstalled, or when the DS is turned off, the Package Center program will call this script with "**stop**" parameter, and a returned value will be acquired along the way.

   For nonzero returned values, you can compile error messages in the **SYNOPKG_TEMP_LOGFILE** file to prompt the user, ex: echo "Stop failed" > $SYNOPKG TEMP LOGFILE. You can also write messages in the **SYNOPKG_TEMP_LOGFILE** file for zero returned values which represent that the process was successful.

   c **status**: When Package Center AP is opened to check package status, the Center will send a request to ask the status of the package with this parameter. It should return the following exit status codes:

   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. Please note, broken status (150) is only supported by DSM 4.2 and later.

d **log**: When a log page is opened in Package Center, the Center will send a request to ask the log of the package with this parameter. When the log filename is sent to STDOUT, the content of the log file will be displayed.

2 **preinst**: This script is run before the package files are transferred to **@appstore**. You can check if the installation requirements meet the DSM or package version, or if some services are enabled in this script.

For nonzero returned values, you can compile error messages in the **SYNOPKG_TEMP_LOGFILE** file to prompt the user, ex: echo "Hello!!" > $SYNOPKG_TEMP_LOGFILE. You can also write messages in the **SYNOPKG_TEMP_LOGFILE** file for zero returned values which represent that the process was successful.

3 **postinst**: This script is run after the package files are transferred to **@appstore**. You can change the file permission and ownership in this script.

For nonzero returned values, you can compile error messages in the **SYNOPKG_TEMP_LOGFILE** file to prompt the user, ex: echo "Hello!!" > $SYNOPKG_TEMP_LOGFILE. You can also write messages in the **SYNOPKG_TEMP_LOGFILE** file for zero returned values which represent that the process was successful.

4 **preuninst**: This script is run before the package is removed.

For nonzero returned values, you can compile error messages in the **SYNOPKG_TEMP_LOGFILE** file to prompt the user, ex: echo "Hello!!" > $SYNOPKG_TEMP_LOGFILE. You can also write messages in the **SYNOPKG_TEMP_LOGFILE** file for zero returned values which represent that the process was successful.

5 **postuninst**: This script is run after the package is removed from the system.

For nonzero returned values, you can compile error messages in the **SYNOPKG_TEMP_LOGFILE** file to prompt the user, ex: echo "Hello!!" > $SYNOPKG_TEMP_LOGFILE. You can also write messages in the **SYNOPKG_TEMP_LOGFILE** file for zero returned values which represent that the process was successful.

6 **preupgrade**: When you upgrade a package, the Package Center program calls this script before uninstalling the old one.

For nonzero returned values, you can compile error messages in the **SYNOPKG_TEMP_LOGFILE** file to prompt the user, ex: echo "Hello!!" > $SYNOPKG_TEMP_LOGFILE. You can also write messages in the **SYNOPKG_TEMP_LOGFILE** file for zero returned values which represent that the process was successful.

7 **postupgrade**: When you upgrade a package, the Package Center program calls this script after installing the new one.

For nonzero returned values, you can compile error messages in the **SYNOPKG_TEMP_LOGFILE** file to prompt the user, ex: echo "Hello!!" > $SYNOPKG_TEMP_LOGFILE. You can also write messages in the **SYNOPKG_TEMP_LOGFILE** file for zero returned values which represent that the process was successful.

During the process of installation, uninstalling, or upgrading a package, the sequence of execution of shell scripts are described as the followings,

To install a package:

- preinst
- postinst
- start-stop-status with start argument if end user choose to start it immediately

To upgrade a package:

- start-stop-status with stop argument if it has been started
- preupgrade
- preuninst
- postuninst
- preinst
- postinst
- postupgrade
- start-stop-status with start argument if it was started before being upgraded

To uninstall a package:

- start-stop-status with stop argument if it has been started
- preuninst

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 name which is defined in **INFO**.
- **SYNOPKG_PKGVER**: Package version which is defined in **INFO**.
- **SYNOPKG_PKGDEST**: Target directory in which the package is stored.
- **SYNOPKG_PKGDEST_VOL**: Target volume in which the package is stored. Please note, **SYNOPKG_PKGDEST_VOL** is only available in DSM 4.2 or above. If you want to get the target volume in older DSM, please parse it from **SYNOPKG_PKGDEST** variable.
- **SYNOPKG_PKGPORT**: Administrator port which is defined in INFO. Packages listed on a specific port to use the management UI.
- **SYNOPKG_PKGINST_TEMP_DIR**: Packages are extracted to a temporary directory whose path is described by this variable.
- **SYNOPKG_TEMP_LOGFILE**: Package Center randomly generates a filename for a script to log the information or error messages.
- **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. Reference: **http://forum.synology.com/wiki/index.php/What_kind_of_CPU_does_my_NAS_have**
- **SYNOPKG_PKG_STATUS**: Package status can be represented by these values: INSTALL, UPGRADE, UNINSTALL, START, and STOP.
    a Status value of a package will be set to INSTALL in the **preinst** and **postinst** scripts while the package is being installed. If the user chooses the "start after installation" option 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.

b Status value of a package will be set to UPGRADE in the **preupgrade, preuninst**, **postunist**, **preinst, postinst** and **postupgrade** scripts sequentially while the package is being upgraded. If the package has been already started before upgrade, the value will be set to UPGRADE in the **start-stop-status** script when the package is started or stopped.

c Status value of a package will be set to UNINSTALL in the **preuninst** and **postunist** scripts while the package is being uninstalled. If the package has been already started before uninstall, the value will be set to UNINSTALL in the **start-stop-status** script when the package is stopped.

d If the user starts or stops a package in Package Center, the status value of the package will be set to START or STOP in the **start-stop-status** script.

e When the DiskStation is booting or shutting down, its status value will be empty. Please note, **SYNOPKG_PKG_STATUS** is only available for the **start-stop-status** script in DSM 4.0 or above.

- **SYNOPKG_OLD_PKGVER**: Existing package version which is defined in INFO (only in **preupgrade** script).

Once the end-user enters or selects some values of the UI components which are configured in **install_uifile/upgrade_uifile/uninstall_uifile** (please refer to sections “conf” through “package.tgz” on pages 21-29), the names and values of the components will be set in the environment variables, but please note that the names of these components cannot be the same as those of the environment variables.

How to Use Package Toolkit

Package Toolkit

This toolkit consists of the following two parts: pre-built environment and front-end scripts. For faster development, we prepare a build environment for package developers. This environment already contains some pre-built projects, whose executable binaries or shared libraries are built on DSM, like **zlib**, **libxml2**, and so on. Therefore, necessary build-time libraries (.a, .so) and headers (.h, .hpp) are ready for use. Developers don't have to build them themselves. We also provide front-end scripts in a folder named "**pkgscripts**" to make environment deployment, package compilation and creation of final package SPK file easier.

Basic Requirements: All you need is a modern Linux PC with bash (>= 4.1.5), Python (>= 2.6.5) and apt-get.

Set Up Environment

First, you have to download our front-end scripts from **here** and use following command to extract the downloaded **pkgscripts.tgz** to the locations of your choice. We use **/toolkit** for example in this document from now on.

```
tar zxf pkgscripts.tgz -C /toolkit
```

Next, you can download and set up pre-built environments by using **EnvDeploy** as follows. Use -v to specify DSM version and -p to specify desired platform. If -p is not given, all available platforms for given version will be set up.

```
cd /toolkit/pkgscripts

./EnvDeploy -v 4.2 -p x64
```

Finally, the whole working directory will look like this, and **ds.${platform}-${version}** is the chroot environment to build your own projects. As we mentioned earlier, this toolkit contains some pre-built libraries and headers and you can find them under **usr/syno/**. Config files like **xml2-config** are under **usr/syno/bin/** and files for **pkgconfig** are under **usr/syno/lib/pkgconfig/**.

```
/toolkit/

 pkgscripts/

 build_env/

      ds.${platform}-${version}/

           usr/syno/{bin,lib,include}
```

Available Platforms

This table summarizes available platforms for DSM 4.2 version.

Platform	DSM Version 4.2
6281	Yes
armada370	Yes
armadaxp	Yes
824x	Yes
853x	Yes
bromolow	Yes
cedarview	Yes
ppc	Yes
qoriq	Yes
x64	Yes
evansport	Yes

Or you can use one of following commands to show available platforms. If -v is not given, available platforms for all versions are listed.

```
./EnvDeploy -v 4.2 --list

./EnvDeploy -v 4.2 --info platform
```

List Pre-built Projects

You can use this command to list pre-built projects for specified versions. If -v is not given, pre-built projects for all versions are listed.

```
./EnvDeploy -v 4.2 --info projects
```

Update Environment

Use EnvDeploy again to update your environments. For example, update x64 for DSM 4.2 by following command.

```
./EnvDeploy -v 4.2 -p x64
```

Prepare Source Code

Your source code must be put in a folder (we call it a “**project**”) under **/toolkit/source**. Besides, we need some metadata about your project to tell us how to build it, its dependency, directory structure of the SPK, and so on. These metadata are put in a folder called “**SynoBuildConf**” under your project, which we will discuss later. Now, the whole working directory looks like this.

```
/toolkit/
```

```
pkgscripts/

build_env/

     ds.${platform}-${version}

          usr/syno/{bin,lib,include}

source/

     ${your_project}/SynoBuildConf/{build,install,depends}
```

You can organize your source code in any structure you like, the only important item is to edit your SynoBuildConf correctly. Therefore, the following section will explain it in detail.

Create a Package - SynoBuildConf

build, **install** and **depends** in SynoBuildConf folder are used to control the creation of a package. The first two files must be written in bash and indicate how to build and install (create final package SPK file) your projects. The last one specifies dependency of your project and build environment for it. It's a text file consisting of several sections. When you're writing your own SynoBuildConf/build SynoBuildConf/install, please remember that both of them are executed in chroot environment.

Build Your Project - SynoBuildConf/build

This file must be written in bash and tells front-end scripts how to build your project. Current working directory is **/source/${project}** in chroot environment. You can utilize pre-defined variables in **/env.mak** in your Makefile. For example:

```
# SynoBuildConf/build


case ${MakeClean} in

       [Yy][Ee][Ss])

               make distclean

               ;;

esac

make ${MAKE_FLAGS}
```

All pre-built binaries, headers and libaries are under /**usr/syno** in chroot environment. You can find headers (.h/.hpp) in /**usr/syno/include**, shared libraries (.so/.a) in /**usr/syno/lib**, config scripts in **usr/syno/bin** and .pc files for **pkg-config** in /**usr/syno/lib/pkgconfig**. You can utilize them when writing your own **build** and **install** scripts in **SynoBuildConf**.

Install Your Project - SynoBuildConf/install

This file must be written in bash and tells front-end script how to install your project. The current working directory is **/source/${project}** in chroot environment. If it's the top project of your package, this file also defines how to create final package SPK file including directory structure and a file named **INFO** in SPK. (Please refer to INFO chapter)

For example, define SynoBuildConf/install for install your project

```
 # SynoBuildConf/install

make INSTALLDIR=/tmp/_install install
```

Utility script **pkgscripts/include/pkg_util.sh** can help create final package SPK file in correct format in an easier way. For example, you can use **pkg_dump_info** to create **INFO**, **pkg_make_spk** to create final package SPK file and so on. Following are some functions in **pkg_util.sh** that may be useful for you.

- **pkg_make_package $1 $2**: Create $2/packages.tgz from files in $1.
- **pkg_make_spk $1 $2**: Create $2/spk from files in $1.
- **pkg_dump_info**: Dump INFO according to given variables.
- **pkg_get_spk_platform**: Return platform for "arch" in **INFO**.

You can use them when creating **INFO** and in **SynoBuildConf/install**. For example:

Define INFO.sh script for generating INFO file

```
# INFO.sh



. /pkgscripts/include/pkg_util.sh

package="package_name "

version="1.2-3456 "

displayname="Package Name "

arch="$(pkg_get_platform) "

[ "$(caller)" != "0 NULL" ] && return 0

pkg_dump_info
```

Define SynoBuildConf/install for installing and packing your package

```
# SynoBuildConf/install



. /pkgscripts/include/pkg_util.sh



./INFO.sh > INFO  # create INFO
```

```
# prepare directory structure for your package.tgz
make INSTALLDIR=/tmp/_install install
# prepare directory structure for your package metadata and files,
including INFO, scripts, PACKAGE_ICON.PNG
make PACKAGEDIR=/tmp/_pkg package

pkg_make_package /tmp/_install /tmp/_pkg  # create
/tmp/_pkg/package.tgz
pkg_make_spk /tmp/_pkg /image/packages  # create /image/packages/*.spk
```

Project Dependency and Build Environment - SynoBuildConf/depends

This file specifies your project dependency and build environment of your project. For example:

```
# SynoBuildConf/depends

[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]
816x="4.3"  # build environment for x64 should be DSM 4.3
all="4.2"   # build environment for other platforms should be DSM 4.2
```

The last section "**default**" is necessary. It indicates each platform to build against some DSM version and the key "**all**" means default DSM version. More details about this file will be described later.

You can use following commands to see whether the dependency order of your projects is correct. Option -x0 means to traverse all dependent projects of ${project}.

```
cd /toolkit/pkgscripts

./ProjDepends.py -x0 ${project}
```

If your applications contains more than one projects, put them in **/toolkit/source** and edit **SynoBuildConf** properly for each of them.

Environment Variables in Build and Install Script

Front-end scripts will pass some environment variables to **SynoBuildConf/build** and **SynoBuildConf/install**, you can utilize them to build and install your projects. You can find most of them in **/toolkit/build_env/ds.${platform}-${version}/{env.mak,env.mak.template}**. The following lists some of these environment variables which might be important to you.

- **CC**: path of gcc cross compiler.
- **CXX**: path of g++ cross compiler.
- **CFLAGS**: global cflags includes -I/usr/syno/include.
- **LDFLAGS**: global ldflags includes -L/usr/syno/lib.
- **CoinfigOpt**: 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).
- **MAKE_FLAGS**: parameters to “make” command, for example “-j4”.

Build and Install Package

How to Build and Install Package

First of all, we build/install source codes under chroot environment. Therefore, you must run all commands with root permission or use sudo.

```
cd /toolkit

pkgscripts/PkgCreate.py -x0 ${project}
```

Once items in previous sections are ready, you can build your project using this command. Option -x0 means to traverse and build all dependent projects in correct order. Each projects is built according to their own SynoBuildConf/build.

It works like this. First, related projects under **/toolkit/source** will be hard-linked to **/toolkit/build_env/ds.${platform}/source**. Then their **SynoBuildConf/build** will be executed in order according their dependency. Again, **SynoBuildConf/build** is executed in chroot environment **/toolkit/build_env/ds.${platform}**.

PkgCreate.py has many other options to control the build flow, please use -h or --help to see its usage.

If you want to install projects without building it, just run **PkgCreate.py** with -i option. For example, the following command will execute **${project}/SynoBuildConf/install** and put the final package SPK file (if any) to **/toolkit/result_spk**.

```
cd /toolkit

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

Full Process to Create a Package SPK File

```
cd /toolkit

pkgscripts/PkgCreate.py -x0 -c ${project}
```

Option -c means to build and install (create a package SPK file) your project and the final package SPK file will be under **/toolkit/result_spk**. Creation of SPK is indicated by **${project}/SynoBuildConf/install**. Finally, different packages are put in different folders like the following.

```
/toolkit/

  pkgscripts/
```

```
build_env/
    ds.${platform}-${version}/
        usr/syno/
source/
    ${project}/
        SynoBuildConf/{build,install,depends}
result_spk/
    ${package}-${version}/*.spk
```

Variables **${package}** and **${version}** depends on “**package**” and “**version**” in **/toolkit/build_env/ds.${platform}-${version}/source/${project}/INFO** which must be created after **SynoBuildConf/install** is executed.

Advanced

The sections below illustrate advanced types of usage for this toolkit. Please continue reading for more details.

Platform-Specific Dependency

Platform-specific dependency means you can have different dependent projects for different platforms by appending "**:${platform}**" to following sections: **BuildDependent**, **ReferenceOnly**. The following example means only on 816x and aramda370 will your projects on libbar-1.0.

```
# SynoBuildConf/depends

[BuildDependent]
libfoo-1.0

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

[default]
all="4.2"
```

Collect SPK File in Your Way

By default, **PkgCreate.py** will move 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 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).

Current working directory of **SynoBuildConf/collect** is **/source/${project}** in chroot environment.

Integrate Your Package into DSM

Manage Storage for Application Files

When you build an application, you can create a directory named **bin** in it for utilities, **sbin** for daemons and system utilities, **etc** for configuration files, and **lib** for your libraries. These directories will be compressed to **package.tgz** during the last phase of application building. When a package is being installed, **package.tgz** will be extracted to **/var/packages/[package name]/target**, which is a symbolic link pointing to a folder in a data volume selected by the end user. **/var/packages/[package name]/target** is also available at the **SYNOPKG_PKGDEST** environment variable, one of the seven files in the script folder. See “scripts” on page 29 for more information.

Despite the fact that the directory **/var/packages** or **/usr/local** is reserved for 3rd-party applications, the storage space of system volumes is limited. If the size of files to be installed exceeds the capacity of the system volume, storage space will run out. Consequently, it is recommended that you directly read or write application files in **/var/packages/[package name]/target** or another space of the data volume. You can also make a symbolic link in **/usr/local** point to **/var/packages/[package name]/target** or another space when running the **postinst** script. It makes the path easier to be accessed in a library or a daemon. Please note that you may need to specify the correct prefix when running a configure script, so that the application can find the correct path information upon execution.

Synology releases DSM updates on a regular basis. Considering that application files might be affected during update procedure, it is important you have your application installed in the correct directory to prevent them from being deleted whenever DSM is upgrading in system partition.

When DSM is being upgraded, the directory **/var/packages/[package name]** and **/usr/local** will be backed up and restored automatically. However, some library files or built-in software might be modified during the upgrade procedure, so if your application depends on the files which are subject to change, it may not work afterward. In this case, you should check the status of these files or re-link them in the **start-stop-status** script to repair them if necessary. Alternatively, you can install them directly to **/var/packages/[package name]/target**.

Integrate Your Package into DSM Web GUI

In Synology DSM 3.0 or 4.0, integrating 3rd-party applications into your DiskStation is easy. When an application is integrated, its icon will automatically appear in the Main Menu of DSM. Users can also use customized icons for these applications. To place the icon on the desktop, you only have to drag it from the Main Menu to the desktop on DSM.

Startup

To integrate an application into Synology DiskStation Manager 3.0 or later, please follow the steps below:

1. Create a directory under the directory **/usr/syno/synoman/webman/3rdparty/**.
2. Create a text file named "**config**" under the directory.
3. Under the same directory, you can create a sub-directory named after your application, such as **/usr/syno/synoman/webman/3rdpaty/[package name]/**. You can put all UI related components, such as images, CSS, and CGI in the directory.

Next we will discuss the details of UI configuration.

Config

The text file "**config**" is used to configure UI behavior. The content of **config** should be in **JSON** format. For example:

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

```
    }

}
```

Details of the content of **application.cfg** are as follows:

Property	Description
com.company.App1 com.company.App2	In "**.url**", each object should have a unique property name.
title (Required)	"**title**" represents the application name that will be displayed in the main menu.
desc	"**desc**" describes more details about this application upon mouse-over.
icon (Required)	"**icon**" describes the icon for the application. It is a template string. The" {0}" can be replaced by "16", "24", "32", "48", depending on different pixels of the image file for the icon. The icon must be put under **/usr/syno/synoman/webman/3rdparty/xxx/** where xxx is the directory name of your application. For example, if you create a directory named "images" and put the icon image file "**icon.png**" in it, the full path for the icon will be: /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 And the icon value should be set as "**images/icon_{0}.png**".
type (Required)	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.
url (Required)	The following is an example of value setting for your URL of the application: "url": **http://www.synology.com/** "url": "3rdparty/xxx/index.html"
allUsers	This key determines whether the menu items can be seen by users when they log in with admin account. If you would like to have all users see the menu items, please set the key value as below: "allUsers": true The default setting is that only the **admin** can find the application.

In the **INFO** file, you can define the key **dsmuidir**, whose value is a DSM directory name in the **package.tgz file**, Package Center will automatically link/unlink it to **/usr/syno/synoman/webman/3rdparty/[package name]** based on the key when you start/stop the package. You should also define **dsmappname** in the **INFO** file to integrate the package with DSM applications.

It is suggested to add a 72x72 PACKAGE_ICON.PNG icon and a 120x120 PACKAGE_ICON.PNG icon to the SPK file which will be shown in Package Center.

Integrate with DSM Web 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.

To check whether a user has logged in, run the command below in the CGI:

```
/usr/syno/synoman/webman/modules/authenticate.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.

Below is an example:

```
#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;
}
```

This CGI runs the command below to check whether a user has logged in. After checking up, it will print out the user's name if the user has logged in.

```
/usr/syno/synoman/webman/modules/authenticate.cgi
```

DSM Backward Compatibility

Weak link is a property of Apple's development framework which can ensure backward compatibility. You can learn more about weak link from **here** and **here**. GCC has a similar property called "weak symbol." We utilize such capability to provide a weak link framework in **libsynosdk** for backward compatibility too. You can find available headers in **usr/syno/include/libsynosdk** in chroot environments. Each function prototype in **synosdk/*_p.h** is decorated by a macro telling you when this function is added into **libsynosdk**. Therefore, you can invoke a function added in DSM 4.2 like this:

```
/* DO NOT include *_p.h directly */
#include <synosdk/user.h>
#include <synosdk/service.h>


/* example, SYNOServiceHomePathCheck is available since DSM 4.2 */
if (SYNOServiceHomePathCheck) {
SYNOServiceHomePathCheck(szPath, TRUE, TRUE, &pResult);
} else {
```

```
/* implement alternative to SYNOServiceHomePathCheck here */
}
```

As a result, when your application runs on DSM 4.2 and later, function **SYNOServiceHomePathCheck** in **libsynosdk.so** is invoked. On DSM earlier than 4.2, **else-part** will be executed as replacement to **SYNOServiceHomePathCheck**.

Show Messages to Users

If you want to prompt users with a message before they install, upgrade, or uninstall a package in Package Center, you can write these messages into the **desc** key in **install_uifile**, **upgrade_uifile**, or **uninstall_uifile**. Please refer to “conf” page 21 for more information.

If you want to send a prompt message to users after they install, upgrade, uninstall, start, or stop a package in Package Center, you can have implemented them into the **$SYNOPKG_TEMP_LOGFILE** variable in the related scripts. For example,

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

If you want to prompt users in the language specified in their DSM settings, you can refer to the **$SYNOPKG_DSM_LANGUAGE** variable for language abbreviation as shown in the scripts below:

```
case $SYNOPKG_DSM_LANGUAGE in
      chs)
           echo "简体中文" > $SYNOPKG_TEMP_LOGFILE
      ;;
      cht)
           echo "繁體中文" > $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 "日本語" > $SYNOPKG_TEMP_LOGFILE
    ;;
    krn)
        echo "한국어" > $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 see the "scripts" section on page 29 for more information.

Otherwise, you can use **/usr/syno/bin/synodsmnotify** to send messages to DSM users. For example, the strings "**title**" and "**messages**" are sent to the "**administrators**" group.

```
synodsmnotify @administrators title messages
```

Create PHP Application

Apache and PHP engines are built-in with DSM, which allows you to develop web applications and store files in the *web* space to build your own website. Hence, these files should be compressed into **package.tgz.** In the **start-stop-status** script, you should link or copy them to the web space when starting a package. In DSM, the default web space is the **web** shared folder. You can get the path via **/var/services/web** which is a symbolic link pointing to the actual path in the volume.

The default web space will be created and Apache will be running after the administrator of the DiskStation enables **Web Station**. You can configure the **install_dep_services** and **start_dep_services** keys with the **apache-web** value in the **INFO** file to make sure **Web Station** is enabled before the package installation. Then, you can provide a URL in order to access your page to **adminurl** (ex:"/myapp/index.html") in **INFO** which will be displayed in Package Center to tell the client which URL to open. When the package is stopped, this URL should not be accessible, and you can remove or unlink the website folder in the web space, or let the webpage redirect to an error page.

In addition, it is suggested to add an icon to DSM's main menu so that users can click the icon to launch the application intuitively.

If you would like to modify the apache config file located in **/usr/syno/apache/conf/httpd.conf-user** when starting the package with the **start-stop-status** script, you need to add the **apache-web** value to **startstop_restart_services** to restart Apache to reload the config file. Besides, you should remove what you have modified when the package is stopped.

Run Scripts When the System Boots

If you would like to run scripts when the system is booting or shutting down, you can write scripts in **start-stop-status**. This script will be executed with the "start" or "stop" parameter, under the condition that the package is enabled. If you would like a script to be executed during the booting or shutting down process, you can put a startup script in **/usr/local/etc/rc.d/**. Following are the rules for the startup script:

1. It must contain the suffix "**.sh**". For example, "**myprog.sh**".
2. The permission must be 755.
3. It must contain the options "start" and "stop". When the system boots up, it will call "**myprog.sh start**"; when it shuts down, it will call "**myprog.sh stop**".

You can refer to the scripts in **/usr/syno/etc/rc.d/**, which are scripts of Synology's default services.

Create a Share Folder

If you would like to create a share folder to be accessed by the end user, the **synoshare** command can help you create it in a specified volume. For example,

```
synoshare --add [share folder name] "" /volumeX/[share folder name] ""
"admin,guest" "" 1 0
```

If you are not sure which data volume to store the share folder, you can choose the volume where the package is installed by **/var/packages/[package name]/target**. On the other hand, you can execute the servicetool command, ex: **"/usr/syno/bin/servicetool --get-alive-volume**" to request one available volume.

If you want to check if a share exists, you can check the return value of the **synoshare** command as follows:

```
synoshare --get [share folder name]

if [ $? != 0 ]; then

//share folder doesn't exist

else

//share folder exists

fi
```

After a package is uninstalled, the share folder which you created can be reserved. Removing the shared folder is not recommended as it will also remove the user's personal data.

Install Package Related Ports Information into DSM

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

Service Configure File Name

The file name should follow the naming convention **SYNOPKG.sc** (ex: **SurveillanceStation.sc**). **SYNOPKG** should be the package name that is specified by key "**package**" in **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 explanation for the strings and keys:

Section/Key	Description	Value	Default Value	DSM Requirement
service_name	Required Usually a package only has one unique service name, but if your package needs more than one ports description, you can define service_name2, service_nam3, … Note: service_name cannot be empty and can only composed from “a~z”, “A~Z”, “0~9”, “-”, “_”, “.”	Unique service name	N/A	4.0-2206
title	Required English title which will be shown on field Protocol at firewall build-in selection menu.	English title	N/A	4.0-2206
desc	Required English description which will be shown on field Applications at firewall build-in selection menu.	English description	N/A	4.0-2206
port_forward	Optional If set to “yes”, your package service related ports record will be list when user set port forward rule from build-in applications. Otherwise your record will not be list.	“yes” or “no”	“no”	4.0-2206
src.ports	Optional If your package service has specified source ports, you can set them to this key. The value should contain at least port numbers, and the default protocol is tcp + udp. Ex: 6000,7000:8000/tcp,udp means source ports are 6000, 7000 to 8000, all ports are tcp + udp.	ports/protocols ports: 1~65535 (separated by ‘,’ and use ‘:’ to represent port range) protocols: tcp,udp (separated by ‘,’)	ports: N/A protocols: tcp,udp	4.0-2206

Section/Key	Description	Value	Default Value	DSM Requirement
dst.ports	Required Each service should have destination ports, the value should contain at least port numbers, and the default protocol is tcp + udp. Ex: 6000,7000:8000/tcp,udp means destination ports are 6000, 7000 to 8000, all ports are tcp + udp.	ports/protocols ports: 1~65535 (separated by ',' and use ':' to represent port range) protocols: tcp,udp (separated by ',')	ports: N/A protocols: tcp,udp	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 configure file is prepared, you can install it with a tool named **servicetool** on the DiskStation. The service configure file should be installed on the DiskStation on the step **postinst**, and remove on the step **preuninst**. Below is an example showing how to install and uninstall **SYNOPKG.sc** on the DiskStation:

For **postinst** script:

```
/usr/syno/bin/servicetool  --install-configure-file  --package
/var/packages/SYNOPKG/target/SYNOPKG.sc
```

For **preuninst** script:

```
if ["UNINSTALL" = "$SYNOPKG_PKG_STATUS" ]; then

 # Note: only remove service configure file when uninstalling the
package

 /usr/syno/bin/servicetool  --remove-configure-file  --package
SYNOPKG.sc
```

Publish Synology Packages

Get Started with Publishing

Starting to publish on Synology Package Center only requires a few simple steps. Here's how you do it:

1. Acquire a Synology Checkout Merchant Account via a Synology specialist.
2. Read and accept the Developer Distribution Agreement. 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 installs, online reviews, engagement, and user retention. Synology users expect high-quality packages, even more so if they've spent money on them.

Submitting the Package for Approval

Before you publish your package on Package Center and distribute it to users, you need to get the package ready, test it, and prepare your promotional materials if needed. Please see the checklist below before submitting your package to us.

Confirm Package Size

The overall size of your package can affect its design and how you publish it on Package Center. Currently, the maximum size for an SPK published on Package Center is **100 MB**.

Free or Paid Package

On 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 registered a MyDS 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 package, it cannot ever be changed to a paid package.
- If you publish your app as a paid package, you can change it to free at any time (but cannot then change back to paid).
- If your package is paid, you need to set up a Synology Checkout Merchant Account before the package can be published.

Set Price for Your Package

If your package costs money, Synology lets you set prices for your package in US currency for users in markets around the world. Before you publish, consider how you will price your package and what your prices will be in various currencies.

Prepare Screenshots

When you publish on Package Center, you must supply a variety of high-quality screenshots to showcase your package or brand. After you publish, these appear on your package details page, or elsewhere. These screenshots are key parts of a successful package details page that attracts and engages users, so you should consider having a professional produce them for you. They show what your app looks like, how it's used, and what makes it different.

Submit Your Package

When you think you are ready to publish, send an email to Synology specialist (package@synology.com) to make your submission.

Make sure that:

- You have the right version of the package.
- You have provided a link to download the package.
- You have provided a link to download all screenshots.
- You have provided package description as what it does.
- You have provided package change log as what was updated in this version.
- Package's pricing to be free or paid. (first time submission)
- You have provided the correct link to your web site and the correct support email address.
- You have acknowledged that your package meets the Developer Distribution Agreement and also the **Terms of Service** from Package Center.

Responding to User Issues

After you publish a package, it's crucial for you to support your customers. Prompt and courteous support can provide a better experience for users, which results in higher downloads and more positive online reviews for your packages. Users are likely to be more engaged with your package and recommend it if you are responsive to their needs and feedback.

There are a number of ways that you can keep in touch with users and offer them support. The most fundamental is to provide your support email address on in your package details page. Beyond that, you can provide support in any way you choose, such as a forum or mailing list. The Synology technical support team does provide 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:

- Put a link to your support resources on your web site and set up any other support such as forums.
- Provide an appropriate support email address on your package details page and respond to users when they take the time to email you.
- Acknowledge and fix issues in your package. It helps to be transparent and list known issues on your package details page proactively.
- Publish updates as frequently as you are able, without sacrificing quality or annoying users with too-frequent updates.
- With each update, make sure to provide a summary of what's changed. Users will read it and appreciate that you are serious about improving the quality of your package.

Revision History

This table describes the changes to the Synology DiskStation Manager 3rd-Party Apps Developer Guide.

Date	Note
2008-06-16	1. Document originally released
2009-02-09	1. Add Freescale 8533 tool chain information
2009-03-09	1. Add Marvell 6281 tool chain information 2. Add Desktop Icon chapter
2010-05-20	1. Add related information for 10 models 2. Change all DSM 2.0 & 2.1 to DSM 2.3 3. New screenshots for desktop icon & DSM application 4. Change configuration files and tool chains 5. DS1010+ uses Intel Atom D510
2010-05-28	1. Revise Tool Chain description 2. Revise 88f5182-config to 88f5281-config in Kernel Module 3. Revise path description of application.cfg / desktop.cfg
2010-11-29	1. Rename the document to “Synology DiskStation Manager 3rd-Party Apps Developer Guide” 2. Add DSM 3.0 Integration section
2011-10-31	1. Add Synology package creation section 2. Update DSM tool chain information
2012-03-20	1. Add the guideline to application storage, web application running on Apache and create a shared folder
2012-09-19	1. Add Freescale QorIQ tool chain information
2013-03-11	2. General update for DSM 4.2 release. 3. Added section regarding payment framework. 4. Updated formatting.
2013-05-06	Add Marvell armada370 tool chain information.
2013-05-29	Add Marvell armadaxp and evansport tool chain information.
2013-06-05	Update qoriq, armadaxp, armada370, evansport $CC variable.