NTxPKG Specification V1.2.0 (Draft)

💡 **版本：V1.2.0 (Draft)**

主编： **Kai HAN 韩锴**

适用范围： NT3

版本

Version	Status	Date	Content	Comment
V1.0.0	Released	2022/12/30	Initial Version	Official Release @DMT 2023/1/13
V1.1.0	Release	2023/04/03	Add "assembly" package. Update metadata.	Official Release for D3

(The current content is V1.2.0 Draft. The latest offical release refers to V1.1.0)

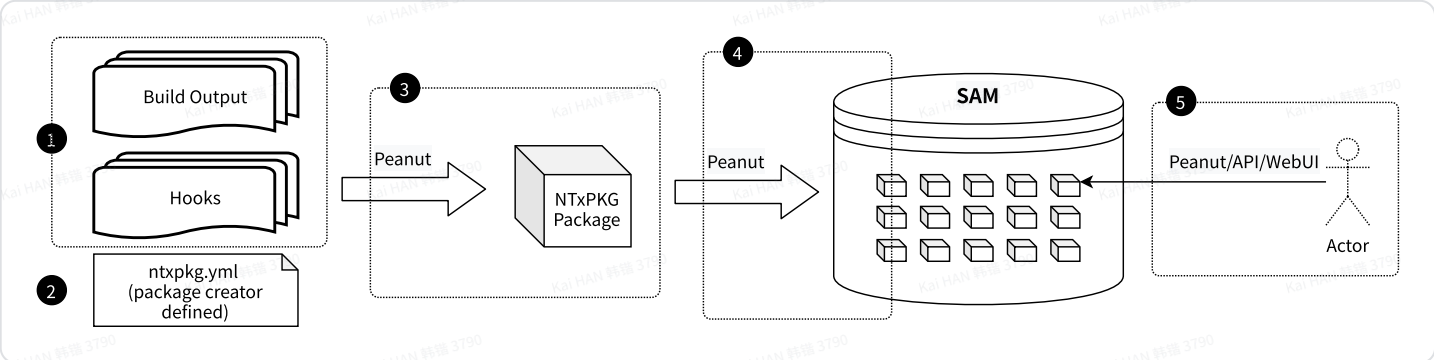
For end-user facing documentation about "peanut", check out the "Peanut User Guide v0.0.3".

This document is an attempt to record the internal data structures of the ".ntxpkg" package manager, named "peanut". The canonical implementation of the ntxpkg format is peanut and much of this information is gleaned from reading the peanut source code (if this document cannot fulfill your curiosity).

Background

NT3 strategies of serviclization and platformization complicate package management. It is needed that the vehicle-level **consistent interface and format** to package and deliver software to simplify cross-team integration. Therefore, rather than pervasive ad-hoc .zip files in the NT2 era, a unified format ".ntxpkg" and its management tool - peanut - were raised.

1. Lifecycle



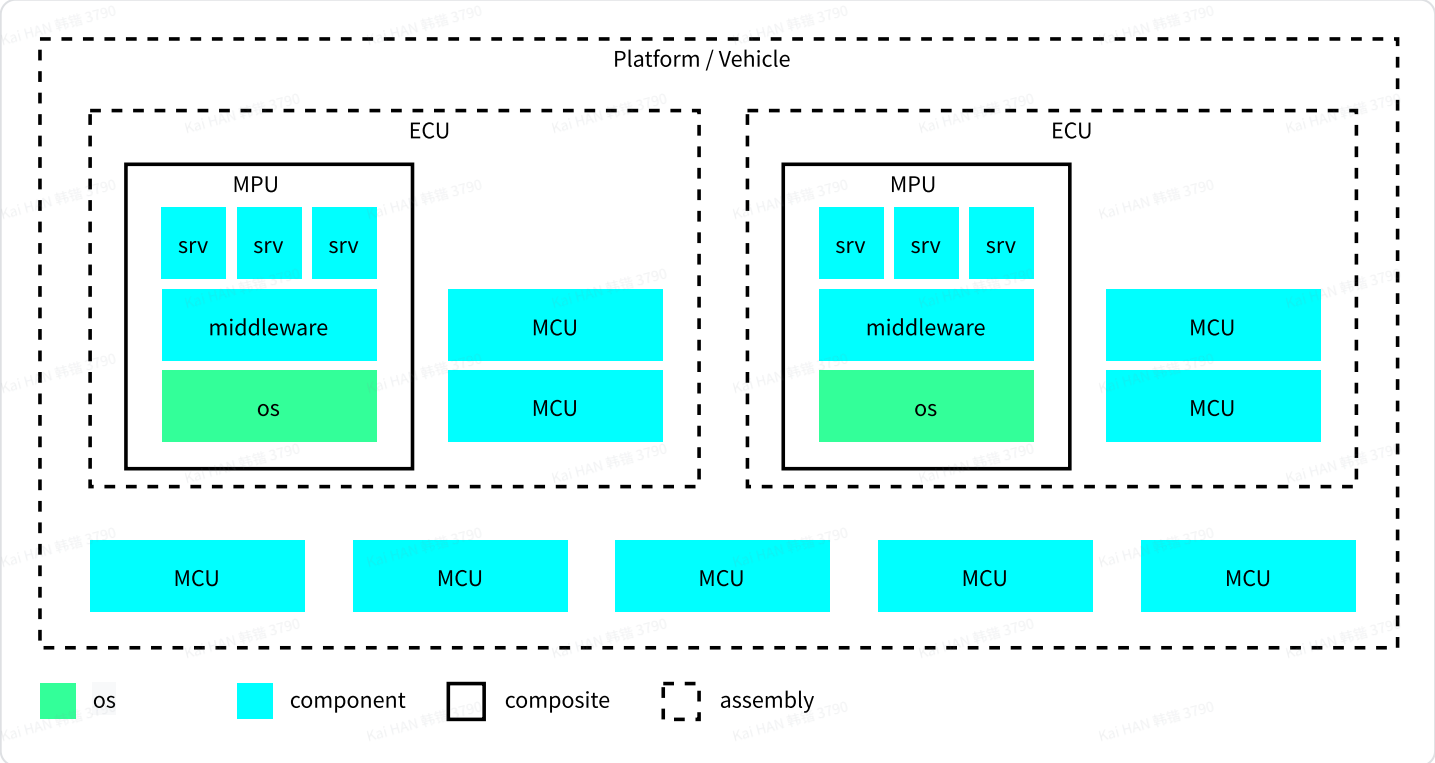
NTxPKG packages' lifecycle is made up of the following phases,

1. Package creator prepares package's content including binaries, configuration, data, hook scripts, source code etc.
2. Package creator defines "NTxPKG.yml", which contains a series of metadata.
3. "peanut" creates a .ntxpkg package with #1, #2. The created package also contains a NTxPKG.yml whose content is slightly different from the one in #2.
4. "peanut" publishes the .ntxpkg package to SAM.
5. The package in SAM will be viewed, and consumed by other teams or packages.

2. Four Types of NtxPKG Packages

NTxPKG defines 4 types of packages: 'component', 'os', 'composite', and 'assembly'.

The diagram below shows the relationship between these 4 types.



'os' and 'component' packages, which contain binary files, data etc., are concrete.

'os' package contains a complete operating system, which typically has kernel, root filesystem, bootloader etc. For instance, realtime linux, android, qnx, vendor customized ubuntu, etc. However, RTOS is NOT 'os' but a 'component' package.

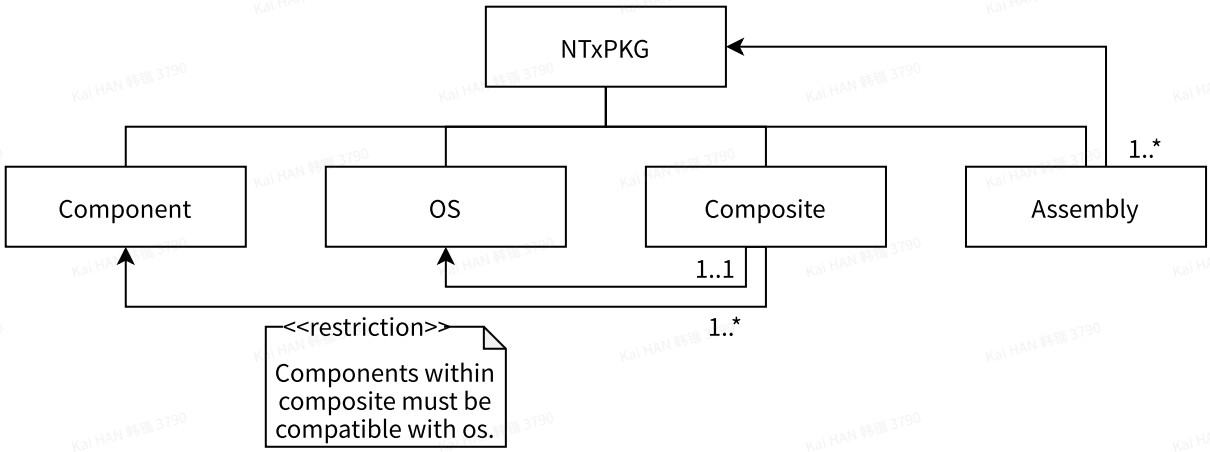
'component' package refers to different things. A soa service, an application, the middleware are both 'component'. A component package is very much like an user application. They can both be installed on an OS. Besides these, a MCU software, a bootloader are also 'component' although they won't be really installed on any OS.

Unlike 'os' and 'component', 'composite' and 'assembly' are simply the relationships with other packages. They are not concrete. But as a container, they define the dependencies of sub packages which are made up of them. With the dependencies and hook scripts, 'composite' and 'assemly' packages are always able to re-build the concrete packages - 'peanut assemble' command will do it.

'composite' and 'assembly' packages are slightly different. A composite must have one and only one 'os' package and multiple components. The other components must support the previous os and its arch. For example, to create a composite with os 'rt-linux' and arch 'aarch64', and some applications. It is sure that all applications must support os 'rt-linux' and arch 'aarch64'.

On the other hand, 'assembly' can contain arbitrary and heteroid packages. For example, one 'assembly' can have a mpu with rt-linux/x86_64, a mpu with Android/aarch64 and a MCU with RTOS.

The following is the logic model of 4 types.



3. NTxPKG.yml (package creator defined)

As mentioned in Lifecycle #2, the package creators have to define a NTxPKG.yml file to guide peanut to create the package. The section depicts the specification of this NTxPKG.yml.

NTxPKG.yml is in yaml format. It contains the following items,

Name	Data Type	Mandatory	Description
name	string	yes	the unique identifier of a package. Only cha number, "-", and "_" are allowed.
type	string	yes	Define the type of pacakge. Only 4 valid opt - component - os - composite - assembly The more detail about the 4 types please re section "Types of Packages"
version	3 numbes seperated by "."	no	A version is made up of 3 positive numbers If not specified in NTxPKG.yml, then must p CLI parameter.
prerelease	string with an optional number	no	Prerelease have a lower precedence than tl associated normal version.
os	array of string	- yes for "component" and "composite" - no for "os" and "assembly"	All supported "os"es that a component or c can use. The value must refer to an existing typed package.
arch	array of string	- yes for "os", "component" and "composite" - no for "assembly"	All supported "arch"es that a component o composite can use. The value must refer to existing "os" typed package.
group	string	no	The 'group' of a package. Within a group, a name is not allowed to be repeated. A 'group' refers to a car type, car model etc

			The default value is "default-group".
variant	array of key-value pair	- yes for 'assembly' - no for all except 'assembly'	In addition to "os" and "arch", other indicat to distinguish products. Package creator can define ANY key / value There is no restriction on content and quan See examples in the right column.
deployDependencies	array of 'package expression'	- yes for 'assembly', 'composite' and 'component' - no for 'os'	List of deployment dependencies. See belo for more detail about 'package expression'.
revisions	array of git revision	no	The latest revision of git repo which are ma the package. If the package contains multip repos then there will be more revisions.
git	boolean	no	Default is 'False'. 'True' indicates that the pa just contains git revision information which used to fetch all source code.
include_src	string, separated by ";"	no	Default is empty. If it is not empty, then it in that package contains some folders and/or located by 'include_src'.

4. NTxPKG.yml (package generated)

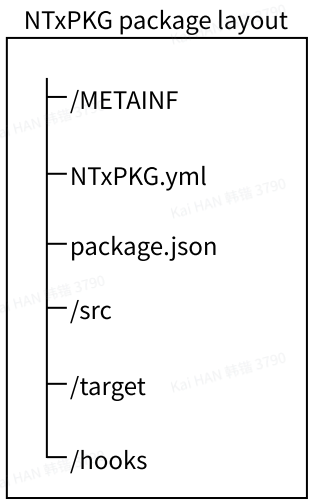
The NTxPKG package contains another ntxpkg.yml file. This file (created by Peanut if package is created by Peanut) is slightly different from the one written by package creator. The following table lists all the differences.

Name	Data Type	Description	Example
os	~~array of string~~ string	While package creator defined NTxPKG.yml lists all supported 'os'es, creating package needs to specified the 'os' of current package, and it must be included in supported os list.	os: vdf-rt-linux
arch	~~array of string~~ string	While package creator defined NTxPKG.yml lists all supported 'arch'es, creating package needs to specified the 'arch' of current package, and it must be included in supported arch list.	arch: aarch64
deployDependencies	array of 'package expression'	If package creator defined NTxPKG.yml defines deployDependencies using "<=", ">=" etc, generated package will "lock" the version, and use "==" to indicate the version.	From (package cr deployDependen - servicexyz == 1. - vdf-rt-linux >= 1 To (package gene deployDependen - servicexyz == 1. - vdf-rt-linux ==

5. Package Content Specification

This section explains the internal layout of the ".ntxpkg" package.

Package Layout



A NTxPKG file is a ZIP archive that must contain the following files and directories:

- `METAINF` directory:
  - *Reserved* for security-related content, eg. certificate, signature etc.
- `NTxPKG.yml` file: the file contains metadata.

- `package.json` file: An optional file that describes the very detailed components and dependencies information. The schema has dedicated grammar defined NT3 Package Schema .
- `target` directory:
  - If the ntxpkg is a "component" or "os" typed one, then "target" directory contains all binary files, config, data of the software. These are all valid contents for target directory: a npk file, an apk file, soa framework, operating system (kernel image, rootfs, dtb, etc.) ...
  - If the ntxpkg is a "composite" or "assembly" typed one, then the "target" directory intentionally remains empty.
- `src` directory:
  - If package is created with argument "--include-src", then the specified folders and files will copy to src directory.
- `hooks` directory: the directory contains hook scripts that will be invoked in stages of package lifecycle. "peanut" offers default hook scripts. Provide your own scripts only when you have different behavior.
  - `preinstall.sh` : The script will be executed before installation. The default behavior is empty.
  - `install.sh` : The script will be executed to install software. The default behavior is to copy the content from "target" directory to the operating system, and simply replace the existing content.
  - `postinstall.sh` : The script will be executed after installation. The default behavior is empty.
  - `mount.sh` : The script is only available in 'os' packages, and it will mount rootfs to a folder in host machine.
  - `unmount.sh` : The script is only available in 'os' packages, and it will unmount rootfs which mounted by mount.sh previously.

6. Semantic Versioning

Peanut's version complies with semantic versioning standards.

It comprises two sections, the "version" and the "prerelease". The version includes 3 numbers, MAJOR.MINOR.PATCH. The prerelease is a dot-separated identifier and number. **BE CAUTION** that semversion accepts any number of identifiers in prerelease but NTxPKG allows only one. See samples for version and prelease in section "3. NTxPKG.yml (package creator defined)".

7. Package Filename

The scheme of NTxPKG name is

```
[PACKAGE_NAME].[VERSION]{-[PRERELEASE]}.[OS].[ARCH].{[VARIANTS]}.ntxpkg
```

As the name implies, PACKAGE_NAME, VERSION, PRERELEASE, OS, ARCH and VARIANTS are from package's metadata - they are name, version, prerelease, os, arch and variants, correspondingly.

On the other hand, to comply with the NT3 NPT software version naming rule v0.1, tools like 'peanut' will convert the NTxPKG's original name to platform's standard name (using NTxPKG's metadata and some other input). Due to this document defines the name to facilitate project management, NTxPKG doesn't include all things they need.

8. DeployDependencies

DeployDependencies is a critical section for defining a NTxPKG. Its value refers to different topics for different types of package.

- For "component" package: "deployDependecies" depicts all dependecies that must be installed that the package needs during runtime. For example, a "LoBeamMgr service" needs "soa-framework" to be installed to launch, therefore, the "deployDependencies" of "LoBeamMgr service" must contain "soa-framework", as shown below,

```
deployDependencies:
    - soa-framework == 1.2.3
```

- For "composite" package: "deployDependecies" lists all packages that will be included in the final composition. It must contain **one and only one** "os" typed package. For example,

```
deployDependencies:
    - nvos == 1.8.2
    - soa-framework == 1.8.2
    - srv1 == 1.2.3
    - srv2 == 1.3.4
```

- For "assembly" package: similar to composite package, "deployDependecies" lists all packages that will be included in the final composition. Due to an assembly containing heteroid packages, there is an important difference - the package owners must specify the os/arch (for component and composite) and variants (for assembly) for each package, as the following samples did,

```
-- NTxPKG.yml for NT3-platform --
deployDependencies:
    - vdf==1.8.2;target=target # vdf is another assembly, variant "target=target
    - cdf==1.2.2;target=target # cdf is another assembly, variant "target=target
```

```
-- NTxPKG.yml for VDF --
deployDependencies:
    - vdf-1-a==1.8.2;os=rt-linux;arch=aarch # vdf-1-a is a composite, os and arc
    - vdf-1-m==1.2.0;os=nvos;arch=tc399 # vdf-1-m is a component, os and arch ar
    - vdf-2-m==1.2.0;os=nvos;arch=tc399 # vdf-2-m is a component, os and arch ar
```