NPM Package Specification

NOTE: This page is under the management of FHIR Infrastructure. All substantive changes to this specification require formal review and approval by the FHIR-I Working Group.

A FHIR package groups a coherent collection of conformance resources, like StructureDefinitions and ValueSets into an easily distributed NPM package. FHIR packages use a subset of the features used by npm packages.

One use of packages is for FHIR Implementation Guides - they are all published as NPM packages, one package for each IG. See below for further discussion.

Package name

Each package has a canonical name (a globally unique identifier). A package name consists of two or more namespaces separated by a dot. Each namespace starts with an lowercase alphabet character followed by zero-or-more lowercase alphanumeric characters or a dashes.

The first part of the namespace should identify the author, the authoring organization, or region. The second part of the namespace should identify the functional scope or purpose of the package.

Examples:

```
hl7.fhir.core (main build)
hl7.fhir.us.core
hl7.au.base
hl7.nl.medmij
ihe.pix
ihe.mhd
```

Names and Versions

If the package spans multiple FHIR versions, it may be appropriate to add that in the package name in the form of: Rx, where x is the release version.

Example:

```
hl7.fhir.r3.core
```

See further notes below when considering whether this is a good choice.

Scoping

A fhir package should not contain an npm scope. Packages published by HL7 or the FHIR foundation SHALL not contain a scope.

Note: Npm packages can [contain a scope]. Scopes are a way of grouping related packages together. A scope is usually the owner, and was introduced relatively late in the npm standard. Only the user can add packages to his/her scope. It's a way to identify the official packages from organizations. In Nuget that same logic is enabled with the package prefix, which is cleaner and more readable. A npm scope, starts with at (@) and ends with a slash '/'. It makes both the client and the server more complex to implement. And we want easy and broad adoption.

Name Management

Names under hl7.fhir. or fhir. are assigned by the FHIR Product Director - contact fhir-director@hl7.org for assistance. Implementers may (of course) use any names they wish in other namespaces, but should be careful to preserve global uniqueness.

For Packages published by HL7, when implementers create a subset of the package, implementers are pre-approved to use the following suffixes:

.terminology .conformance .[type]

e.g. hl7.fhir.us.core.structuredefinition - creating packages with names like this does not require approval. Other combinations - use a name in a different namespace, or ask for approval.

Versions

All packages have a mandatory version. [SemVer] 2 SHALL be used

When comparing two versions that start with a digit (0..9), they SHALL not be interpreted as a string, but as a structured numerical version reference. Package generators SHALL ensure that versions starting with a digit have more recent versions with higher numbers.

Version strings SHALL contain only letters, numbers, and the characters ".", "_", and "-"

Version references

When packages point to dependencies they should refer to the whole package version number and not use wildcards, except for the patch version in a semver version reference:

```
 "hl7.fhir.core" : "3.0.x"
```

This x here means that it should a package resolver should accept the package with the highest found patch number.

Note: Npm has elaborate logic package version references. It allows version forwarding, ranges, wildcards, etc. The FHIR package standard does not allow this. The only ranges that are allowed are wildcards (x) for the patch version as described here. Most of the matured package managing tools, have acknowledged that having advanced version references would create instability and more confusion than help. Fhir packages also need to be consumed by non technical users. Installing package dependencies should not need a manual.

Note: the npm version format file:[path] may be supported by some tools that use FHIR NPM Packages, but is not acceptable as a package location in the FHIR Package server, the FHIR current build, or any other public FHIR servers.

Version selection strategy

One of the main problems that a package managing standard has to solve is how to resolve deep dependency collision - dependencies that each have their own dependency on a different version of the same package.

There are several strategies in play with most packaging standards.

1. The owner of the deep dependency should strictly follow the semver rules, and not introduce breaking changes 2. The owner of the consuming dependencies can play loose and fast with their dependency range 3. The client package tool can apply a set of algorithms to upgrade one deep dependency, downgrade the other, or keep multiple dependencies. This can cause type mismatches if the consuming tool lets the shallow dependencies interact.

See also this analysis] of go packaging.

For FHIR we still need to define more precisely how a project that consumes different versions of the same package should resolve these issues. This is part of a bigger versioning discussion in FHIR.

Format

A FHIR package is a tarball (tar in gzip). The package contains

- a subfolder named 'package'
- a package manifest (package/package.json)
- an index file (package/.index.json)
- A set of resource files, also in the package subfolder
- It MAY contain additional content, like example resources or documentation:
    - such files SHALL not be in the package subfolder
    - Example resources SHOULD be placed in an /examples folder. Tools working with examples, SHALL understand this. Note that the interpretation of what is an example resource can be unclear in some cases
    - this may include XML schemas in an "xml" subfolder
    - this may include openAPI files in an "openapi" subfolder
    - this may include turtle RDF representations in an rdf folder
    - Package consumers SHALL ignore content in other subfolders that they do not use (and most consumers will only use the resources in /package)

Tarballs SHALL be in the original tarball format (e.g. a 99 character file name length limit).

Note: discussion on this - see [[1]]

Package manifest

A package manifest is a json file called 'package.json'. It conforms to the npm package.json format, but contains only a subset of properties. Other properties are allowed, but should be ignored by a FHIR package consumer.

```
{
 "name": "hl7.fhir.us.acme",
 "version" : "0.1.0",
 "canonical" : "http://hl7.org/fhir/us/acme",
 "url" : "http://hl7.org/fhir/us/acme/Draft1",
 "title" : "ACME project IG",
 "description": "Describes how the ACME project uses FHIR for it's primary API",
 "fhirVersions" : [ "3.0.0" ],
 "dependencies": {
    "hl7.fhir.core": "3.0.0",
    "hl7.fhir.us.core": "1.1.0"
 },
 "keywords": [
   "us",
   "United States",
   "ACME"
 ],
 "author": "hl7",
 "maintainers": [
   {
     "name": "US Steering Committee",
     "email": "ussc@lists.hl7.com"
   }
 ],
 "license": "CC0-1.0"
}
```

Package Manifest Properties

- name - mandatory - the globally unique identifier of the package as described above
- version - mandatory - SHOULD use [SemVer])
- canonical - optional (but required for IGs - see below)
- url - optional - where the human readable representation (e.g. IG) that represents this version of this package is published on the web (if there is such a thing)
- homepage - optional - The url to the project homepage (e.g. where information about the project that produced the package can be found on the web)
- title - optional short description for the package
- description - mandatory
- fhirVersions - an optional list of FHIR versions that this package depends on. This is only used in the case where there is no dependency on a core package. Note: this usually happens when the package is based on a transient unpublished version of FHIR, which was allowed in the past but will no longer be allowed. So this should only be seen in old balloted packages
- dependencies - SHOULD be at least one to a fhir core package
- keywords - optional
- author - mandatory
- maintainers - optional
- license - optional. Follow the [spdx naming convention]

Other properties (e.g. from base NPM spec) are ignored if present

Dependencies

A FHIR package may have dependencies.

A FHIR package SHOULD have a dependency to at least one FHIR core package (e.g. hl7.fhir.r3.core, hl7.fhir.r4.core)

Package consumers should be aware of these dependencies and resolve them by downloading and installing each dependency recursively.

Package Content

A package contains a set of FHIR resources in the JSON format for the specified FHIR version. Each resource is saved in a separate .JSON file in the 'package' directly under the root.

Resources may have a filename of [Type]-[id].json, but other names are possible

The package SHOULD have a file .index.json that indexes the resources in the package folder.

.index.json

This file exists to allow tools to rapidly find the resources they are looking for without having to load and read all the files. It is a json file with the following format:

```
 {
   "index-version": 1,  // a fixed number that identifies the version of this file; tools should rebuild the .
index.json file if they encounter an unrecognised number
   "files": [  // an array, with one entry per file in the same folder as the .index.json
      {
        "filename": "[name].json",  // the name of the file
        "resourceType": "[Type]",   // the type of the resource
        "id": "[id]", // the id assigned to the resources. Note: resources SHOULD have an id, but in some
workflows, none is assigned in the package
        "url": "url", // the canonical url, if the resource has one (e.g. a property "url" which is a primitive)
        "version": "[version]", // the business version, if the resource has one (e.g. a property "version" which
is a primitive)
        "kind": "[kind]",      // the value of a the "kind" property in the resource, if it has one and it's a
primitive
        "type": "[type]"      // the value of a the "type" property in the resource, if it has one and it's a
primitive
    }
  ]
}
```

The .index.json file can be rebuilt at any time. it SHALL not contain any information not extracted from the resources in the same folder as it.

Examples

Resources that are examples are found in an examples folder that is a sibling to the package folder. File names follow the same pattern.

Te exampes folder may also have an .index.json file that describes the examples.

Meta packages

Packages MAY instead of having content only reference other packages. This is called a meta package. Its purpose is to group certain packages and their use case together.

Example: acme.api

Containing only references to acme.api.terminology acme.api.extensions acme.api.profiles

Implementation Guides and packages

All FHIR Implementation Guides are published as NPM packages, one package for each IG. This is the primary way to distribute IGs for computational use (validation, code generation, etc).

Package Manifest properties for IGs

- name = ImplementationGuide.packageId
- version = ImplementationGuide.version - note: Semver SHALL be used for packages published by HL7 or the FHIR Foundation
- canonical = ImplementationGuide.url - required for IGs. This matches the name, and is constant through the life cycle of the IG
- url = ImplementationGuide.manifest.rendering - required for IGs
- title = ImplementationGuide.title
- description = ImplementationGuide.description
- dependencies = from ImplementationGuide.dependsOn
- author = ImplementationGuide.publisher
- maintainers = ImplementationGuide.contacts
- license = ImplementationGuide.license - mandatory for packages published by HL7 or the FHIR Foundation

IG Subsets

Additional packages containing subsets of a package information can be created. E.g. if a package is hl7.fhir.us.core, then the package hl7.fhir.us.core.tx would contain only terminologies resources. Except as described for multi-version IGs, sub-packages SHALL not additional information not in scope for the main package, though they may contain additional kinds of computable information not in the main package e.g. hl7.fhir.us.core.xml might contain schematron not found in the main package (but only for profiles that are found in the main package).

Known subsets (others can be used, but if these sub names are used, they must be for the described purpose):

- .profiles - all structure definitions
- .tx - all code systems, value sets, concept maps
- .api - CapabilityStatements, Search Parameters, Operation Definitions

Multi-version support

When an implementation guide covers multiple FHIR version, the package structure that represents it follows this pattern:

`hl7.fhir.example`

- in \package - only package.json and an IG resource in the latest version
- in package.json: dependencies on hl7.fhir.example.r2 and hl7.fhir.example.r3
- in IG: latest resource

`hl7.fhir.example.r2`

- in \package - package.json + resources applicable to R2
- in package.json: dependencies on hl7.fhir.r2.core

`hl7.fhir.example.r3`

- in \package - package.json + resources applicable to R3
- in package.json: dependencies on hl7.fhir.r3.core