

Using the FHIR Validator

This page documents the use of the FHIR Validator jar for validation. See [[Validating Resources](#)] for further information and other options for validating resources.

- [1 Downloading the validator](#)
- [2 Running the validator](#)
- [3 Choosing what to validate](#)
- [4 Managing Output](#)
- [5 Choosing the version](#)
- [6 Validating against an implementation guide](#)
 - [6.1 Loading an implementation Guide](#)
 - [6.2 What to validate against](#)
 - [6.2.1 Validating a single resource in a bundle](#)
 - [6.3 Validating References](#)
 - [6.3.1 assumeValidRestReferences](#)
- [7 Other Validation Parameters](#)
 - [7.1 Locale](#)
 - [7.2 Terminology Server](#)
 - [7.2.1 SNOMED CT](#)
 - [7.3 MustSupport](#)
 - [7.4 Extensions](#)
 - [7.5 Questionnaires](#)
 - [7.6 Extensible binding warnings](#)
 - [7.7 Level](#)
 - [7.8 Best Practices](#)
 - [7.9 Language / Display](#)
 - [7.10 Native Validation](#)
 - [7.11 Logging](#)
 - [7.12 SecurityOnly](#)
 - [7.13 Crumb Trails](#)
 - [7.14 Show Validator Internal Times](#)
 - [7.15 Validating CDA Documents](#)
- [8 Other functionality provided by the Validator](#)
 - [8.1 Version Conversion](#)

Downloading the validator

To download the validator: [https://github.com/hapifhir/org.hl7.fhir.core/releases/latest/download/validator_cli.jar]

Running the validator

Note that you should always use the current validator (see above), irrespective of which FHIR Release you are validating. You need a current version of java to run the validator:

```
java -jar validator_cli.jar [params]
```

The params control how the validator works, and are documented here.

You can also use the validator for other things than validation: [Using the FHIR Validator to transform content, todo...](#)

Note if you get an error from java "Out of Memory Error" -- Increase your [java heap size configuration](#)

Choosing what to validate

The validator takes a series of parameters that indicate the resources to validate. There must be at least one source param.

Each source parameter can contain either:

- a URL that returns the resource to validate (authentication is not supported)
- a filename (relative to the current directory, or absolute)
- a directory that contains resources to validate (all files are validated if they are recognised as resources)
- a pattern: a directory followed by a filename with an embedded asterisk. E.g. foo*-examples.xml or someresource.*, etc

All other parameters are 'named parameters' - e.g. -name value. Any parameter preceded by a recognised name is interpreted as a source parameter

```
java -jar validator_cli.jar /tmp/resource.json
```

or

```
java -jar validator_cli.jar c:\temp\patient.xml
```

Managing Output

By default, the outcome of validating is simply printed to std out (the console / terminal / command window).

If the parameter `-output` is defined, a file will be created to contain the output. The file will contain an `OperationOutcome`, or if more than one resource is found, a `Bundle of OperationOutcome` resources

```
java -jar validator_cli.jar c:\temp\patient.xml -output c:\temp\validation.xml
```

Choosing the version

The validator checks the resource against the base specification. By default, this is the current build version of the specification. You probably don't want to validate against that version, so the first thing to do is to specify which version of the spec to use.

```
java -jar validator_cli.jar c:\temp\patient.xml -version 3.0
```

Valid values for version are `1.0 | 1.4 | 3.0 | 1.0.2 | 1.4.0 | 3.0.1` and the similar 3 or 5 letter versions strings for the current build

Note: alternatively, you can specify the version `-defn` or `-ig` parameters:

```
java -jar validator_cli.jar c:\temp\patient.xml -defn hl7.fhir.r3.core#3.0.2
java -jar validator_cli.jar c:\temp\patient.xml -ig hl7.fhir.r3.core
```

These are all synonymous with specifying the version, and maintained for backwards compatibility. It's simpler to use the version parameter.

Note: validating against the base FHIR specification requires that the package for the specific version be installed in your [FHIR Package Cache](#). If it's not, the validator will download it and install it.

Validating against an implementation guide

The validator can validate against an implementation guide. Do this involves 2 steps:

- loading the package for the implementation guide
- telling the validator what to validate against

Loading an implementation Guide

Tell the validator to load the package for an implementation guide using the `-ig` parameter:

```
java -jar validator_cli.jar c:\temp\patient.xml -version 3.0 -ig hl7.fhir.us.core#1.0.1
```

The package validator loads the relevant implementation guide. Rather than nominating the the package id, you can also provide the canonical url of the implementation guide:

```
java -jar validator_cli.jar c:\temp\patient.xml -version 3.0 -ig http://hl7.org/fhir/us/core#7C1.0.1
```

You don't have to specify the version:

```
java -jar validator_cli.jar c:\temp\patient.xml -version 3.0 -ig hl7.fhir.us.core
java -jar validator_cli.jar c:\temp\patient.xml -version 3.0 -ig http://hl7.org/fhir/us/core
```

In this case, the current published version of the IG is used.

Alternatively, the `-ig` parameter can contain:

- a URL that returns a relevant resource (profile, extension definition, code system or value set) to load
- the name of a file that contains relevant resource to load, i.e. this can be a single profile file
- the name of a directory that contains relevant resources (scan and load `*.xml`, `*.json`, and `*.ttl` - any that can be parsed)
- the name of a gzipped tarball (e.g. `package.tgz`) that contains the relevant resources to load

Notes:

- Packages will be installed in your [FHIR Package Cache](#) as required
- Packages must have the same underlying fhir version as that specified in the `-version` parameter
- if you want to validate against the current build version (pre-publication) of an implementation guide auto-published through [build.fhir.org](#), use 'current' as the version
- if you want to validate against an implementation that you built yourself using the IG publisher on your own machine, use 'dev' as the version
- if you have problems with an implementation guide, please ask on [\[the conformance stream on chat.fhir.org\]](#)
- you can load more than one version of an implementation guide, though this is not usually very useful

```
java -jar validator_cli.jar c:\temp\patient.xml -version 3.0 -ig hl7.fhir.us.core#1.0.1 -ig hl7.fhir.us.core#1.0.1
```

What to validate against

Use the `-profile` parameter to tell the validator what to validate against:

```
java -jar validator_cli.jar c:\temp\patient.xml -version 3.0 -ig hl7.fhir.us.core#1.0.1 -profile http://hl7.org/fhir/us/core/StructureDefinition/us-core-patient
```

The `-profile` parameter is the canonical URL for the profile you wish to validate against. This is usually clearly specified on the page where the profile is published. If the profile you specify has not been loaded through one of the implementation guides, the validator will try and load it directly from the canonical url, but it's better to load it with an `-ig` parameter first.

You can nominate more than one profile to validate against

```
java -jar validator_cli.jar c:\temp\patient.xml -version 3.0 -ig hl7.fhir.us.core#1.0.1 -profile http://hl7.org/fhir/us/core/StructureDefinition/us-core-patient -profile http://example.org/fhir/StructureDefinition/example
```

If the Implementation Guide specifies a global profile that applies to all uses of a conformance resource, then you can nominate the canonical URL of the implementation guide resource itself:

```
java -jar validator_cli.jar c:\temp\patient.xml -version 3.0 -ig hl7.fhir.us.core#1.0.1 -profile http://hl7.org/fhir/us/core
```

If the implementation guide doesn't specify a global profile for the relevant type, you'll get an error.

Validating a single resource in a bundle

You might want to validate a single resource in a bundle without validating the entire bundle. You can't use the `-profile` parameter because that's the profile for the bundle itself. It's possible to write a bundle profile that invokes the right profile for the right resource, but this is very tedious. Instead, you can just ask the validator to validate a particular resource in the bundle against a given profile:

```
-bundle {entry rule} {profile url}
```

This invokes the nominated profile (by canonical URL) on any entry in any bundle validated that meets the entry rule. The entry rule is either a Resource name, a integer index, or both:

- Patient - validate any patient against the nominated profile
- 1 - validate the 1th resource (actually the second - index is 0 based) against the nominated profile
- Patient:0 - validate the first patient resource against the nominated profile

E.g:

```
java -jar validator_cli.jar c:\temp\bundle.xml -version 3.0 -ig hl7.fhir.us.core#1.0.1 -bundle Patient:0 http://hl7.org/fhir/us/core/StructureDefinition/us-core-patient
```

Validating References

In many cases, profiles make rules about the content of references. E.g. the subject of an Observation must be a Patient, and it must be a patient that conforms to a particular patient. In order to test rules like this, the validator must be able to fetch the target of a reference, and test whether the target conforms to the profile.

Fetching .. todo...

assumeValidRestReferences

If the validator can't fetch target resources, it can at least be instructed validate that the target resource is correct. For instance, if the allowed target types are (Patient or RelatedPerson), and the reference is Group/12345, then this would be usually be an error. However this is not actually explicitly stated in the FHIR specification - a reference could be to <http://some.server/somewhere> - it doesn't need to look like a valid FHIR RESTful reference. In fact, it's not an error even if it happens to. So by default, the validator can assume nothing about the type of resource from the content of the reference itself. However most implementers do follow those rules - a reference to either Group/12345 or <http://some-server/somewhere/Group/12345> is definitely to a Group resource. (and it's definitely recommended to follow these rules).

The flag `assumeValidRestReferences` instructs the validator to use the type found in references that look like valid RESTful URLs when validating the type of the reference.

Other Validation Parameters

There are other validation parameters that affect validation:

Locale

Specifies the locale/language of the validation result messages

```
java -jar validator_cli.jar c:\temp\patient.xml -version 3.0 -locale de
```

Locales can only be used if translations have been provided. See <https://github.com/hapifhir/org.hl7.fhir.core/tree/master/org.hl7.fhir.validation/src/main/resources> for which translations are available. See also [How to add translations to the FHIR Validator](#)

Terminology Server

The validation engine uses a terminology server to validate codes from large external terminologies such as SNOMED CT, LOINC, RxNorm, etc. By default, the terminology server used is tx.fhir.org, which supports most of these terminologies. If you want to use another terminology server, you can specify one using the `-tx` parameter:

```
java -jar validator_cli.jar c:\temp\patient.xml -version 3.0 -tx http://myserver/r3
```

If you nominate another terminology server, the following rules apply:

- it must support the same version as the `-version` parameter
- it must implement the `$validate-code` operation in the FHIR specification
- there must be no authentication
- There are multiple open source servers that support these requirements. The software that runs tx.fhir.org is available from [\[Health Intersections\]](#)

Alternatively, you can run without any terminology support:

```
java -jar validator_cli.jar c:\temp\patient.xml -version 3.0 -tx n/a
```

External codes are not validated when run like this.

SNOMED CT

You can specify which edition of SNOMED CT for the terminology server to use when doing SNOMED CT Validation using the `-sct` parameter

`-sct intl`

The valid choices are:

intl	International edition (900000000000207008)
us	US edition (731000124108)
uk	United Kingdom Edition (999000041000000102)
es	Spanish Language Edition (449081005)
nl	Netherlands Edition (11000146104)
ca	Canadian Edition (20611000087101)
dk	Danish Edition (554471000005108)
se	Swedish Edition (45991000052106)
au	Australian Edition (32506021000036107)
be	Belgium Edition (11000172109)

Notes:

- editions can only be supported if they are loaded / configured on the terminology server. (To add to this list, or ask for additional editions to be loaded on tx.fhir.org, ask the [terminology stream on chat.fhir.org](#).)
- If you're loading implementation guides, and validating against them, and they specify value sets that bind to particular editions of SNOMED CT, the edition specified in this parameter will be ignored for those valuesets.

MustSupport

In some cases (e.g. when creating examples for implementation guides or when checking for potential interoperability issues with a new communication partner), it can be useful to know when data elements are present in an instance when those elements are not "mustSupport" in the profile(s) the instance is being validated against. Identifying situations where this occurs might drive a change to the profile or cause a designer to drop an element from the instance. In other cases, the presence of the element can be fine and the information message ignored.

To get the validator to flag such issues, invoke it with the parameter `-hintAboutNonMustSupport`

```
java -jar validator_cli.jar c:\temp\patient.xml -version 3.0 -hintAboutNonMustSupport
```

Extensions

note: this parameter is not presently supported

The `-extensions` parameter controls how extensions are validated by the validator. By default, unknown extensions are prohibited.

```
java -jar validator_cli.jar c:\temp\patient.xml -version 3.0 -extension *
```

This allows all unknown extensions

```
java -jar validator_cli.jar c:\temp\patient.xml -version 3.0 -extension http://example/org/
```

This allows extensions from the specified domain (by matching the URL for the extension). This parameter can repeat any number of times

Questionnaires

note: this parameter is not presently supported

By default, the validator will validate a `QuestionnaireResponse` resources against the specified questionnaire resource, if one is specified. The behavior of this can be controlled by the `-questionnaire` parameter:

```
java -jar validator_cli.jar c:\temp\patient.xml -version 3.0 -questionnaire none
```

Possible values for this parameter:

- none: do not validate questionnaire responses against the matching questionnaire
- check: validate questionnaire responses against the matching questionnaire, if one is provided
- required: validate questionnaire responses against the matching questionnaire, and report it as an error if none is specified

Extensible binding warnings

When the validator encounters a code that is not part of an extensible binding, it adds a warning to suggest that the code be reviewed

```
java -jar validator_cli.jar [src] -no-extensible-binding-warnings
```

The validator can't determine whether the meaning of the code makes it an inappropriate extension, or not; this requires human review. Hence, the warning. But the code may be valid - that's why extensible is defined - so in some operational uses of the validator, it is appropriate to turn these warnings off

Level

Set the minimum level for validation messages

```
java -jar validator_cli.jar [src] -level warnings
```

Possible values: `- hints` - report all hints, warnings and errors. (same as if not present) `- warnings` - report all warnings and errors, but not hints `- errors` - report all errors, but not warnings and hints

note: this parameter is not presently supported

Best Practices

note: this parameter is not presently supported

There are a few constraints in the specification that are warnings but marked as 'best practice'. These are typically rules that the committees believe that should be followed, but cannot be due to legacy data constraints. by specifying the parameter `-best-practice`, these will be treated as errors not warnings. This parameter has no value

Language / Display

note: these parameters re not presently supported

The `-lang` parameter tells the validator what language to default to if content has no specified language. The value is the same as for `xml:lang`. This is most useful when checking displays on coded values.

The `-coding-display` parameter controls to what degree code displays are checked. Possible value are `Ignore`, `Check`, `CheckCaseAndSpace`, `CheckCase`, `CheckSpace`

Native Validation

note: this parameter is not presently supported

By default, the validation engine only validates using the FHIR structures and profiles. The publication processes also generate a variety of xml, json and RDF schemas. You can ask the validator to validate against these as well using the native parameter:

```
java -jar validator_cli.jar c:\temp\patient.xml -version 3.0 -native
```

Note that there is nothing in these schemas that is not validated directly by the engine itself anyway, so the main use for this is to see the kind of errors that would be reported from these schemas by other software.

Engines:

- xml: Xerces
- json: tba
- rdf: tba

Logging

To help with debugging validation problems, you can ask the validator to produce a log of terminology queries. You may be asked to turn this log on and share it if you report issues with the terminology validation

-txLog filename

```
java -jar validator_cli.jar c:\temp\patient.xml -version 3.0 -txLog c:\temp\txlog.txt
```

SecurityOnly

(this is not yet supported)

Crumb Trails

When crumb-trails is set, the validator will create hints against the resources to explain which profiles it has validate the resource against, and why

-crumb-trails

```
java -jar validator_cli.jar c:\temp\patient.xml -version 3.0 -crumb-trails
```

Show Validator Internal Times

When show-times is set, the validator will produce a line in the output summarising how long some internal processes took

-show-times

```
java -jar validator_cli.jar c:\temp\patient.xml -version 3.0 -show-times
```

Validating CDA Documents

The FHIR Validator can validate CDA documents.

```
java -jar validator_cli.jar c:\temp\cda.xml -ig hl7.fhir.cda
```

This is not yet supported

Other functionality provided by the Validator

The validator also provides additional functionality beyond just validation. This section documents those various functions

Version Conversion

The validator can convert between versions of FHIR (r2, r3, and r4). To use the validator to convert versions, provide the following parameters:

```
java -jar validator_cli.jar c:\temp\observation.xml -version 3.0 -to-version 4.0 -output c:\temp\observation4.json
```

The key parameter is "-to-version" which causes the validator to invoke the version conversion routine.

Technical notes:

- the validator can use either it's own internal routines, or the structure maps found at <https://github.com/FHIR/packages/tree/master/intversion>.

- By default, the internal routines will be used for resources with a canonical URL (e.g. code system etc) and the structure maps will be used otherwise
- If the internal routines fail, the structure maps will be used anyway
- you can use the parameter `-do-native` to get the validator to try the internal routines first for any resource, and the parameter ~~`-no-native`~~ to tell it not to try them at all
- Issues with the structure maps should be discussed on the chat.fhir.org implementers channel, or submitted as PRs against the github repo above