FHIR Tooling Ecosystem

This page describes the FHIR tooling ecosystem and the vision for its future development, including the FHIR community's collective view of what types of tooling are needed, how those tools might best interact with each other and how the tooling community itself will interact and collaborate to maximize the benefit of its collective investment. It also serves as an entry point for those who are new to the FHIR tooling community and are interested in what tooling is available and/or how they might best contribute to the community.

We recognize that developers of tools will do so for their own reasons and to meet their own needs. This document has no binding force on anyone who wishes to go forth and build FHIR tooling. However, by working together with the community and by adhering to the principles and processes described here, developers are more likely to create tools that work well together and provide a better experience to their user community.

This is a living document. It will evolve as the needs and vision of the community change.

Key sections:

- #The Community - Who the tooling community is and how to get involved (including getting committer access)
- #The Architecture - Points to consider in the design of FHIR support tools including how integration can occur
- #The Tooling Environment - a list of the tooling areas, what the needs are, what exists and how the areas interrelate

- The community
  - Who and what is the FHIR tooling ecosystem?
  - How do I engage with the FHIR tooling community?
  - Being a committer
- The architecture
  - Key requirements
  - Integration Mechanisms
    - Shared files
    - RESTful operation invocation
    - Library integration
    - CDS Hooks
- The Tooling Environments
  - Design Tooling
    - Relevant Specification References
    - Features
    - Potential Tool Relationships
    - Existing Implementations
    - Wish List
  - Open Issues
  - Conformance Publication Tooling
    - Relevant Specification References
    - Features
    - Potential Tool Relationships
    - Existing Implementations
    - Wish List
    - Open Issues
  - Reference Implementations
    - Relevant Specification References
    - Features
    - Potential Tool Relationships
    - Existing Implementations
    - Wish List
  - Validation Tooling
    - Relevant Specification References
    - Features
    - Potential Tool Relationships
    - Existing Implementations
    - Wish List
  - Registry Tooling
    - Relevant Specification References
    - Features
    - Potential Tool Relationships
    - Existing Implementations
    - Wish List
  - Terminology Tooling
    - Relevant Specification References
    - Features
    - Potential Tool Relationships
    - Existing Implementations
    - Wish List
  - Instance Maintenance Tooling
    - Relevant Specification References
    - Features
    - Potential Tool Relationships
The community

Who and what is the FHIR tooling ecosystem?

The FHIR tooling ecosystem is the collective set of tools that support the development and maintenance of FHIR artifacts - both FHIR specifications and FHIR implementations. I.e. It is software that supports FHIR. It does not include the systems that actually use FHIR to deliver healthcare services (registries, decision support, clinical systems, etc.). Principally, the ecosystem will be those tools that make use of the Conformance resources. Many of the tools will be FHIR-specific, but some will support a variety of standards including non-HL7 standards.

It’s entirely possible for FHIR tooling to be developed that provides re-usable plug-and-play components for other features such as document authoring, order set creation, etc. Such tools are not covered here.

The set of tools within this ecosystem are and will continue to be developed by a variety of stakeholders including HL7 international and its volunteers, HL7 affiliates, other standards development organizations (SDOs), researchers and the implementer community. HL7 does not place restrictions on who can develop FHIR-related tooling, beyond the [licensing constraints on the use of the FHIR name and logo]. Similarly, the concepts expressed here are not an exhaustive list of what tools might potentially exist. New contributions and ideas are welcome.

How do I engage with the FHIR tooling community?

The FHIR tooling community, like the FHIR implementer community, is global. It lives both within the standards space (principally under HL7, though also under other SDOs) as well as within the FHIR implementer community as represented by the FHIR Foundation.

Within HL7 tooling issues are primarily dealt with by the [FHIR Infrastructure] (FHIR-I) work group. Oversight of tooling related to the development and publication process is provided by the [FHIR Management Group] with the support of the HL7 Publishing work group and overall oversight by the Technical Steering Committee and HL7 CIO. Work happens both at [HL7 Working Group Meetings] as well as on [FHIR-I conference calls].

Within the implementation community, discussion and coordination primarily happens on the [chat.fhir.org] discussion space. Discussions about general FHIR tooling issues, tooling integration issues and even updates to this page, happen on the [tooling] stream. However, conversations about more specific types of tooling also happen on other streams including:

- [conformance] (profiling and IG tooling)
- [crucible] and [touchstone] testing tools
- [terminology] terminology services tools
- various reference implementations (dotnet), [hapi],[javascript], [python]

New participants are welcome, but are encouraged to read the [community expectations]. Participants do not need to be HL7 members, though membership is encouraged to better support the FHIR community as a whole.

Being a committer
Tooling projects sponsored by HL7 will typically have their source managed under HL7's SVN or GIT environments. Read-only access is available to everyone. Committer access can be received by emailing lmckenzie@gevityinc.com. To gain committer privileges, send an email to lmckenzie@gevityinc.com stating the following:

- That you've read and agree to abide by chapter 16 of HL7's [and Operations Manual] (GOM)
  - The chapter deals with IP rights, patents, etc.
- That you are signed up for and agree to actively monitor the [Committer's stream] on chat.fhir.org
  - Specifically that you will abide by freezes or other constraints on committed content and that you'll monitor commits to ensure they don't break anything
- What SVN or Git user id should be granted the access

The architecture

Key requirements

The following represent ideals we encourage FHIR tools to meet. Aligning with these requirements will maximize community benefit for the tool

- **Modular**: Tools should be designed in such a manner that their features can be leveraged by tools developed by others and also to be able to leverage tools created by others. Ideally, the user of the tool should be able to configure which of the available implementations of a particular piece of functionality will be used.
- **Standardized interfaces**: To allow tools to integrate, standard interfaces and integration points are needed. One of the main purposes of this document is to identify and define such interfaces.
- **FHIR artifacts as interface**: Tools should consume and/or produce artifacts that are conformant with the FHIR specifications. Tooling should work with the XML and JSON version of FHIR artifacts or with the .zip and .pack files produced by FHIR publishing processes. Any additional custom files or configuration files should be kept to a minimum - and ideally discussed within the tooling community to see if the needs might be met in an interoperable way through FHIR resources.
- **Extensible**: It should be possible (and ideally easy) to extend the tool to accommodate extensions that may be defined to add capability to Con artifacts being produced or manipulated.
- **Version independent**: While tools are expected to be dependent on specific versions of conformance resources, they should be able to work with other versions of the FHIR specification (including draft versions with new resources and complex data types) so long as the conformance resources defining the content is available.
- **Multiple OS support**: At a minimum, FHIR tools should run on Windows and on Macs within common Windows emulators. Ideally, tools should run on Windows, Mac and Linux without emulation.
- **Free for use**: Essential functions should not require payment. HL7 tries hard to ensure that there are minimal financial barriers to using the FHIR specification. While open-source is encouraged, other financial models such as freemium, "pay what you can", "paid support" are welcome as well.
- **Escrowed source**: Non-open source tooling implementers are encouraged to keep source in escrow to be made available to the FHIR foundation should the authoring organization no longer be able/willing to maintain the tooling going forward
- **Multi-language**: Tools should be written in such a way that their user interfaces can support labels and instructions in a variety of languages if the FHIR community chooses to provide the needed translations.

Integration Mechanisms

There are a number of ways to allow different tools to interact together:

**Shared files**

Tools can operate on a shared repository of conformance instances, either in a local persistence structure or in a #registry_tooling registry. Files containing individual resources or resource bundles can also be passed as an argument when invoking a tool and/or returned as the result when invoking a tool.

**RESTful operation invocation**

Server-hosted tools can have their behaviors invoked by calling operations defined on those servers.

**Library integration**

One tool can be packaged inside of another tool and have its functions directly invoked by library calls

**CDS Hooks**

todo: Grahame or Josh to explain how this would work

The Tooling Environments

FHIR tooling requirements (those identified to date at least) can be organized into a set of sub-ecosystems:

- #Design Tooling
Design Tooling

These tools support the creation and editing of the various conformance artifacts used to define both the FHIR specification itself as well as implementation guides and other packages of conformance artifacts. Without this tooling, artifacts must be hand-edited using one of the FHIR syntaxes which requires a much steeper learning curve and is more error prone. It is unlikely that a single tool will support all resources.

Relevant Specification References

All of these artifacts should (eventually) have authoring and editing tools. They are grouped into related areas:

Implementation Guides:

- [ImplementationGuide]
- [StructureDefinition]
- [CapabilityStatement]
- [SearchParameter]
- [MessageDefinition]
- [OperationDefinition]
- [CompartmentDefinition]
- [GraphDefinition]
- [WorkflowExample]

Terminology:

- [CodeSystem]
- [ValueSet]
- [ExpansionProfile]
- [ConceptMap]
- [NamingSystem]

Others:

- [StructureMap]
- [TestScript]

In addition, the [Binary] resource may be used to convey images, page text and other content for inclusion in the publication.

Other areas of interest include:

- [Conformance Rules]
- [Compartment Definition]
- [Extensibility]
- [Operations]
- [Messaging]
- [Profiling FHIR]

For comparing resources, the following operations may be helpful:

- [CapabilityStatement/$conforms]

We may introduce “compare” operations on resources such as StructureDefinition, OperationDefinition, CapabilityStatement, etc. We might also introduce a “publish” operation on ImplementationGuide.

Features
Must have:
- load resources for editing and serialize resources once created/updated
- render resources for the users doing editing/creation

Nice to have:
- enforce design rules (both those in the core specification as well as in referenced/enforced conformance artifacts)
- look up other artifacts for reference
- provide general editing capabilities such as undo/redo; copy & paste, etc.
- use WYSIWYG editing of markdown elements
- show changes between two different artifacts or versions of an artifact
- allow managing changes between versions of resources as PATCH instances and be able store changes as PATCH instances

Potential Tool Relationships
- Invoke other design tools such that when encountering a needed missing artifact, it's possible to jump to a new tool to create the needed artifact, passing back a reference. (E.g. While editing a structure definition, have the ability to go create a value set, then come back and automatically fill in the value set reference.) Also used to view/edit artifacts already referenced by the artifact being displayed.
- Invoke `#Conformance Publication Tooling` to see rendered views of artifacts being edited
- Invoke `#Registry Tooling` to locate resources for use
- Invoke `#Validation Tooling` to confirm that instances are valid against the base resources and any declared/enforced profiles
- Invoke `#Instance Maintenance Tooling` to create or modify instances based on changes to underlying conformance resources
- Go to `#Collaborative Review Tooling` to look at comments submitted for an artifact or to note changes made
- Invoke `#Support Services` to validate or edit XPath, FHIRPath, XHTML, Markdown, etc.
- Invoke `#Support Services` for specialized behavior such as snapshot generation for StructureDefinitions
-Invoke `#Terminology Tooling` for validating or editing XPath, FHIRPath, etc.

Existing Implementations
todo - links
- Forge - .NET (mac emulator friendly) tool that supports creation and editing of StructureDefinition (profile, logical model) and ImplementationGuide
- Lantana's Trifolia - supports creation and editing of StructureDefinition (profile), ??? others?
- Health Intersection's FHIR Toolkit supports ValueSet, CodeSystem and CapabilityStatement
- Open Mapping Software supports StructureMap
- MITRE's Clinical Information Modeling and Profiling Language (CIMPL) for creating Detailed Clinical Logical Information Models (DCLIMs) and automatically producing FHIR profiles (StructureDefinition) CIMPL
- FhirpathTester - .NET tool for testing fhirpath expressions on resource instances fhirpath source (supports DSTU2 and STU3 concurrently)
- LinkEHR - Supports the definition of FHIR StructureDefinitions and exports them as FHIR profiles or ADL archetypes

Wish List
Eventually, we need good quality editors for all of the Conformance artifacts. However, CapabilityStatement, OperationDefinition, SearchParameter and TestScript are probably the most urgent.

Open Issues
- We need to decide how we want to handle "pages" adhoc narrative for security, use-cases, etc. Should these be Binaries? A profile on Basic? A proper resource?

Conformance Publication Tooling
This tooling takes a packaged set of conformance artifacts and renders them into a form suitable for implementers, analysts and others to navigate, read and understand the expectations of a particular implementation environment. Typically this is done as HTML, but PDF, wiki and other representations are also possible. In theory, this can include both tooling used to define the core FHIR specification as well as implementation guides. However, the core specific tooling is highly specialized a

Relevant Specification References
The most important resource is the [ImplementationGuide] as it defines the organization of all the content to be published.
- [StructureDefinition]
- [CapabilityStatement]
- [SearchParameter]
- [MessageDefinition]
- [OperationDefinition]
- [CompartmentDefinition]
Terminology:
- [CodeSystem]
- [ValueSet]
- [ExpansionProfile]
- [ConceptMap]
- [NamingSystem]

Others:
- [StructureMap]
- [TestScript]

In addition, the [Binary] resource may be used to convey images, page text and other content for inclusion in the publication.

**Features**

**Must have:**
- Be able to interpret the ImplementationGuide resource to find the resources to package and publish - as per the organization defined by the ImplementationGuide
- Generate a human-readable view of the package of resources
- Package up the generated publication for sharing

**Nice to have:**
- Support to publish all types of conformance resources
- Include link navigation between specification elements and from specification elements to the relevant portions of the FHIR specification.
- Validate that content is valid (valid examples, profiles, no broken links, etc.)
- Support automatic publication as content is edited
- Support publishing in alternative languages (if the necessary "translation" extensions are present)
- Support searching of the content from within the specification interface

**Potential Tool Relationships**
- Invoke [Registry Tooling] to retrieve referenced resources
- Invoke [Terminology Tooling] to expand value sets for publication
- Invoke [Support Services] to perform snapshot generation and/or narrative generation as part of publication
- Invoke [Validation Tooling] to confirm that interlinked artifacts, including examples, are valid at time of publication
- Push published content to [Registry Tooling] for publication/sharing
- As part of the publication, provide integrated links to [Collaborative Review Tooling] to support gathering feedback about published artifacts.

**Existing Implementations**
- Forge
- Trifolia?
- HL7 IG publisher

**Wish List**
- Improved renderings for CapabilityStatement, OperationDefinition, CodeSystem, etc.
- Renderings of any kind for MessageDefinition, GraphDefinition, NamingSystem
- Additional rendering widgets that make artifacts easier to understand/navigate
- Integrate publications with collaborative review tooling

**Open Issues**
- It would be nice to be able to share code that renders artifacts across publication tools - how could we do this?
- Need to agree on how much configuration should live inside the ImplementationGuide, how much should be external (but possibly shareable?) and how much should be tool-specific.

**Reference Implementations**

Reference implementations are sets of generated code that support implementation of FHIR instances in a particular programming language. These reference implementations provide a starting point for developers that allows them to avoid deciding how to represent FHIR objects within their environment.

**Relevant Specification References**

The key concepts needed to produce object or other language-specific representations of resources are found here:
The syntaxes to parse and serialize are found here:

- [XML](http://hl7.org/fhir/xml.html)
- [JSON](http://hl7.org/fhir/json.html)
- [RDF](http://hl7.org/fhir/rdf.html)

Most reference implementations also provide build in support for "fixed" terminology bindings, leveraging information found here:

- [CodeSystem]
- [ValueSet]

Reference implementations may also provide assistance with dynamic behavior, in which case, these links are also relevant:

- [CapabilityStatement]
- [ImplementationGuide]
- [SearchParameter]
- [OperationDefinition]
- [CompartmentDefinition]
- [GraphDefinition]
- [RESTful API]
- [Search]
- [GraphQL]
- [Operations]
- [Messaging]
- [Documents]

**Features**

Must have:

- Provide a language-appropriate representation of all resource elements
- Can parse and serialize instances in at least XML and JSON

Nice to have:

- Support parsing and serializing RDF
- Provide helper support for REST, operations, messaging and/or documents
- Provide helper support for search
- Be available under an open source license

**Potential Tool Relationships**

- Incorporate or allow integration with [#Terminology Tooling](#) for run-time terminology support
- Incorporate or allow integration with [#Validation Tooling](#) for run-time instance validation

**Existing Implementations**

- [.NET]
- [Java]
- [Javascript]
- [Pascal]
- [Python]
- [Swift]

**Wish List**

- Additional languages of interest to a decent percentage of implementers?
- Support for additional runtime dynamic behavior configuration based on the message definition, graph definition, operation definition and similar resources

**Validation Tooling**

This tooling allows instances of FHIR resources (including conformance resources) to be checked for validity against the FHIR resources that define them or against which they claim conformance.

**Relevant Specification References**

The two primary sources for validation information are:
In addition these resources can also provide information that is relevant to the validation process:

- [ImplementationGuide] (defines 'default' profiles)
- [GraphDefinition] (sets expectations for referenced resources)

The [Validating Resources] page covers the expectations of the validation process.

The [Resource/$validate] operation defines how validation is invoked as a service. It will return validation issues as [OperationOutcome] instances.

**Features**

Must have:

- Ability to validate both XML and JSON resources of any type
- Check structural and syntax validity of the instance

Nice to have:

- Perform validation of terminology against specified bindings
- Check FHIRPath constraints
- Check references to external artifacts

**Potential Tool Relationships**

- Invoke #Registry Tooling to retrieve referenced profiles, value sets and other resources
- Invoke or incorporate #Terminology Tooling to check for code validity

**Existing Implementations**

- [HL7 Java validator jar]
- [.NET Validation UI]

**Wish List**

- Continue to build the test case library of both valid and invalid cases
- Create validation libraries in additional languages
- Refactor validator to provide multi-lingual support

**Registry Tooling**

Registries host conformance resources as publicly accessible websites that can be searched and navigated to find relevant resources. Registries allow implementers to leverage existing artifacts rather than creating their own. They also provide a means of ensuring that implementers get the most "current" version of a particular artifact.

**Relevant Specification References**

The most critical resources to register are these:

- [ImplementationGuide]
- [StructureDefinition]

Specialized registries might support only terminology artifacts:

- [CodeSystem]
- [ValueSet]
- [ExpansionProfile]
- [ConceptMap]
- [NamingSystem]

Ideally, the registries would support all the remaining conformance resources as well:

- [CapabilityStatement]
- [SearchParameter]
- [MessageDefinition]
- [OperationDefinition]
- [CompartmentDefinition]
- [GraphDefinition]
- [WorkflowExample]
- [StructureMap]
- [TestScript]
[Binary] can be used to store images and other elements that are part of ImplementationGuides. As well, example instances will also need to be stored - and they can be of any type of [resource] at all.

The FHIR interface supported by the interface is defined in the [REST] and [Search] portions of the specification.

**Features**

Must have:

- A web-based user interface for searching for and viewing all supported resources with links that support downloading results
- A FHIR interface that supports submission, update and retrieval of resources
- User security to ensure that responsibility for submissions can be tracked and access can be managed

Nice to have:

- Ability to navigate to related resources
- Ability to download a resource and its dependencies
- Ability to allow users to assert "usage" of resources
- Ability to apply PATCH instances to existing resources (particularly for large resources such as CodeSystem)

**Potential Tool Relationships**

- Registries might leverage #Conformance Publication Tooling to render the artifacts
- Registries should support integration with the #Collaborative Review Tooling to enable commenting on registered artifacts

**Existing Implementations**

- Furore's Simplifier.NET

**Wish List**

- Enhanced support for tracking "usage" of resources
- Support for retrieving dependencies

**Terminology Tooling**

FHIR standards have heavy reliance on terminologies, including complex terminologies such as SNOMED-CT. Terminology services handle the heavy lifting of validating codes, translating codes and determining what codes are permitted in a particular space.

**Relevant Specification References**

These tools make use of the following resources:

- [CodeSystem]
- [ValueSet]
- [ExpansionProfile]
- [ConceptMap]
- [NamingSystem]

Background areas and operations include:

- [Terminology]
- [Terminology Services]
- [CodeSystem/$lookup]
- [CodeSystem/$subsumes]
- [ConceptMap/$translate]
- [ConceptMap/$closure]
- [ValueSet/$expand]
- [ValueSet$/validate-code]

A new operation is likely to be added to NamingSystem to support finding the canonical system for a given identifier or code 'system' value.

**Features**

The features of a terminology service are defined [in the FHIR spec]

**Potential Tool Relationships**

- Terminology services will typically either incorporate or depend on #Registry Tooling to look up the terminology artifacts needed to execute their operations.
Existing Implementations

todo: Add links

- Health Intersections
- Apelion
- Art-Decor
- VSAC
- SNQuery

Wish List

- Add support for NamingSystem resolution

Instance Maintenance Tooling

Implementers are highly dependent on instance examples to understand the specification and to act as templates when creating their own solutions. Large volumes of sample instances (sometimes with specific characteristics) are often also needed in order to test implementations.

Relevant Specification References

- [StructureDefinition]
- [MessageDefinition]
- [GraphDefinition]
- [ValueSet]

Features

Must have:

- Ability to author FHIR instances of any type with a "friendly" user interface
- Ensure instances are valid against the core specification

Nice to have:

- Support authoring against various versions of the specification, including "current build"
- Generate 'template' resources based on the specification and example values specified in the specification
- Check validity of terminology and resource references
- Support lookup of allowed terminology values
- Support lookup of relevant identifier systems
- Generate bulk data that is "clinically relevant"

Potential Tool Relationships

- #Registry Tooling can be used to retrieve instances to edit or copy and to store new or revised instance examples
- #Validation Tooling can be used to help ensure that instances are valid
- #Terminology Tooling
- #Reference Implementations can be used to convert instance examples from one FHIR syntax into the others

Existing Implementations

Individual instance creation

- ClinFHIR

Bulk instance creation

- mihin (Jeff Eastman)
- Synthea - Mitre (https://github.com/synthetichealth/synthea)

Wish List

- ???

Implementation Test Tooling

Part of standards development is the ability to test (and potentially certify) that implementations are compliant with the standard. Test tools support the automation (or at least guidance and monitoring) of the testing process and reporting the results.
Relevant Specification References
These tools make use of [TestScript] and will produce [TestReport]
The testing process is described in [Testing]

Features
Must have:
- Can consume TestScript resources
- Can execute TestScripts

Nice to have:
- Can act as a "man in the middle" over secure communications
- Can support testing client systems by monitoring outgoing and incoming communications while someone manipulates and records the behavior of the user interface

Potential Tool Relationships
- Retrieve TestScripts, conformance artifacts and sample data from #Registry Tooling and potentially store TestReports in the registry as well
- Validate results using #Validation Tooling
- Integrate with #Design Tooling to allow scripts and sample data to be adjusted during the testing process

Existing Implementations
- TouchStone
- Crucible

Wish List
???

Conversion Tooling
FHIR implementation often involves converting to or from other syntaxes including HL7 v2, CDA, OpenEHR as well as proprietary formats. It may also be necessary to convert between different versions of FHIR or between logical model instances and conformant FHIR instances. Conversion tooling supports conversion message instances between these different formats. Note that conversion tooling may be used to convert between syntaxes where neither the source nor the target is FHIR.

Relevant Specification References
Mappings are defined using:
- [StructureMap] for structures
- [ConceptMap] for terminologies

Structure map makes use of a FHIR-defined [Mapping Language] which in turn relies on [FHIRPath] which is a syntax-neutral path language.

The transformation process can be initiated RESTfully using the [StructureMap/$transform] operation

Features
Must have:
- Ability to consume structure maps
- Ability to execute those transforms against instances

Nice to have:
- ???

Potential Tool Relationships
- #Registry Tooling can be used to retrieve referenced StructureMap and ConceptMaps
- #Terminology Tooling can be used to support code translation as part of instance translation

Existing Implementations
Grahame's tool (todo: name)
Open Mapping Software FHIR Transform Engine
LinkEHR Studio: Define mappings between different clinical models (from potentially different reference models such as HL7 FHIR, HL7 CDA, openEHR, ISO13606, CDISC ODM, etc.) and generate automatically transformation programs

Wish List

Collaborative Review Tooling

A key part of developing standards and derived specifications is soliciting feedback on the specifications as well as on proposed changes to those specifications. These reviews might be from a technical perspective, a clinical perspective or other perspectives. This tooling supports that process.

Relevant Specification References

None - as yet

Features

Must have:

• Allow users to provide feedback about specific elements within a specification, including identifying that feedback as positive or negative.
• Allow users to update feedback already provided
• Provide a consolidated view of feedback associated with an artifact

Nice to have:

• Support discussion about feedback points
• Integrate with source control so that changes applied can be noted as being associated with particular feedback
• Provide control over who's invited/permited to provide feedback and monitor who's done so yet

Potential Tool Relationships

• Be able to link from provided feedback back to the published version produced by the #Conformance Publication Tooling

Existing Implementations

• HL7 ballot site
• OpenEHR tool suite?

Wish List

• Integration of feedback tooling directly into the HL7 core publication and IG publications

Open Issues

• Do we need a resource to capture this feedback or do we leverage Observation or QuestionnaireResponse?

Support Services

Support services are libraries or utilities that aren't intended to be used stand-alone but which provide features that are useful for other tools and which are sufficiently complex enough that it makes sense for tools to re-use the capability rather than developing the capability themselves.

Relevant Specification References

• [Narrative]
• [FHIRPath]

Features

FHIRPath validation and evaluation: This is used to both check that FHIRPaths are 'valid' against the StructureDefinitions they're intended to apply to as well as to assert FHIRPaths against FHIR (and potentially other) instances.

Narrative generation: This generates the XHTML content for the "text" narrative for a resource providing a human-friendly review of the resource content

Snapshot generation: This takes the "differential" present in a StructureDefinition and generates the "snapshot" view applying the changes from the "differential" to base StructureDefinition's snapshot (and potentially the snapshot of referenced StructureDefinitions)
Potential Tool Relationships

- FHIRPath validation may need to load referenced StructureDefinitions and terminology artifacts from #Registry Tooling.
- FHIRPath evaluation may depend on #Terminology Tooling for terminology-related operations
- Narrative generation may depend on #Terminology Tooling for terminology-related to look up display names
- FHIRPath validation may need to load referenced StructureDefinitions from #Registry Tooling.

Existing Implementations

FHIRPath

- Java, .NET, ???

Narrative

- Java

Snapshot

- Java, .NET

Wish List

- Broader language support for existing implementations
- Improve the user friendliness and clinical utility of Narrative views (and ideally get consistent representation across tools)
- Incorporate relevant "core" extensions as part of narrative rendering
- Increased test cases for Snapshot generation
- Create a "differential-generation" feature where two snapshots can be "diffed" to identify the changes.

Open Issues

There's been discussion about defining 'subsets' of FHIRPath which might allow for