HL7® FHIR® DevDays

JUNE | 2023

Future of FHIR Conformance

Lloyd McKenzie – Dogwood Health Consulting

HL7 FHIR DevDays 2023 | Hybrid Edition, Amsterdam | June 6–9, 2023 | @HL7 | @FirelyTeam | #fhirdevdays | www.devdays.com

ORGANIZED BY firely | HL7 International

Who am I?

- Name: **Lloyd McKenzie**
- Role: Chief Standards Officer
- Company: DOGWOOD Health Consulting
- Background:
  - One of FHIR's initial editors
  - Cochair of FMG and FHIR Infrastructure
  - Heavily involved in healthcare interoperability for over 20 years
  - Unofficial FHIR community manager
  - Technical lead for SDC project

Learning Objectives

- Explain Cardinality vs. Must Support
- List some of the challenges with Must Support
- Be familiar with the new capabilities of Obligation
- Understand how Actor, Requirements, CapabilityStatement, TestPlan & TestScript come into play

Some basics

- Cardinality
  - min..max
  - “What must appear in an instance”
  - Used in validation
- Constraint
  - FHIRPath-defined rule
  - Allows co-occurrence constraints (within a resource)
  - Also used in validation

Some basics (cont'd)

- Must Support
  - Defines system behavior
    - must store/display/process/allow editing/etc.
  - What sender must do before sending
  - What receiver must do after receiving
  - NO impact on validation (but might impact testing)
- Why?
  - E.g. deceasedDate

Problems with mustSupport

- Meaning is contextual
  - Have to read text (anywhere)
- Meaning is the same everywhere
  - Unless overridden in a comment
- It’s binary
  - No ‘should’ support or ‘may’ support
- Interferes with profile re-useability

Enter ‘Obligation’

- Extension (for now)

• code	1..*	Coding (extensible)
• actor	0..*	canonical(ActorDefinition)
• documentation	0..1	markdown
• usage	0..*	UsageContext
• filter	0..1	string
• filterDocumentation	0..1	string
• process	0..*	uri

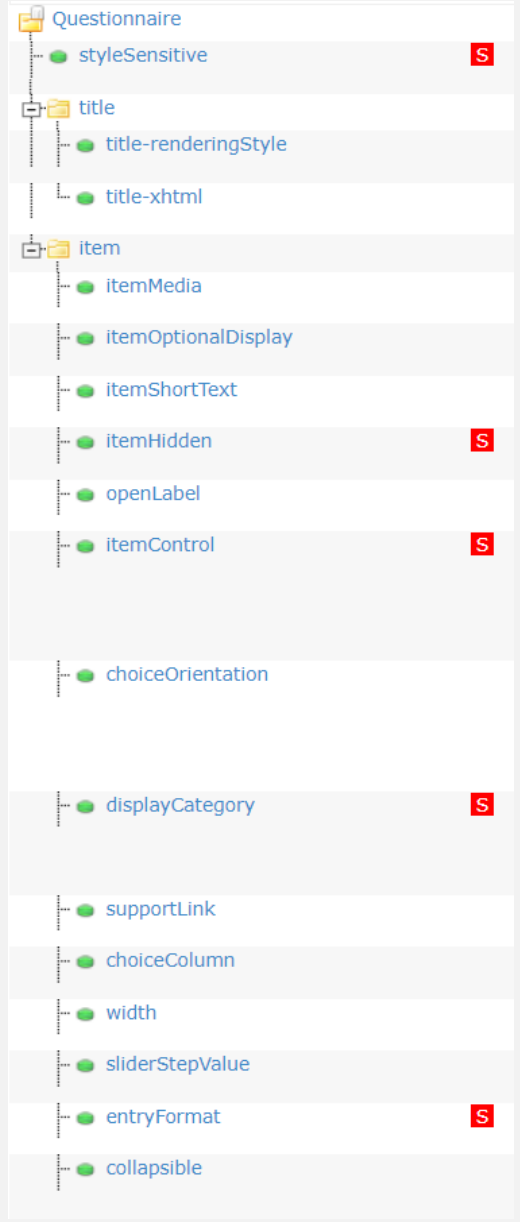
Types of obligations

- All codes can be SHALL, SHOULD or MAY (post-coordination)
- Examples
  - can-send
  - must-send
  - in-narrative
  - in-ui
  - in-store
  - unaltered
  - process

Examples from SDC

- Actors for different CapabilityStatements
  - Form Designer
    - will-send, will-present, in-ui, must-display, must-share
  - Form Filler
    - must-process
  - Form Manager
    - will-send, preserve, must-share

- Sub-Actors for different display features
  - Tables, sliders, xhtml-rendering, dynamic questions, etc.



Additional Bindings

- Like Obligation but applies to codes
- Some of the bindings also are about behavior, not validation
  - minimum – what codes SHALL be supported
  - current – new data must apply this binding (but not external/legacy)
  - ui – appropriate set of codes for display
- Some bindings are context-specific
  - Can have different required/extensible bindings for
    - different types of patients
    - different jurisdictions

Requirements resource

- Formal documentation of where requirements come from
- Supports requirements traceability
  - Can link conformance artifacts to requirements

What is an 'Actor'?

- System or Person
- Referenced by for Requirements and Obligations
- Can exist in a hierarchy
- E.g.
  - "A Table-enabled SDC Form Filler" is an "SDC Form Filler"
    - (and inherits all of the requirements/obligations from it)
- Unlike profiles, an actor can inherit from multiple other actors

CapabilityStatement2

- There are an ever-growing number of capabilities to describe
  - Search combinations, SMART launch capabilities, CDS Hook capabilities, etc.
- Some server CapabilityStatements are **huge**

- Solution:
  - Move many capabilities to codes
  - Define operations to ask “Can you do these things”
    - (rather than “Tell me everything you do)
- We need developers to try it out…

TestPlan and TestScript

- TestPlan
  - Organizes intentions for testing
  - Can link to TestScript and other scripting languages
- TestScript
  - Not new, but potential for greater support in IGs
    - Auto-generate some basic tests from CapabilityStatement and obligations

Kinks to work out

- It would be nice to:
  - re-use the same code combinations with different ‘strength’
    - E.g. SHOULD (store/display/edit) vs. SHALL (store/display/edit)
  - re-use contexts?
  - have a standard ‘icon’ for standard codes
    - and an icon property for new ones?
  - define what actors are ‘relevant’ for a profile (even if others are inherited)
  - establish a ‘default’ set of obligations for an actor’s elements, and show the differences
  - reference/display formal ‘Requirements’ in narrative pages
  - point to Actors from CapabilityStatements rather than vice versa
    - With SHALL/SHOULD/MAY?

What did you learn

- **ActorDefinition** to define people & system participants
- **Requirements** to document business, technical, other requirements
- **Obligations** to more robustly define system behaviors on data elements
- **CapabilityStatement2** to more robustly define (and discover) interface behaviors
- **TestPlan** and **TestScript** to allow sharing of tests to verify all of the above

Q&A

- lloyd@dogwoodhealthconsulting.com

- Or better yet, include the community and ask/discuss on chat.fhir.org

ORGANIZED BY