WHITEPAPER

Refining Conformance: An Analysis of the Evolution from Must Support to FHIR Obligations

By Ward Weistra, Product Lead at Firely and member of the FHIR Management Group at HL7 International. *This document is based on my presentation at the HL7 Europe Work Group Meeting on December 2, 2025.*

Contents

1. The Challenge of Conformance - The Ambiguity of "Must Support"

While standards like FHIR provide a common syntax for healthcare data exchange, true interoperability hinges on clear, unambiguous definitions of implementation requirements. For years, the primary tool for defining these requirements within a FHIR profile has been the "Must Support" flag. However, its historically broad definition created ambiguity, leaving implementers to interpret its meaning within the context of narrative documentation.

This section explores the foundation of FHIR conformance and the specific challenges posed by "Must Support" that necessitated a more granular, machine-readable solution.

1.1. The Foundation of Conformance Language

FHIR's conformance language is built upon the well-established keywords defined in RFC 2119, which provide a clear hierarchy of requirement levels:

- MUST (or SHALL): This term indicates an absolute, non-negotiable requirement of the specification.
- SHOULD: This term signifies a strong recommendation. While valid reasons may exist to deviate from a SHOULD requirement, the full implications must be understood and carefully weighed.
- MAY: This term defines an item that is truly optional, granting implementers full discretion without penalty.

1.2. Defining "Must Support" in FHIR (Pre-R4B)

Prior to the FHIR R4B release, the official specification defined the "Must Support" flag with intentional vagueness. The core specification stated that implementations "SHALL provide 'support' for the element in some meaningful way."

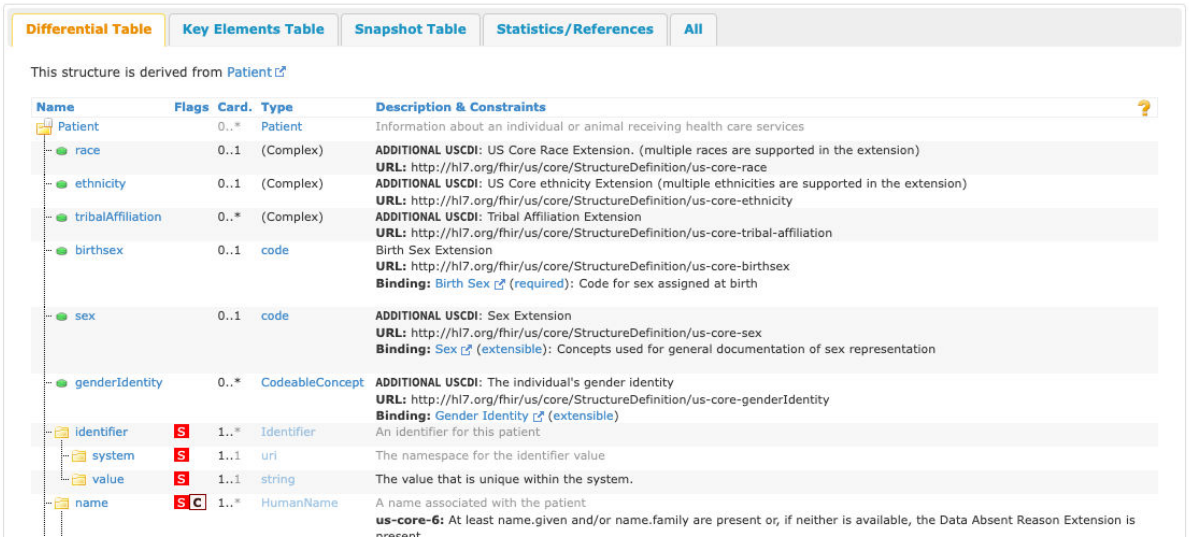
This definition was strategically designed to be flexible, but it placed the burden of clarification squarely on the authors of individual Implementation Guides (IGs). The specification explicitly required that any profile using "Must Support" must "also make clear exactly what kind of 'support' is required."

This could involve a wide range of expectations, including requirements for data storage, display, capture, inclusion in decision logic, or transmission to other systems.

The result was that the precise meaning of "Must Support" was locked away in human-readable narrative, varying from one IG to another.

1.3. Case Study: US Core Profile

The US Core Implementation Guide provides a clear example of this narrative-based approach. The US Core Patient Profile, for instance, marks several elements with the "S" flag for "Must Support."



▲ *A screenshot of the US Core Patient Profile, in which Must Support elements are marked with a red 'S' symbol.*

To understand its meaning, an implementer must consult the guide's conformance section, which provides explicit definitions that vary for different system actors. For example, US Core defines distinct requirements for a "US Core Responder" (typically a server) versus a "US Core Requester" (typically a client). When an element is marked as Must Support, a US Core Responder "shall be capable of populating all data elements as part of the query results."

This case study highlights how a single flag required detailed, actor-specific narrative to become actionable, a model that works but lacks machine-readability.

To address the inherent ambiguity and documentation overhead of this approach, the FHIR community developed FHIR Obligations—a formal, structured, and machine-readable solution for defining conformance.

2. The Solution - Introducing the FHIR Obligation Extension

The FHIR Obligation extension represents the official evolution of the "Must Support" concept. Its strategic importance lies in shifting conformance from ambiguous narrative descriptions to granular, machine-readable behavioral expectations that can be embedded directly within a FHIR profile. This allows a single profile to communicate precise, testable requirements for multiple system actors, significantly enhancing the clarity and verifiability of interoperability standards.

2.1. Core Purpose and Goals

The primary goals of the Obligation extension are to:

- **Clarify "Must Support":** Provide a precise, actionable, and coded meaning to the Must Support flag, removing ambiguity and reliance on external narrative documentation.
- **Define Actor-Specific Expectations:** Enable a single profile to communicate different requirements for various system actors (e.g., servers, clients, data producers, data consumers), thereby avoiding the need to create and maintain a multitude of similar profiles for different roles.
- **Enhance Behavioral Validation:** Provide explicit, machine-readable instructions for automated behavioral testing tools (like Inferno), making it possible to programmatically verify if a system adheres to its stated obligations. It is important to note that this does not affect standard FHIR resource validation.

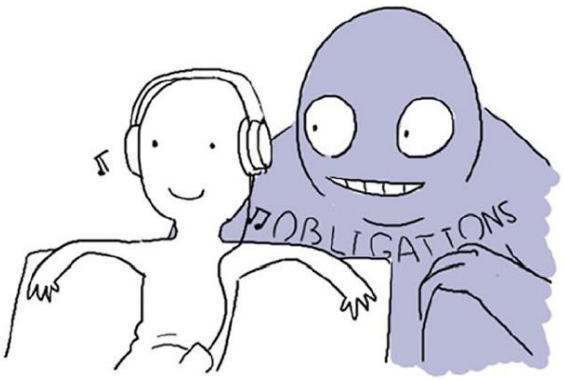
2.2. Status and Availability

The Obligation extension is part of the **FHIR Extensions Pack**, a collection of extensions that can iterate more quickly than the core FHIR specification.

Its key details are:

- **Standards Status:** Trial-use
- **Maturity Level:** 1
- **Availability:** Usable in FHIR versions STU3 and higher.

The power of this extension lies in its structured components and the specific codes used to define required behaviors, which are detailed in the following section.



3. Technical Deep Dive - Anatomy of the Obligation Extension

For architects and developers, understanding the structure of the Obligation extension is critical for effective implementation. It is not an extension on the resource data itself, but an extension on the elements within the profile's StructureDefinition.

This section deconstructs its components, contexts of use, and the standardized codes that define required behaviors.

3.1. Contexts of Use

The Obligation extension can be applied in three distinct contexts within a FHIR profile, offering significant flexibility to profile authors:

a. **On an individual element:** This is the most common use case, where an obligation is applied directly to a specific element, such as Patient.identifier.
b. **On the root of a profile:** An obligation can be placed on the root of the StructureDefinition and then targeted at a subset of elements using their element IDs. This is an efficient way to apply the same obligation to multiple elements at once.
c. **On the type of an element:** The obligation can be scoped to apply only when a choice-of-type element (e.g., deceased[x]) is of a specific data type (e.g., deceasedBoolean).

3.2. Key Structural Elements

The Obligation extension is composed of several key sub-elements that work together to define a complete behavioral requirement:

- **code**: This is the only mandatory element. It contains a code from the Obligation Codes value set that defines the specific behavioral expectation (e.g., SHALL:populate, SHOULD:display).
- **actor**: An optional element that identifies the system actor to whom the obligation applies (e.g., a data producer or consumer). This allows a profile to specify different rules for different systems. It uses an ActorDefinition, a resource type introduced in FHIR R5.
- **filter**: An optional element that uses a FHIRPath expression to narrow the scope of the obligation. For example, it could be used to specify that an obligation applies only to the first three instances of a repeating element.
- **process**: An optional element that links the obligation to a specific documented process, workflow, or OperationDefinition, providing additional implementation context.

3.3. The Obligation ValueSet

The behavior defined in the code element is drawn from a standardized value set. The codes are categorized by function and prefixed with a conformance verb (MAY, SHOULD, SHALL) to indicate the requirement level.

Category	MAY	SHOULD	SHALL
Populate		SHOULD:populate	SHALL:populate
Populate if known		SHOULD:populate-if-known	SHALL:populate-if-known
Able to populate	MAY:able-to-populate	SHOULD:able-to-populate	SHALL:able-to-populate
Narrative: include	MAY:in-narrative	SHOULD:in-narrative	SHALL:in-narrative
Narrative: exclude		SHOULD:exclude-narrative	SHALL:exclude-narrative
User input	MAY:user-input	SHOULD:user-input	SHALL:user-input
Explain (justify irrelevance or populate)		SHOULD:explain	SHALL:explain
Persist (retain/store)	MAY:persist	SHOULD:persist	SHALL:persist
Alterability	MAY:alter	SHOULD:no-alter	SHALL:no-alter
Error tolerance when present		SHOULD:no-error	SHALL:no-error
Invalid handling: reject		SHOULD:reject-invalid	SHALL:reject-invalid
Invalid handling: accept		SHOULD:accept-invalid	SHALL:accept-invalid
Display (render)	MAY:display	SHOULD:display	SHALL:display
Process (workflow/logic)	MAY:process	SHOULD:process	SHALL:process
Print (paper/PDF)	MAY:print	SHOULD:print	SHALL:print
Handle (interpret correctly)		SHOULD:handle	SHALL:handle
Ignore (deliberate no-op)	MAY:ignore	SHOULD:ignore	SHALL:ignore

▲ *A table overview of the values in the Obligation Codes ValueSet.*

Having explored the technical structure, the next section outlines the best practices for harmonizing this new extension with the long-standing Must Support flag.

4. Best Practices - Integrating Obligations and Must Support

While the Obligation extension provides granular detail, the Must Support flag remains a vital tool for signaling broad implementation expectations. Using them cohesively is essential for creating profiles that are both precise and widely compatible. Following established best practices ensures that systems can progressively adopt Obligations while maintaining backward compatibility with simpler conformance mechanisms.

4.1. The Primary Directive: Obligations Imply Must Support

The fundamental relationship between the Obligation extension and the Must Support flag is governed by a clear community consensus: **any element with an Obligation extension should also be marked as Must Support**.
The rationale for this directive is to provide a compatibility "backstop." Since the Obligation extension is new, many existing systems and tools do not yet understand it. By including the Must Support flag, profile authors ensure that these systems still recognize the element as important, even if they cannot parse the specific behavioral details. This position was reinforced by the "Not Persuasive" decision on community ticket FHIR-43615, which proposed allowing Obligations without the Must Support flag.

4.2. Profiling Guidance and Relationships

Beyond the primary directive, several related best practices help create robust and clear profiles:

- **Embrace Open Profiling:** Prioritize using Must Support over strict cardinality to define requirements. For example, marking an element as 0..1 with Must Support allows systems with legacy data (where the element may be missing) to remain conformant, while still setting a clear expectation that the data should be shared when available. Cardinality of 1..1 should be reserved for cases where receiving data that lacks the element is completely unacceptable.
- **Prioritize Modifier Elements:** Elements marked with the is-modifier flag (meaning they can fundamentally change the interpretation of the resource) are prime candidates for being marked as Must Support. If an implementer ignores a modifier element, they risk misinterpreting the entire resource, making support for these elements critical.
- **Maintain Parent/Child Cohesion:** A logical consistency should be maintained in element hierarchies. If a child element (e.g., Patient.name.family) is marked Must Support, its parent element (Patient.name) should be as well. Conversely, if a parent element is Must Support, at least one of its child elements should also be Must Support to clarify what "support" for the parent entails.
- **Document the Rationale:** As a best practice shared by Simone from HL7 Germany, it is highly recommended to add a comment to the profile explaining *why* an element was marked as Must Support. This preserves the design rationale for future profile maintainers and implementers.

These practices ensure that profiles are not only technically sound but also practical and interpretable, paving the way for wider adoption.

5. Real World Adoption and Future Outlook

While the FHIR Obligation extension is officially in a "Trial-use" phase, this status belies its rapid and significant adoption by major national and international standards bodies. This strong momentum signals that the extension effectively addresses a long-standing need within the FHIR community for more explicit and testable conformance requirements.

5.1. Early Adopters

Several prominent organizations and projects are actively implementing FHIR Obligations in their specifications, demonstrating its real-world utility:

- **Canadian FHIR Registry:** The CA Core+ project, a national initiative to harmonize Canadian core specifications, uses Obligations to define actor-specific behaviors for producers and consumers.
- **HL7 Europe:** The European Laboratory Report profiles leverage Obligations to specify requirements for different actors in the lab reporting workflow, such as the *curator*, *repository*, and *consumer*.
- Other national standards bodies in **Australia**, **Germany**, and **Austria** are also actively exploring or adopting the extension for their national implementation guides.

5.2. Conclusion and Forward Look

FHIR Obligations effectively resolve the long-standing ambiguity of the Must Support flag. By providing a granular, actor-specific, and machine-readable framework, they empower profile authors to define clear, verifiable conformance requirements directly within a StructureDefinition.

This evolution moves a critical aspect of interoperability from ambiguous narrative into structured code, enabling more robust automated testing and reducing implementation friction. As the FHIR standard matures and the demand for verifiable interoperability grows, the adoption of FHIR Obligations is expected to become a cornerstone of high-quality, implementable specifications.