## Null Requirements for all Data Types

The FHIM supports null values for cases in which a scenario may require a property that the partner cannot provide. It effectively makes a required property not required, but it establishes an expectation, without which virtually all properties would have to be optional.

* Requirement: all data types shall support the option of specifying that a property is nullable, i.e., an element instance can be present without containing a value.

This specification does not stipulate any particular representation of “null”: this is an implementation decision. HL7 V3 artifacts will use null flavors for this feature; V2 uses empty strings delimited with double quotes (""); other platforms may simply omit the property or use other conventions.

In addition to the general concept of “null”—no value provided—HL7 recognizes several “flavors of null,” which add metadata about why a value may be null. This feature uses a property inherited by every data type, and it can create a lot of overhead, both in type instantiation and value checking. We may decline to adopt this approach in most cases, as most platforms do not have similar features, preferring to model such metadata explicitly where necessary.

However, we recognize three special cases.

First, it is sometimes useful to distinguish a positive assertion that a value is unknown from the mere absence of a value. That is, in addition to the “I’m not telling you anything about this value,” we want to be able to say “I considered this question and I don’t know the answer.”

* Requirement: all data types shall support the option of specifying that a value is unknown.

Second, it is sometimes useful to distinguish the circumstance where a value is not applicable, e.g., gender-specific observations for patients of the other gender.

* Requirement: all data types shall support the option of specifying that a value is not applicable.

And third, there is a need for the concept of “other” in cases where the domain of an element is not a formally defined, infinite set of patterned values (integers, times, decimal numbers), but a finite, enumerated set of concepts—i.e., coded concepts. In these cases, it is not uncommon for users to need to express concepts not anticipated by the authors of the enumerations, where specification designers permit this latitude.

The first step in supporting these cases is to indicate, in the binding of the property to the value set, that unspecified values are allowed. This could simply be another specialization of the coded type; there is no additional instance data to record, so there are no new properties to define for the data type. Instances using “other” concepts would look just like instances using recommended concepts; the difference would only be detected when validating the coded value and finding it not present in the value set.

It may be necessary, however, to explicitly state that an instance invokes the “other” option in order to distinguish “other” instances from errors. If the instance does not assert that it is “other,” then the receiver can validate the value against the value set. If this is necessary, then we have another requirement:

* Requirement: coded data types shall support the option of specifying that a value is not from the specified value set.

Whether this is a requirement or not, we assume that the coded value would be modeled in the same properties as a value from the specified value set, and that there is no need for a parallel set of code/system/text properties to support these “extensional” values.

This leaves us with up to four values that could be considered flavors of null—no information, unknown, not applicable, and other—providing a glimpse into the thought process that resulted in the flavors of null design in the HL7 RIM. However, note that they are not all that similar.

* No Information: Null
* Unknown, Not Applicable: Null, plus metadata about why it is null
* Other: Metadata about a value that is not null.

If we support Null in the manner described above, then the additional requirement is to support metadata about a particular element, whether null or not.

There seem to be 4 general solutions to this requirement:

|  |  |  |
| --- | --- | --- |
|  | Model property metadata in the information model | Model property metadata in the data type |
| Model property metadata as individual properties | A  Include properties in the specification where metadata is required.   1. This is a generally consistent solution, though it may introduce a level of detail that some find inappropriate to a conceptual model. 2. This approach will require a decision about whether these metadata properties can be included as siblings to their target properties, with the semantic relationship evident only from the name or description, or should be broken out into classes to associate the two values. The latter approach verges on quadrant B. | B  Provide each type with properties to indicate whether any of the metatdata applies.   1. This path may create a proliferation of types, and many of the types will have to re-define properties already defined elsewhere in order to avoid multiple inheritance (unless we can do it with stereotypes). |
| Model property metadata as one coded property | C  Include a coded property in the information model to capture required metadata.   1. This approach faces the same “property or class” decision as quadrant A, and has the same options as quadrant D. | D  Include a coded property in the data types to capture metadata.   1. This is the HL7 RIM approach, though we could leave “null” out of it, as well as many other more specialized values in the HL7 value set. |

And a fifth option, E: don’t model this metadata in the FHIM; defer it to the “use case/specification” process.

## Text requirements for coded types

We anticipate N possible scenarios for recording text in addition to the text defined in the value set as the appropriate representation of the concept.

1. The original text from which code was derived.
   * This case involves the conversion of information recorded as natural language (e.g., a clinical note) into structured data, whether manually or automatically.
2. Text displayed on screen for selection of code, if different from system-specified text.
   * This is semantically very similar to case #1, the only difference being the author of the text.
3. Qualifying text for an accurate but imprecise coded value.
   * This is a property that could be useful for many elements. We might expect use case specifiers to identify in the information model, or we might adopt a style in which this dimension is always available, built into the types.
4. The value for an “other” selection.
   * “Other” values should be communicated in the normal coded value attributes, even if there is no code available for a concept entered as text.

We propose that one “originalText” value will meet requirements 1 & 2.

Requirement 3 should be addressed by an information model element, in cases where qualification is required.

Requirement 4 should be met by the code and text elements used for values from the value set.

## Other code properties

Translation: Our specification is for the specified value. Participants may include specified values, or, if they use a different system and it is permitted, they may use that system and tag it with the “other.” We have no use cases at this time that specify the use of a primary value and an additional translated value.

Composition: Value set definitions may permit or prohibit expressions. This has no impact on the model, apart from an implementation need to address field lengths.

Rank, score, value: Ordinals may need magnitudes. One property should be able to support all three of these variant uses.

Order: This seems to be a display property that can be inferred from the value set specification, and does not need to be specified in the instance.