Guide to Designing Resources

`Introduction

TODO: add hyperlinks for key terms to the relevant place in the FHIR spec for definitions/explanation

This section provides guidance in the design of resources, profiles and extensions. It does not answer technical "how to" questions, but rather provides guidance for work groups and other authors in the design decisions they make. For technical instructions on how to define resources and the meaning of various resource and extension properties, refer to FHIR Guide to Authoring Resources.

Design guidance is provided in the form of questions with associated answers. In many cases, the guidance provided is non-binding advice. However, in some cases the answers reflect formally approved methodology which is expected to either be followed or have an exception granted and may be reviewed as part of QA processes. These are clearly marked as "**Rule**". All rules and many guidelines will also have additional content that provides information on *why* the rule or guideline is in place, referencing the principles and precepts associated with FHIR.

This page is subject to ongoing change as HL7 gains experience in FHIR development and implementation. Feel free to add new questions, guidance on existing questions, etc. This page is curated and managed by the Modelling and Methodology committee who has authority for formal approval the creation and modification of "Rules". If a change is desired to a rule, please add a comment rather than changing the rule directly. The guidelines used to evaluate methodology guidelines can be found in FHIR Methodology Process.

In some cases, methodology questions may also be of interest to implementers. These topics are marked with a "**" and may be migrated into the FHIR specification directly

Todo: Move more content across from Authoring FHIR Resources

Contents

Resource Guidance

Naming Rules & Guidelines

Rules

- Names for operations, resources and resource elements ***must***:
    - be lowerCamelCase for elements, UpperCamelCase for resources, be lowercase for operations
    - be U.S. English (spelled correctly!)
    - be expressed as a noun, with a preceding adjective where necessary to clarify the semantics and to make unique
    - not make use of trade-marked terms

Guidelines

Determination of names needs to take into account the “scope” of what the resource/element/operation will be used for. I.e. What are the boundaries?

Names should:

- use the "most broadly recognized" industry term for the object that is unlikely to be mistaken for a different element.

- assume no familiarity with other HL7 standards such as v2, v3 or CDA except within the broader context of what industry is familiar with overall
- as much as possible without interfering with the intuitiveness of the name to implementers
- express the full breadth of meaning of the concept
- be sensitive to cross-national political considerations
- be expressed as "singular", particularly if the element is repeating
- use abbreviations only where extremely well understood by the entire target market.
    - Where used, treat the abbreviation as a “word” in terms of following the casing strategy. E.g. “targetUri for an element name”
- use different names for different elements within the hierarchy unless the semantic is the same in different context.
    - E.g. using “.name” for “sponsor.name” and “contact.name” is ok if the meaning of “name” is consistent for both sponsor and name.
- use names that distinguish from concepts that are not part of the resource (not considered core), but still exist in the resource space and might reasonably be confused.
- be concise (differentiation of semantics through the name is only necessary with other elements in the resource. Full semantic description is in the definitions.)
    - This includes non-redundancy with the context. I.e. Don’t repeat part of the parent in the name of a child
- be consistent with naming of similar elements in the same resource and other resources unless driven by common use of different terminology in industry.
    - E.g. “birthDate” and “dateOfDeath” would be bad
- be sufficiently different from other names in the same space (other elements in same resource, other resources for a resource, other operations for an operation) to avoid confusion in verbal communication (“update” and “updates” is bad)
- not include a suffix whose purpose is to identify the data type of the element (i.e. don’t end a name in “code” or “indicator”, etc. to identify what it is.
- where a concept is known by multiple names in different places or contexts, use aliases to convey additional names
- Names that lend themselves to being frequently abbreviated in committee use are likely too long and should be viewed with suspicion.
- Use skmtglossary.org and a thesaurus to evaluate candidate names when unsure
- If you’re having a really hard time coming up with a commonly recognized industry name for a thing, it may be a sign that the granularity of your “thing” isn’t aligned with industry needs.

Purpose is to give maximum insight for the unfamiliar developer to "what is this thing". Also used for searching. Name length impacts instance size and code readability, so there's a trade-off between semantic precision and ease-of-use.

Creating Resources

When should a new resource be created?

- See FHIR Resource Considerations

How should resources be named?

- See General naming guidelines

Creating resource elements

When does an element qualify for inclusion in core?

The "rule of thumb" on element inclusion is that elements are only included "if 80% of systems will support the element"

This has several qualifiers/caveats:

- This is not a strict calculation and no numeric value is required. Primary objective is "what will the significant majority of systems use"
- This doesn't mean 80% of instances will contain the element, only that systems will support conveying/consuming the element when it's relevant. (E.g. Date of Death is widely supported, but often unpopulated/not applicable)
- There's often a question of whether the 80% focuses on existing systems (with legacy limitations) or expected future implementations (what will systems using FHIR do). For the DSTU period, the answer is to lean towards what legacy systems do now. Normative track will examine implementation patterns of actual FHIR use and adjust accordingly.
- (see below for bad-practice legacy data elements)
- The set of systems being considered are those covered by the complete scope of the resource (human/vs. veterinary, breadth of domains, etc.)
- The "weighting" of a system may be adjusted somewhat for level of use (e.g. widely implemented systems affecting large numbers of patients will be given somewhat more credit than special purpose systems used in only one or two clinics)
- When dealing with new area in healthcare that has little (or no) implementation track-record, the 80% needs to be determined based on the best guesses of those developing the specification and will need to be confirmed by significant DSTU experience prior to going normative.
- There has been argument made that there should there be exceptions for data elements that are expected to be widely adopted based on expected early implementation communities (e.g. Patient.picture). Recommendation is that the "what exists in systems now" rule be followed, handling these as extensions and leaving it to the DSTU implementation experience to guide what should be migrated to Core.
- In the event that elements identified as Core based on existing usage are found not to be widely used in DSTU implementations, they are candidates to drop from Core to Extensions when the resource becomes normative.

In most cases, the above set of rules will be sufficient for a Work Group to determine whether an element is part of core or not. In most cases, the answer will either be a clear "yes" or a clear "no". In cases where the answer is not clear, Work Groups are asked to exclude the element for now and look to the DSTU implementers to provide guidance on whether the element meets the requirements for Core.

In some cases, the answer will be "No, but most systems *should* support this element because . . ." or "No, but it's going to be a problem if it's not in Core because . . .". In most of these cases, the answer is still to leave the element out of core, but perhaps to define a "best practice" profile to nudge the DSTU implementers in a direction the WG believes systems should evolve. However, the WG may appeal to the FMG for an exception. The FMG will look at several considerations:

- What rational does the WG have for believing the element should be in core?
- What evidence do they have to support the rationale?
- What will the impact of adding the element be on the implementation community at large?
- Can the use-case be clearly differentiated from other similar use-cases to avoid a slippery slope?
- Most importantly, what will be most beneficial for the adoption of FHIR in the community at large?

What if an element qualifies for core based on commonality, but is known to be bad practice?

In some cases, clinical systems will capture information in ways that is known to be flawed or not in keeping with best healthcare practices. In these circumstances, aligning with the 80% means perpetuating a business practice that is known to be problematic. In these circumstances, a Work Group **may** choose to instead support the more "current"/"best practice" approach to data representation. The primary considerations for making this call is whether it's a reasonable imposition to expect at least 80% of systems to support the new approach in a realistic adoption period (5-10 years). Factors affecting this include how easy or hard it is to migrate from old approach to new, how much added complexity is caused by the new approach and how much benefit to practice and patient care is provided by the change. If the work group agrees to make the change, they should note the exception to the 80% criteria and their justification in the Notes column. Serious consideration should also be given to an extension or set of extensions that support the "old" approach as well.

When faced with multiple alternative expression mechanisms, which do we choose?

I.e. The same information could be organized according to a number of different element structures. (See also questions on inline vs. by-reference and data type choice)

Primary driver for how data is expressed is "what most systems do". However, consideration should be given to simplicity and extensibility. I.e. It should be possible to use the structure to support the vast majority, if not all existing and anticipated implementations (with appropriate use of extensions). Focus is not to cover all edge cases in core.

Where different parts of healthcare represent the same information completely differently, consider the possibility of supporting both mechanisms within a single resource or even of creating distinct resources for each approach (see guidelines on when to create a resource).

When should element content be handled inline vs. by reference?

There are 3 possible ways element content can be conveyed - as elements within a resource, as "contained" full or partial resource instances bound within another resource, or as independently referenced resources.

If the elements could reasonably have a life-cycle (own set of statuses, ability to create/modify independently) that's independent of the current resource, they should be captured as a distinct resource. Also, if the elements overlap with an existing known resource, that existing resource should likely be used.

The choice of whether a resource should be "contained" vs. treated as a reference is an implementation decision, not a design decision at the resource design level (though it might be constrained in profiles).

If the primary purpose of the element(s) is to describe or qualify the resource or information about the resource, then that's an indication the data should be sent in-line as part of the resource content.

Where within the hierarchy of nested elements in a resource should a new element be added and when should new nesting levels be added?

Nesting levels in resources serve 3 purposes:

- They allow sets of elements to repeat as a group
- They allow sets of elements to be treated as "optional" as a group
- They allow sets of elements with related meaning/purpose to be visually identified as related

Of these, the first two are most important and the last should be used with caution due to the added complexity in instances. It should only be used if there are sufficient elements at that nesting level that grouping is necessary to ensure data is not missed. (If sorting data elements is not sufficient to accomplish this consider whether there might be too many elements in the resource . . .)

Elements should be placed within a particular level of the hierarchy based on how the element relates to other elements within that hierarchy. I.e. If an element needs to repeat with elements in a nesting level, it should be part of that nesting level. If it *doesn't* need to repeat, the element should be included closer to the root level of the hierarchy.

Are there design patterns for common semantic scenarios?

Yes. Refer to FHIR Design Patterns

When should you re-use an element definition vs defining a new element?

It's possible to put an element in a resource, and indicate that it's definition comes from another element. (for an example, see ValueSet.compose.include and ValueSet.compose.exclude). Doing this means that the elements will are the same type in all the implementations, and profiles on one are generally profiles on all (though they can be separated if need be). See GF task 14958 for more info

Resource element content

How should resource elements be named?

- See General naming guidelines

What order should elements appear in?

Elements should be ordered roughly in terms of importance, with related concepts grouped together.

What if an element is known by different names in different contexts (discipline, region, use-case, etc.)?

General naming guidelines should be used to pick the "most appropriate" of the names - simplest, most widely used, most likely to be understood in the broadest number of contexts. All other names that are either in common use or are needed to increase recognition within the target implementation community should be captured as aliases.

When should an element have a minimum cardinality of 1?

In general, minOccurs should only be set to 1 (or anything other than 0) if the resource could never be useful or meaningful without the element. This consideration should take into account use-cases involving the transmission of partial or incomplete data.

In addition, when determining cardinality for elements on the root of the resource, designers must consider the possibility of a text-only version of their resource. In most cases, this will mean making elements on the root optional or conditional. However, in some cases, elements conveying essential metadata that "must" be specified even text only situations can be left minOccurs=1. In rare cases, a work-group may determine that a text-only version of the resource is totally useless, in which case they may make root elements required as they see fit.

When should an element repeat?

The decision on whether an element should repeat in core is similar to whether an element should be included in core - if a significant majority of systems that implement FHIR (i.e. 80% rule) will support capturing multiple repetitions of the element, it should repeat.

Note: This means that elements that might theoretically repeat but are not treated as repeating by most systems will not be treated as repeating in FHIR core. Additional repetitions can be conveyed using extensions.

Should the order of repeating elements matter?

The presumption, unless stated otherwise, is that repeating elements are not ordered. However, when defining resources (and extensions) authors *may* choose to define an element where order is significant. E.g. "Most significant first", "in order of occurrence", etc. This ordering must be defined in either the resource definition or base definition of an extension. I.e. It must be explicit in the definition text for the repeating element whose order is significant. Expectation around ordering cannot be introduced by profiles.

NOTE: Requiring ordering of data is problematic because not all systems will have the information necessary to determine appropriate order, and there's no easy way for them to declare this. As a result, it is better practice to explicitly expose an element that represents the semantic on which ordering is expected to occur (timestamp, priority number, etc.) such that ordering can occur when necessary and lack of ordering information can be explicitly conveyed.

Corollaries:

- Order should generally be retained by systems.
- Systems are expected to populate elements based on order when possible but may not always do so

When should elements be marked as 'IsModifier'?

The flag draws attention to elements that influence the meaning of other elements so that implementers don't accidentally misinterpret data by ignoring them. Particularly important for extensions that might not be recognized.

The notion of whether an element is a “modifier” element or not is a key concept in FHIR. FHIR’s simplicity depends on systems being able to freely introduce extensions to convey ‘local’ requirements without breaking interoperability. This in turn, depends on systems’ ability to safely ignore unrecognized /unsupported extensions. We can only safely ignore “regular” extensions because we have partitioned them from “modifier” extensions which are not safe to ignore. It is therefore essential that those defining extensions understand when a new element needs to be treated as a modifier.

In some cases, it may not be clear whether an element could potentially cause the interpretation of other elements to diverge from their definitions because the definitions themselves might not be totally clear. For example, is a prescription for a particular medication that doesn’t explicitly allow substitution a prescription for only the prescribed medication or is some degree of substitution allowed regardless of an explicit declaration? Because the resource definition does not clearly allow for substitution, the element must be marked as a modifier.

Modifier elements must be placed in the instance such that they only have the potential to cause the divergence of the interpretation of their parent element or its descendants. If a modifier element appears on the root of a resource, it impacts the meaning of the entire resource.

Note: In the core specification, because we can always adjust the definitions of elements to ensure that no child element could cause a divergence from the definition, “modifier” elements never *have* to appear. However, in some cases, modifying definitions in this way would cause the element definition to diverge from implementer convention/expectation. In these cases, definitions are kept consistent with convention and the element is marked as a modifier. Ideally, documentation should call attention to the impact of the modifier element and examples provided to make clear to implementers how the element behaves.

Note to designers: A modifer attribute which is missing leaves the resource unreliable, and potentially uninterpretable. So a modifier attribute SHOULD have either a default value or a minimum cardinality of 1. But we do not impose this on all resources as a SHALL; domain committees may be working with the reality that all existing information is unreliable - this is not uncommon. We do expect that most implementation guides - which can be more proscriptive - will nail this down.

What combinations of cardinality and 'IsModifier' are allowed?

Elements with mustUnderstand=Y should generally have a minimum cardinality of 1. If an element with mustUnderstand is omitted, it means that the parent element and all content within it can't necessarily be safely interpreted. No default value can be inferred. If it is possible for the value to be unknown, consider making minOccurs=1 and add an explicit coded value of "unknown" to make this explicit.

Default values are unsafe and interpretation can vary by context. Explicit values are safer

How do I interpret 'Must Support' with respect to slicing?

If an element is marked as 'Must Support' and is then sliced, that does not imply that each of the slices is 'Must Support'. If the intention is that a slice must be supported, then that slice should be marked as 'Must Support'. As an example, if Medication.identifier is marked as 'Must Support', and is then sliced to specify different types of identifiers, unless marked otherwise, the different slices are not 'Must Support'. This implies that a system must support Medication.identifier but can choose to support any of the slices, unless those slices were intentionally marked as 'Must Support'.

If an element is marked as 'Must Support' in its parent's slicing base, that does imply that the element is 'Must Support' in each of the slices. As an example, if Medication.identifier.system was marked as 'Must Support', and then Medication.identifier was sliced, the identifier.system would be 'Must Support' in each of those slices. NOTE: This means that if you do not intend an element to be 'Must Support' in every slice, that you should not mark it as 'Must Support' in the slicing base.

Resource element types

What are the criteria for selecting an appropriate data type?

General considerations

- What level of discrete data do most systems capture for this element?

I.e. Don't select a complex data type if most systems only capture a string or simple date

- When possible use an existing data type rather than creating your own custom structure within a resource to accomplish the same end. It's important to balance the careful tuning of elements to requirements with the ability to leverage commonly re-used structures.
- If all properties of a data type aren't appropriate/relevant for a use-case, consider profiling the data type in a re-usable way and then referencing the data type with its profile.

Coded vs. non-coded

Codes should be used when computer systems are expected to make decisions on the basis of the data or when the information will be subject to statistical analysis. Examples include decision support, business rule validation, enhanced querying, etc. There is no value to using codes if the data is only for human consumption and there is no need to categorize by the element.

Choice of coding data type

See #What coded data type should be used?

Choice of Identifier datatypes

There are 3 data types that can potentially be used to identify things: id, uri and its specializations (oid, sid and uuid), Identifier and HumanId. Each is intended for use in specific circumstances:

id is intended for use within the data types themselves and for "internal" identifiers to FHIR such as referencing another resource. An id is generally not globally unique so it is generally also only useful within a context. Direct use may be appropriate in extensions where a greater degree of control is needed than that offered by the standard data types. For example, specifying a "version" that is expected to be scoped by a companion Identifier element. Work Groups considering using this type within a resource or within an HL7-defined extension are encouraged to discuss this with the MnM Work Group.

uri is also generally intended for use within the data types, though it may appear as an element when communicating network addresses and the complexity of the Contact datatype is inappropriate (see below). Using URIs or one of the specializations for identifiers is only appropriate if a Work Group is 100% confident that all implementers currently and in the future will use a bare URI/OID/UUID/etc. for the identifier rather than the more typical "system + id" approach supported by the Identifier and HumanId types.

Identifier is intended for use in communicating identifiers that have a "type" associated with them and may be split into a human-readable identifier and a namespace in which that identifier is unique.

Note: All non-infrastructural resources (w5 = 'infrastructure') should have identifier 0..* : Identifier. If the committee chooses not to have an identifier, or chooses that the cardinality should be 0..1, they should document the rationale clearly in the published committee notes

Encoded data

There are 4 different mechanisms that can be used to convey binary and other encoded data such as images, embedded XML, PDF documents, etc.: Attachment, base64binary, uri and the Binary resource. Each is intended for use in specific circumstances:

base64binary is intended primarily for use within the data types or extensions or the rare circumstance in resources where the precise type of binary data to be conveyed is explicitly known (and will be fixed for all time). E.g. If an attribute is defined that is fixed to only including a JPEG image. Because this latter situation is unusual, this type will rarely be used inside resource designs.

Attachment is used when there is a need to convey data that is specific to a particular resource and is unlikely to be shared (used by multiple resource instances). It is used when the resource needs to actually contain the raw data. Attachment conveys its data with base64binary but adds additional information for rendering, identifying content type, etc. As such, it should be used in most situations in preference to base64binary.

uri is used when there is no need to include the data within the resource and it is sufficient for the receiver to retrieve the data by reference (and there is no expectation that the data will be hosted by a FHIR system). Because uri provides no context as to the type of data, check sums, human descriptions, etc. it should generally only be used for "system" references where the context is known. Otherwise Attachment is preferred.

Binary is used when there is a likelihood of a resource needing to be shared across multiple instances. It can only be used when the expectation is that the binary content will be managed within the FHIR system. Note that Binary allows for data to be retrieved by a separate query or as part of a bundle or as a 'contained' resource within another resource.

string vs. markdown

String should be used for simple strings where formatting is unnecessary, inappropriate or rarely supported.

Markdown should be used for text that spans multiple paragraphs and/or where bullets and other formatting are appropriate.

Quantities

There are several data types that allow numeric values to be conveyed. These are: [hl7.org/fhir/datatype#decimal], [hl7.org/fhir/datatype#integer], [hl7.org/fhir/datatype#Range], [hl7.org/fhir/datatype#Ratio], [hl7.org/fhir/datatype#Quantity] and the various constraints on Quantity: [hl7.org/fhir/datatype#Age], [hl7.org/fhir/datatype#Distance], [hl7.org/fhir/datatype#Duration], [hl7.org/fhir/datatype#Count] and [hl7.org/fhir/datatype#Money]

decimal is used when dealing with numbers that may have fractional components and where the "unit" of the thing being counted, measured or otherwise expressed is implicit in the data element itself. For example, if there's a need to capture a percentage, it is appropriate to use decimal rather than using Quantity and express a constraint on the unit.

integer is used in the same circumstances as decimal, but where fractional values are not permitted.

Range is used where both a lower bound and upper bound may need to be communicated (even if only one of the two - or even neither - might be known for a given instance. This may be used to express uncertainty (e.g. URG from ISO data types) or to express a particular measured extent (e.g. IVL from ISO data types). When this type is used, the meaning of the range (uncertainty or extent) SHALL be made clear in the definition.

Ratio is used for quantities where both a numerator and denominator exist and need to be communicated separately. It is relevant for things like titres. It should not be used to convey values where a decimal value is appropriate (use either *decimal' or* Quanitity*)*

Quantity is used for any single numeric value that may have units and where the units are not fixed by context. If one of the standard profiles of Quantity are appropriate (see below), the standard profile should be used instead.

Standard Quantity profiles (Age, Distance, Duration, Count or Money) Standard Quantity profiles should only be used if the constraints of the profile will apply for all uses of the data element.

Timing

There are 5 data types that deal that capture time-related information: [1], [2], [3], [4] and [5]

Where none of these types are appropriate, string (possibly with constraints on its format) may be used. (E.g. to capture a stand-alone time with no date)

date is used where the majority of systems capture information with no finer granularity than a date. It allows incomplete dates as well (year or year-month). A constraint profile may be used to enforce full date, though this would be unusual in a resource design. This type should not be used if timezone information is required.

dateTime is used where the majority of systems capture at least some degree of time-specific information (to at least the hour). It also supports capturing of time-zone for dates. The data type allows for incomplete dates, however if any hour-specific information is included, it requires full precision down to the second. Therefore, zero-filling lower components is the convention used to accommodate less precise times.

instant is used where a full timestamp is expected to be supported by the vast majority of systems. This will generally be for system-level elements rather than those to be entered by humans.

Period is used when both a start and end date/time are relevant (though in a given instance, it may be that only one or neither will be specified).

Schedule is used to convey complex timing information involving multiple occurrences and/or repeating times. If all that is required is a list of specific times, dateTime with a 0..* cardinality should be used instead.

Contact

uri, Contact

Anything vs. string

Most common for HumanName and Address, but possible for others

What are the criteria for including a type in a choice of target in a Reference or Canonical?

The bar for including a type in a choice is lower than the "80% rule" because the cost of a type being omitted is higher. Types should be included if the WG can foresee a reasonable likelihood of a type being used in a production system for the specific data element.

What are the criteria for including a type in a choice of multiple types?

The rule for including a type in a choice of multiple types are the same as the rule for including a type in a choice of Reference/Canonical targets.

When should multiple types be selected (i.e. a 'choice')?

When should an element reference a base data type vs. a profiled data type?

When should content be expressed inline rather than referencing another resource vs. both?

When should a resource have multiple elements rather than a single element with a choice of types?

When referencing resources, when should a list of allowed resources be enumerated and when should it be left open to Any?

Vocabulary

Every element that can have a code (types "code", "Coding", and "CodeableConcept") must have a "binding", which defines what codes can be used in the element. (There are a few special exceptions in the infrastructure itself)

How should bindings be named?

A binding name should be a title case word like "ProblemType". The binding name is not particularly meaningful - it's just a name that refers to the definition of the Binding.

Binding names can be changed at any time without creating any implementation impact.

What information is supplied for a binding?

When you build a binding, you specify some or all of the following information on the bindings tab of the spreadsheet:

- Definition: The meaning associated with this binding. Very often this is a clone of the element definition to which it is attached with minor grammatical corrections - but not always
- Binding: How this binding is resolved to a set of codes. Possible values:
    - unbound - there is no particular codes associated with this binding (this equates to a v3 concept domain) - this option will be withdrawn soon
    - special - this is bound to a set of codes that are internal to the specification (generally, this is only used in infrastructural resources)
    - reference - the set of codes is established simply by a general reference to some external web site (generally, this is used for "code" fields where the contents is anything defined by some external framework. A classic case is IANA media types. This should only be used after discussion with the FHIR core project team)
    - value set - this is a reference to a value set. The value set is defined using a value set resource, and provided as part of the spec
    - code list - the specification provides a list of codes, and one of them should be used. The codes can either be defined as part of the specification, or defined elsewhere
- Conformance - one of the following values: required | extensible | preferred | example (must be required for type 'code')
- Reference - a url reference to the source of the definition. It's usage depends on the binding type:
    - (codelist) - an internal reference starting with "#" that refers to another tab in the spreadsheet. The name should be all lowercase with "-" delimiting words. The tab name must be unique in the specification
    - (valueset) - either an absolute URL, or a relative URL which is just the name of a valueset resource (xml or json) found in the same directory as the spreadsheet. In the relative URL case, the value set will be picked up and published as part of the specification, and the relative name must be unique in the scope of the specification
    - (reference) - an absolute URL
- Description - (reference type only) what the build tool shows when the reference is placed in the specification. if this is empty, the URL will be used instead.

Some of these fields can be combined in ways that don't make sense. The build tool generally only allows valid combinations. In addition, resource designers can rely on the build tool to enforce the name uniqueness constraints that are required.

What coded data type should be used?

Most coded elements in FHIR should use the CodeableConcept data type which provides the greatest flexibility. It allows multiple codings to be sent to satisfy the needs of different consumer systems and also provides for text representation to convey what the data enterer actually saw/typed, which is particularly useful when more detail is required than is possible with a code. For some binding strengths, it also allows text only if no code is available. CodeableConcept also provides for ease of transition if an implementation space chooses to switch from using one value set to another as it allows for both old and new codes to be transmitted during the time period when not all systems have yet made the switch.

The Coding data type should *only* be used when the intention is reference a specific code from a code system and not to express a concept generally. For example, when indicating a translation, when linking a resource to a code that defines its meaning, etc. The use of Coding will be rare. It should never be used when there's a potential need for text or translations.

The 'code' data type is only usable if the following conditions apply:

- The binding strength for the value set is "required" (see guidance on binding strength below)
- The element is one likely to be deeply entwined in software behavior in a manner that SELECT / CASE type statements are likely to be desired
- The work group does not see a likelihood for 'most' systems to need to convey alternate coded representations and are comfortable with requiring that extensions be used to convey alternate codings if necessary

In some cases 'string' or 'url' may also have terminology bindings. This should only happen when the data doesn't represent coded concepts and is expected to behave as a string or a url.

NOTE: If the data type is 'code', the expansion of the value set bound to is considered to be 'locked' on the date of the release of version of the StructureDefinition with the binding. Subsequent changes to the bound value set or any underlying valuesets or code systems will not be reflected until a new version of the StructureDefinition is published. On the other hand, if the data type is not 'code', it is possible that the expansion of the associated value sets might change over time as the referenced value set and/or any underlying value sets and code systems change.

What should the binding strength be?

FHIR strives to achieve as much interoperability as possible across systems. Interoperability requires consistent terminologies as well as data structures. Therefore, when possible, work groups should strive to standardize terminologies as well as data structures. Such standardization does not imply that HL7 will mandate what terminologies are used internally, merely what terminologies are expected to be supported for exchange purposes. However, such standardization can be challenging. For that reason, FHIR supports a range of binding strengths.

Required bindings should be used when:

1. A value set can be defined that covers the entire space of the concept across all countries, domains and use-cases and the conceptual space is unlikely to evolve between releases. I.e. It's possible to define a set of codes that covers known needs and those needs are not expected to change
2. The value set represents the concept space with a granularity sufficient to provide useful interoperability for most, though not necessarily all use cases
3. It is reasonable to expect almost all existing implementations to implement maps between their current internal terminologies and the selected standard value set. This mapping may involve some loss of granularity
4. The benefit of global interoperability achieved through standardizing the terminology for exchange is deemed to be worth imposing the cost of mapping on implementers. (This means that the larger the value set is, the greater the interoperability benefit must be.)

NOTES:

- The codes used in the value set must be free for use for all implementers. This means implementers must be able to use the codes, display names and relationships between the codes in the value set without payment of any fee or making any other licensing arangement that would reasonably pose a barrier to global addoption of the value set.
- Where multiple value sets meet the above criteria, the choice of value set for the binding should be driven by lowest overall mapping cost for the global community - i.e. which value set will be easiest for the global community to map to.

Extensible bindings should be used when: The requirements for a Required binding are met, with the exception that complete coverage of the conceptual space is not possible; or where the conceptual space is sufficiently dynamic that new concepts are likely to appear within the implemented lifespan of a single version of the specification.

NOTE: The use of required and extensible bindings doesn't prevent an implementer from continuing to use whatever codes they wish internally, nor does it prevent them from sharing alternative codes that may provide additional information or be required for inclusion due to regulation, legacy requirements or implementer preference. Alternative codes may be sent using additional CodeableConcept.codings or using extensions (if the data type is 'code' or 'Coding'.)

Preferred bindings should be used when the requirements for Required or Extensible bindings are not achievable, but:

- A value set can be defined that is globally appropriate and covers most or all of the desired space
- The work group feels that adoption of or mapping to and from the specified value set would significantly benefit interoperability

NOTES:

- Preferred value sets *should* be free for use. However, a work group may consider a preferred binding to a non-free for use terminology if no free for use terminology meets the requirements for the space
- While not enforcable through conformance processes, preferred bindings make it more likely that systems will choose the same terminology, thus improving interoperability

Example bindings are used when no global consistency of terminology exists or where the conceptual space is too large to be covered by any defined and maintained value set. An example value set merely gives a sense of what sorts of codes might be present. Such value sets should aim for international diversity in the sets of concepts exposed and types of terminologies used.

Should I use a value set or a code list, or leave it unbound?

Firstly, note that a list of codes can incorporate codes defined in some other specification (often v2 or v3), or it can define it's own codes. A value set always refers to codes defined elsewhere.

You would use a code list in one of two case:

- the data type is "code" and the schema should enforce that only the codes allowed are used (irrespective of whether the codes are defined here or elsewhere)
- the data type is Coding/CodeableConcept, and you are defining a set of codes that can be used in resource, along with other codes. This is unusual and you should consult the core team prior to creating this kind of code list.

In other cases, if there is enough agreement on a set of codes, use a value set that defines the set of codes. When there is not enough consensus for that, you provide an example value set.

How do I define a code list

You should only define a code list that has internally defined values if the codes are inherently related to the use of the resource itself. Typically, these are of type "code" and have names like "status", "state", "kind", and a very short list of codes (<10, usually <5) that are often involved in invariants, or discussed in the narrative. These codes are generally specific to FHIR, and not copied in and out of other representation formats. Longer code lists, or concepts that are copied into and out of v2 messages, cda documents, dicom or xds etc, or that are exposed directly on the user interface **should** generally be incorporated from somewhere else, and if they don't already exist, some other place should be created (such as in the general vocab terminology through harmonization)

A code list is a tab on the spreadsheet with the following columns:

- Id - an internal identifier that is never changed
- System - a url that identifies the external system that defines the concept (usually a v2 table, a v3 code system, loinc, snomed)
- Code - the code that is used in the instance (if a System is provided, the code must be valid in the referenced code system)
- Parent - if the code is subsumed by another code (only 0..1 parent supported for now)
- Display - a display name/print name to be used with the code. Display is optional, on the grounds that the codes are related to system workflow concepts, not direct user input, and the users would never be expected to see the code directly. i.e. if the display is missing, the code shouldn't be display directly to the users (display must be provided for externally referenced codes)
- Definition - the formal definition for the code (required for all codes)
- Comment - additional usage information.

Entries in the code list have either an Id (internally defined) or a System (externally defined). You cannot mix internally defined codes and externally defined codes in the same list (for now).

This list provides help for importing codes from other code systems:

- v2 - use a namespace as defined here: http://hl7.org/implement/standards/fhir/terminologies-v2.htm
- v3 - use a namespace as defined here: http://hl7.org/implement/standards/fhir/terminologies-v3.htm
- loinc - use http://loinc.org. Note that some countries do not use LOINC, so LOINC value sets can only ever be preferred or example
- snomed-ct - use http://snomed.info. Be very careful using snomed codes - with snomed-CT, implementers require licensing agreements to use the relationships, and **most** uses will imply that implementers can reason (usually implicitly) with the relationships. Generally, Snomed value sets should be examples only

When should content (negation, uncertainty, null values) be handled in terminology vs. distinct attributes?

Mostly, for internally defined code systems, null value issues don't apply, though the code 'unknown' may need to be defined sometimes.

For FHIR, there is no ubiquitous nullFlavor concept, so the code list/value set should include these concepts directly. Typically, these concepts are included by including the codes from the v3 NullFlavor code system etc.

Descriptive Content

Guidelines for Short descriptions and definitions

Short Description:

- Short descriptions are optional but a build warning will be created when not present.
- They should be included whenever possible, but must do the following - they must further explain or clarify the meaning of the element without introducing ambiguity.
- They should not be redundant with the element name (i.e. they should not merely be a re-statement of the name or the name with the context of its containing element)
- Short descriptions should be short enough that no wrapping occurs when the XML representation is generated in the spec

Definition:

- Definitions are mandatory.
- They need not be a formal definition of a term if the term is well-understood (even by non-domain experts), but must provide additional explanation of the meaning and use of the element.
- They must not be redundant with the name or the short description.

Note: an element has 3 descriptions: name, short description, and definition. All of these define the element, so some overlap is expected. Note that it's easy to mistake that the definition defines the name, but this not correct. All of them are defining the element itself, though it may come to the same thing for some elements.

What makes for a good Short Description?

A short description should:

- Provide an additional information about the use of the element (i.e. not a just repetition of the element name with additional context such as "identifier: the identifier of the resource")
- Should be short (<60 characters) - the resource definition should not wrap
- May be a phrase or an enumeration of allowed codes
    - enumerations should only be provided when
        - there is a single mandatory binding
        - the semantics of the codes are clearly expressed in the code itself
        - there are a small number of codes (no more than 5-6) - i.e. they fit without wrapping
    - enumerations should match the spelling and case of the code and be delimited with " | "
    - if there are too many codes to reasonably express all of them within the length limit, use a representative set and put " +" at the end
- Must be in U.S. English
- Must start with upper-case unless an enumeration
- Should use terminology the majority of implementers will find familiar

The purpose of a short description is to provide more information to the reader than is conveyed by the name of the model element while still being quick and easy to read. Starting with an upper-case helps the short description show up well in the XML

What are the criteria for a good Definition?

A good definition:

- Provides additional clarity for the selected name
- Can be easily consumed and understood by the reader. A guideline is that a definition does not exceed 1 paragraph in length though many definitions can be considered complete in a single sentence. Some difficult concepts may however require lengthy discussions but this is best left to comments and examples (see below)
- Is unambiguous
- Allows the reader to understand the definer's intent for the field
- Should be grammatically correct. The definition should be composed of complete, grammatically correct sentences.
- Is not reflexive
    - It should not rephrase the name in a statement, for example a definition for ConcernCode as "A Code value for the Concern" simply repeats the information conveyed in the name. At the very least, a definition provides synonyms for the concepts that comprise the name to allow the reader a better understanding of the definer's intent.

What goes into Definition vs. Requirements vs. Comments vs. Committee Notes?

The following contrasting sections attempt to explain and illustrate what is placed in each of the aforementioned sections:

DefinitionA short concise (no more than 1 paragraph) description of the purpose of the described element. Every element will have a definition. (the WHAT) RequirementsThe reason(s) for including the element in the specification. A description of why the element is necessary (and under what conditions). May be documented using formal methodologies for business requirements or simple notes. Not all elements will have a requirements discussion. (the WHY and the WHEN)CommentsThe thought processes and design ideas/constraints that went into the creation/definition of the element. The area is also suitable for detailed discussion on the element including such topics as background and use. This section can also be used to augment the definition. Committee NotesThere could be much overlap between Comments and Committee Notes. Essentially, Comments provide additional information about the element whereas Committee Notes document the process, decisions, and recommendations of the responsible Work Group. The Committee Notes are often formal in nature.Whenever Work Groups find a need to debate the inclusion/exclusion of an element, its type, its cardinality, its vocabulary or other constraints, they should capture the rationale for the end decision in the spec to allow this to be remembered and considered in subsequent maintenance. Committee notes is the appropriate place for this information. FMG should monitor for resources with no rationale filled in

As a side note, the name of the element is the Who and the examples provide the How, thus answering all of the essential questions on the specification of an element.

Text Content & style

This section describes methodology rules on formatting and text usage within descriptive content

Headings

Top-level introduction content should be organized into child elements. I.e. the root element should only contain child div elements. Each child div element should start with an <h2> element. The headings should be in the following order. Optional sections may be omitted if not relevant for a given resource.

Introduction:

- Scope and Usage (required): Describes some of the situations/use cases intended to be managed with the resource. May include a child div for "Storyboards" and "Related Specifications". The purpose of this section is to help implementers understand why the resource is needed and how it might be used.

- Boundaries and Relationships (recommended): Explains how this resource relates to others. Particularly important is to differentiate between appropriate usages for related resources when an implementer might be confused about what to reference when.
- Background and Context (optional): Provides additional detail on exactly how the resource is to be used

Here's an XHTML template to work from for the content of the section:

```
<div>
<h2>Scope and Usage</h2>
<p>
<%definition%>
</p>
</div>
<div>
<h2>Boundaries and Relationships</h2>
</div>
<div>
<h2>Background and context</h2>
</div>
```

Notes: There are no pre-defined sections under 'notes'. Notes may be presented as either an un-ordered list of bullet points if the list of considerations is simple. Alternatively, it can be broken into sub-sections with <h3> headings. If subsections are used, the first element in the notes section should be the label of the first sub-section

Markup

- All references to a resource name within the page defining that resources should appear in **bold**. Resources references should always be expressed in UpperCamelCase
- All references to other resources should be bold with hyperlinks to the respective resource.
- All references to resource element names should be in italics. Where the context is evident, only the right-most node is required (e.g. *causalityExpectation*). Otherwise full names should be used (e.g. *exposure.causalityExpectation*. If referencing elements from other resources (and not identifying the resource within context), then the resource name must be included in the name (e.g. *AdverseReaction.exposure.causalityExpectation*
- All headings should make use of the <h2>, <h3> and similar tags. Top level tags within Introduction are <h2>. Top level tags within Notes are <h3>
- Markup should be limited to <i>, <b>, <code>, <pre>, <blockquote>, <p>, <a>, tables, lists (ordered or not), and images. Use of span and styles should be avoided.

Key words

The only conformance terms used are: SHALL, SHALL NOT, SHOULD, SHOULD NOT and MAY. When used to assert conformance expectations, they shall be written in upper-case form

- The conformance terms MUST, MUST NOT, REQUIRED, RECOMMENDED and OPTIONAL will not be used, though they may be expressed in lower case with their usual English meaning
- The phrase "may not" is prohibited due to the likelihood of confusion - use "might not" instead
- The term "resource" should be avoided except when referring to FHIR resources. When used, it refers to the resource design. To refer to instances, use the word "instance" or the phrase "resource instance" or a specific resource name + instance (e.g. "**AllergyIntolerance** instance")
- The term "profile" should be avoided except when referring to FHIR profiles

Mappings

What mappings should be created?

Resources should have mappings to HL7 v2 and the RIM wherever such mappings apply. In addition, each resource should have mappings to at least 1 and preferably more external specifications

How precise should a mapping be?

Mappings are not intended to be sufficient for transformation, but rather to give a general idea of where content from the mapped specification exists in FHIR. As such they can be "hand-wavy" mappings

What are the objectives of RIM mappings?

The primary objective of a RIM mapping is to ensure that the work group has a good understanding of the meaning of the elements in their model. It also helps to provide guidance on equivalences in other resources, helps identify situations where semantics are not defined or where unintentionally duplicate information may be represented. A secondary use is to provide guidance for implementers of v3-based specifications who are interested in interoperating with FHIR. Because the RIM can often be used in multiple ways to represent the same information, these mappings may have limited utility in some cases.

Invariants

What rules should be expressed as invariants?

What should the context be for an invariant?

Can invariants reference elements inside another resource or data type?

How should invariants be named?

Do I need to specify OCL or XPath?

What are the criteria for the English expression of an XPath?

When should an invariant be linked to specific elements?

Events

What events should be created for a resource?

When should an event have multiple request or response resources?

What should go into an even description?

When should an event have follow-ups?

What should go into event description vs. notes?

Search

What search parameters should be included in a resource?

How should search parameters be named?

- Use element name
- unless the element name is meaningless without scope (i.e. Resource.something.somethingelse.type) (in this case, consider just using the parent name)
- unless that creates a duplicate search parameter
    - if it does - then pick one (or both) or use [parent]-[child]
- search parameter names must be lowercase (the build tool enforces this)
- search parameter names that are composed of multiple words should be separated by dashes, though compound words not the dictionary are allowed at committee discretion where they are used by the domain.

What should go into a search description?

How should the type of a search parameter be determined?

When should a repeating parameter be handled as union vs. intersection?

Examples

How many examples should be created?

How realistic should examples be?

What should examples cover?

Additional documentation

What content should go into the introduction and notes section?

When should content be in the introduction and when in notes?

Where should content go that doesn't easily fit in either place?

Tags

What requirements should be handled using tags vs. elements vs. extensions?

Can tags be "must understand/isModifier", and if so how is that handled?

Can HL7 define tags, and if so, how are they published?

Versioning

How are different versions of FHIR distinguished?

What are the rules for changes between DSTU versions and from DSTU to Normative?

FHIR promises no backward or forward compatibility between resources, syntaxes, URLs or any other aspect of the FHIR specification. This is necessary to ensure that lessons learned from DSTU implementation can be applied in the normative specification and represents the purpose of a DSTU. That said, changes will not be made without cause. Any change needs to be driven either by an identified flaw in the specification or the belief it will make a significant difference to the ease with which future implementers are able to adopt the specification.

Where possible (and where sufficient resources are available), transforms will be provided between DSTU versions and subsequent versions.

What are the rules for changes between Normative versions?

The element paths for all elements defined in prior versions must be retained in subsequent versions. This means that:

- elements cannot be renamed (unless a redundant element is retained with the original name and required to be populated with the same value - which seems of limited utility)
    - **OPEN ISSUE**: Do we need a means of allowing linkage between redundant elements in either the instance or in the resource specification?
- elements cannot be moved (same qualifier as above)
- new elements *can* be introduced, provided they do not change existing paths
- non-repeating elements cannot be made to be repeat unless there is a guarantee that the first element of the repetition would correspond to the element that would be sent by systems that do not support repeating elements and that there is no danger in the loss of repeating elements
    - general pattern is to add a new element for the repeating element structure and send a redundant copy of one of the elements in the collection in the non-repeating element

OPEN ISSUE: Can the sequence of elements be changed if that doesn't modify the xpath?

Profile Guidance

Todo

Extension Guidance

An extension definition must make it clear why the extension exists, where and how it's used.

Description

- What information does the extension convey?

Rationale

- Why is the extension necessary and why can't this information be conveyed in existing core elements or other extensions
- Clear use-cases defining the set of circumstances for which the extension applies
- Note whether scoped for either international or realm use (i.e. U.S., Netherlands, etc.)

Status

- Draft/Live in production

--Lmckenzi (talk) 23:14, 4 December 2016 (EST)I think we just use the status attribute for this. Don't know we can track in production or not

Maintained by:

- HL7 Work group: Patient Administration
- Or, Organization that created and maintains. For example US Veterans Administration

Context of use: <Resource|Element>

Example: Link to an example which includes the extension

List of places where used: <location or sites where used> --Lmckenzi (talk) 23:14, 4 December 2016 (EST)I think we can only show this for extensions we use for stuff we publish