FHIR chapter for HL7 on-line training

Introduction 3

Why FHIR? 3

Background 3

Scope (content, infrastructure, business use) 5

Governance & Methodology 6

Relationship with other SDOs 6

License 7

Using FHIR – example architectures 8

Message Broker 8

Native FHIR server with existing back end 9

Native FHIR server with FHIR back end 9

Key Concepts of FHIR 10

Resources 10

What is a resource 10

Types of Resource in FHIR 12

FHIR on the wire 12

Definition & Documentation in the Specification 13

Key Parts of a resource 15

Resource Metadata 18

Datatypes 20

Code 20

CodeableConcept 20

Resource reference 21

Contained Resources 22

Bundles 22

Profiles 23

Security & Audit 24

Notes on specific resources 25

Patient 25

Interoperability Paradigms 25

REST 25

Messages 25

Documents 25

Service 25

REST basics 26

Playing with FHIR 26

Add a new resource 27

Client assigns the id 27

Server creates the Id 27

Retrieve a resource 27

Search for a resource 28

Simple search 29

Searching across resources 29

Retrieve the history of a specific resource 30

Retrieve a specific version of a resource 30

Delete a resource 30

Batch Updates 31

Conformance Statement 31

Specific REST stuff 32

Cross server identity 32

The Binary endpoint 32

XDS 33

FHIR Documents 34

Key Resources 34

The Document resource 35

The List resource 35

FHIR Messages 35

What is a message 35

The message package (bundle) 35

The message resource 35

Events 35

Services 35

Migration from other versions 36

V2 36

V3 messaging 36

CDA 36

Exercises 36

# Introduction

This chapter discusses the new FHIR standard that is being developed by HL7. Currently, it is a draft specification, which means that it will change rapidly until DSTU (intended to be in September 2013). We will try to keep this document accurate to the specification, but in the event of any differences always treat the specification as the truth. Equally, the links in this chapter may not always be correct.

The specification itself is available on-line at [www.hl7.org/FHIR](http://www.hl7.org/FHIR). It is fully hyperlinked & very easy to follow.

It is highly recommended that have access to the specification as you are reading this module, as there are many references to it – particularly for some of the details of the more complex aspects of FHIR.

There are other resources on-line available, in particular some blogs by the FHIR core team. These can be found at:

* Grahame Grieve : http://www.healthintersections.com.au/
* Ewout Kramer : http://thefhirplace.com/

There are also a number of skype conversations that have been created – one is aimed specifically at fhir implementers. To gain access to this conversation email David Hay – david.hay25@gmail.com.

This document draws heavily on work (presentations and personal discussions) from a number of people, but especially the FHIR Core Team:

* Grahame Grieve
* Lloyd MacKenzie
* Ewout Kramer

<<< try to include sample if not too big – ½ page max >>>

<<< change all ‘chapter’ to ‘module’ >>>

# Why FHIR?

## Background

In January 2011 the HL7 Board commissioned a small task force lead by Grahame Grieve to answer the question: “If HL7 were starting afresh today, what would the interoperability standards look like”. In considering this question the task force noted that:

* Version 2 was (and is) extremely successful, but the technology is old and not well suited to the newer requirements.
* Version 3 while based on a robust model, has not been widely accepted and is perceived as difficult to implement.
* CDA has been hugely successful, but was designed as a document and using it elsewhere doesn’t really fit well in all scenarios
* Tooling for HL7 standards has always been an issue, as these generally need to be designed – and built - specifically for HL7, and this doesn’t always occur in a timely fashion.
* There are new Use Cases – especially involving mobile devices – where the current standards were not a good fit.
* Particularly in the online space, the use of a REST based architecture is widely used in other domains.

FHIR (Fast Healthcare Interoperability Resources) grew out of this work.

The goal is to produce a standard that:

* Is easy to implement (or as easy as healthcare interoperability ever can be).
* Is semantically robust. This means that it can be mapped back to the v3 RIM (and often to other specifications like openEHR archetypes)
* Is ‘implementer friendly’ – e.g. uses common tools and formats, and web based technologies for the specification.
* The artifacts should make sense to a human looking at them. While not intended for direct human viewing, being directly understandable helps both implementers and support personnel.
* The artifacts should be able to be validated electronically – so far as that is possible.
* Both XML and JSON should be valid representations.

At the time of writing, FHIR is still under active development and can be accessed at [www.hl7.org/FHIR](http://www.hl7.org/FHIR). The plan is to have it in DSTU standard by September 2013.

It should be noted that FHIR is not ‘version 4’ of HL7, although it builds on the long history of HL7 messaging standards.

To achieve this, the FHIR team has established:

* A specification that is hosted on the web and is fully hyperlinked. For example clicking on a datatype in a resource definition will take you to the definition of that data type.
* All resources are defined in an easy-to-read format that includes a ‘psuedo-xml’ definition, UML diagrams, and links for formal definitions. The format is such that clinicians are able to understand what a resource contains and represents (though the target audience remains implementers)
* All resources have a number of examples that show how a resource is intended to be used.
* The standard is developed as if it were a software application with a build process that automatically generates all the artefacts from a small number of key definition files. The build process validates all definitions, and the examples, to ensure a high quality result.
* Freely available Reference Implementations in Delphi, Java and C# that implementers can download and use – or can use as the basis of their own developments. Links are available to these (and other useful resources) on the front page of the specification (<http://www.hl7.org/implement/standards/fhir/index.htm> )
* A number of on-line servers that implement the FHIR standard and can be used by implementers to test their developments
* Regular ‘connectathons‘ (inspired by the IHE connectathons) where implementers can meet and test their work.
* A number of communications channels (List servers, skype conversations) where implementers can contact the core team and other implementers directly.

## Scope (content, infrastructure, business use)

All aspects of healthcare interoperability are within the scope of FHIR. Commonly 4 types – or paradigms – of interoperability are described. These are:

* Messages
* Documents
* Services
* REST (Representational State Transfer – on-line access)

And their support by FHIR will be expanded on in future sections of this module. It is important to note that all of these paradigms use the same resources to represent the content – they are just wrapped in ‘packages’ that suit the particular paradigm.

However, the REST and Service paradigms are new to HL7. Unlike a messaging paradigm where the messages are used to update repositories (as well as implementing behaviour) the REST paradigm means that the information may not be in the client or server that needs it – it is accessed from some other server when needed, it supports more of a distributed model.

Message updating local system

App

Message

Figure: Messaging updating local system

Query Remote server

App

Figure: App querying remote system

Of course, many other variations are possible – for example the Application could query the remote server, and then store a copy of the data locally for subsequent access.

<<< describe how fhir supports new paradigm (eg REST) – information sits where it belongs and a system gets it when it needs it - ie this is new to HL7 which is very message driven (async replication)

picture of current messaging async replication vs new rest based distributed architecture

not always the best (eg chart labs but good enough for mobile)

>>>

## Governance & Methodology

The Development Process is documented at <http://wiki.hl7.org/index.php?title=FHIR_Development_Process>.

Important points related to G&M:

* The FHIR standard is open source. While it is developed by HL7, there is no need to be an HL7 member to use it (though participation is encouraged).
* There is a Governance Board that oversees FHIR development and has final say on what resources are defined. The intention is that there will only be a small number of fundamental resources (100-150) that form the building blocks of all FHIR artefacts.
* The FHIR Management provides day-to-day oversight of FHIR-related work group activities including performing quality analysis, monitoring scope and consistency with FHIR principles and aiding in the resolution of FHIR-related intra and inter-work group issues.
* Actual development of resources is performed by the Work Groups to which that resource ‘belongs’. E.g. the Pharmacy Work Group is responsible for all medication related resources. There are some ‘infrastructure’ resources that the FHIR Core team are responsible for, but clinical resources are managed by the work group that is already working on that domain within HL7.

## Relationship with other SDOs

The FHIR team (and HL7 in general) have established working relationships with other Standards Development Organizations where that is applicable. Examples of these relationships include:

* IHE (<http://www.ihe.net/> )– in particular the XDS related resources (currently modelled as the DocumentReference resource that represents the XDS entry. There is a separate discussion on FHIR support of XDS on page 22)
* openEHR (<http://www.openehr.org/>) has done a significant amount of work in modelling the clinical domains. They have taken a slightly different approach to HL7 by creating domain-specific models that are ‘maximal data sets’ for that domain.
* DICOM (<http://medical.nema.org/> ). Working together on the Image resource
* W3C (<http://www.w3.org/> ). As the initial work has been in the REST paradigm, FHIR attempts to be as faithful as possible in the use of HTTP constructs (verbs, headers, response codes, mime types) and other standard constructs such as the Atom standard.

## License

FHIR is released under an open source license (though the details of that are yet to be finalized) – you don't even need to be a member of HL7 to use them (although there are significant benefits in being a member of course).

From the front page of the specification:

* FHIR is © and ® HL7. The right to maintain FHIR remains vested in HL7
* You can redistribute FHIR
* You can create derivative specifications or implementation-related products and services
* Derivative Specifications cannot redefine what conformance to FHIR means
* You can't claim that HL7 or any of its members endorses your derived [thing] because it uses content from this specification
* Neither HL7 nor any of the contributors to this specification accept any liability for your use of FHIR

# Using FHIR – example architectures

There are a number of ways that FHIR could be used, especially as FHIR capable services will need to interact with existing standards (HL7 and other) and existing applications for quite some time. Some of these options include:

## Message Broker

|  |  |
| --- | --- |
|  | When using a messaging paradigm (p24), an application like an integration engine can bilaterally convert between FHIR resources and other standard messages. For example, the FHIR project intents to make available standard transforms for conversion between CCDA documents and FHIR documents, and more general CDA document conversion is quite possible. However, it should be noted that as CDA is, in effect, a ‘format’ for information, such conversions are very likely to be specific to particular templates, and their success will depend on how specific that template is.  There are no current plans to do this for v2 messages as their use is quite variable – however guidance for doing so will be made available, and in many ways v2 will be simpler to map than CDA – for example, in general terms, a v2 segment maps to a FHIR resource.  This post from Grahame (<http://www.healthintersections.com.au/?p=972> ) talks about converting from v2 messages, and this one (<http://www.healthintersections.com.au/?p=979> ) is v3/CDA focussed. |

## Native FHIR server with existing back end

|  |  |
| --- | --- |
|  | This is most applicable where there is an existing data source of some type (eg an EMR of PHR system) and the users want to put a FHIR interface in front of it – either as a read, an update or both effectively ‘FHIR enabling’ the system. There will need to be an application of some sort performing the conversion (perhaps based on one of the reference implementation) – e.g. receive a request for a FHIR resource, query the back end system for the data, then convert to a FHIR resource and return.  Note that the conformance resource (p 22) and profiles (p 16) are likely to be very useful in indicating what resources and functions are supported. |

## Native FHIR server with FHIR back end

|  |  |
| --- | --- |
|  | In this architecture the FHIR resources are stored directly in the back end data store, and queried as required. A number of the early systems (including both of the main test servers available on the net) have taken this approach – one using a ‘nosql’ datastore, and the other an SQL database with a simple structure. |

# Key Concepts of FHIR

There are a number of key concepts of FHIR including:

* Resources
  + Extensions
  + Datatypes
* Bundles
* Profiles

## Resources

### What is a resource

A resource is the smallest unit of exchange that ‘makes sense’ in interoperability – such as an observation, a patient or a problem. They are roughly analogous to a segment in a v2 message, or a CMET in the v3 world.

A resource has the notion of ‘identity’ – something that identifies it as a logical ‘thing’, and will have a location (a URI) where it can be found, which will include both the id and the host where that resource is stored..

A resource is made up of properties, each of which is a particular datatype (like string or CodeableConcept). In many resources, a particular property can be of different datatypes, though the property of a particular instance of a resource will be only of one of those datatypes.

For example, in the Observation resource shown below, the component.value can be a Quantity, a CodeableConcept and a number of others. If a specific instance was of type Quantity, then the name of that element would be component.valueQuantity. If, in another instance it was a CodeableConcept, then the name of the element would be component.valueCodeableConcept

<<< small example where a property can be different datatypes – eg component.value >>>

All resources have a human readable ‘text’ element – much like CDA. However, the contents of the text element (the Narrative datatype) is a more complete subset of HTML, and the contents of the text element is at the discretion of the implementer.

The screenshot below shows the definition for an Observation resource (refer to the specification (http://hl7.org/implement/standards/fhir/observation.htm) for the full definition (which may be different to this one).



Points to note: <<change to have these point in callouts >>>

* The syntax is XML – and very close to what an instance of that resource would look like.
* Each element has a multiplicity 1..1 (required), 0..1 (optional), 0..\* (multiple)
* The allowed datatypes for an element are in green – they are hyperlinked to the resource definition.
* Some elements (like the subject) refer to other resources (this is discussed below).
* Where the element has a choice (like value) then the name in the instance will vary according to the datatype it contains – if it is a Quantity for example, then the element name will be valueQuantity - that is what the ‘[x]’ in ‘value[x]’ stands for.

To see an example of an Observation, click in the ‘Examples’ tab in the specification. You will see both XML and JSON examples there.

<Observation xmlns="http://hl7.org/fhir">

<text>

<status value="generated"/>

<div xmlns="http://www.w3.org/1999/xhtml">**Sept 17, 2012: Blood pressure 107/65 (normal)**</div>

</text>

<name>

<coding>

<system value="http://loinc.org"/>

<code value="55284-4"/>

<display value="Blood pressure systolic and diastolic"/>

</coding>

</name>

<interpretation>

...

Snippet of XML version of an Observation

{"Observation": {

"text": {

"status": {"value": "generated"},

"div": "<div>Sept 17, 2012: Blood pressure 107/65 (normal)<\/div>" }, "component": [

{"name": {"coding": [

{"system": {"value": "http://loinc.org"},

"display": {"value": "Systolic blood pressure"},

"code": {"value": "8580-6"} }]},

"valueQuantity": {

"value": {

"value": "107"},

"units": {"value": "mm[Hg]"}

...

Snippet of JSON version of an observation

<<include the examples of xml & json here – fragments at least, of a smaller resource>>

### Types of Resource in FHIR

There are a number of different types of resource that FHIR defines, and these are described at <http://www.hl7.org/implement/standards/fhir/resourcelist.htm>

Currently these are:

|  |  |  |
| --- | --- | --- |
| **Name** | **Purpose** | **Examples** |
| Administrative | Tie clinical concepts to supporting administrative processes | Patient, Practitioner |
| Foundation | Infrastructural | List, Category |
| Clinical | Clinical information about the patient (excluding medications and diagnostics) | Problem, Procedure |
| Medications | Related to drug therapy | Medication, MedicationAdministration |
| Diagnostics | Investigations | DiagnosticReport, Specimen, ImagingStudy |
| Financial | Replated to payment | Coverage |
| Device Communications | Support communication with devices and clinical systems | Device, DeviceObservation |
| Technical | Supporting exchanges with other systems | Document, Message |
| Conformance | Specifying behaviour and resource usage | Conformance, Profile. |

### FHIR on the wire

One thing that is worth mentioning early in the discussion of a resource is the way that it can be represented. Any FHIR resource can be represented either as an XML document, or as a JSON document – indeed all the examples in the specification (on the example tab of each resource) have both representations.

The FHIR team have defined a JSON syntax that is very similar to the XML syntax both for ease of conversion between the two, but also to ensure that the extensibility of FHIR can be expressed in both formats.

There is a post at <http://thefhirplace.com/2013/03/16/attributes-versus-elements-in-fhir-xml/> that discusses this representation, and the reason why it was chosen.

### Definition & Documentation in the Specification

<<< put an example of each bullet point, or refer to it if already present>>>

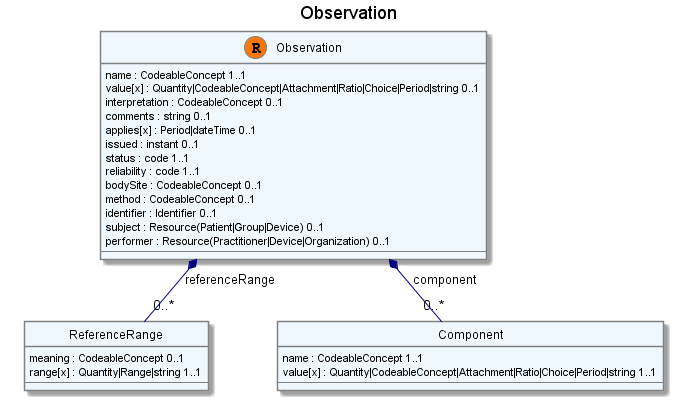
<<< split into summary list for detail section (paragraph) below each one (or refer to)>>>

<<< pull out search into a separate section here>>>

In the specification, resources are defined in a number of different ways (and incidentally this is where the value of building the specification as if it was a software project really has benefits as all the ways are consistent – they are validated and enforced during the build process – including all the examples).

The following are the artifacts you will see for each resource in the specification

**1. A UML diagram showing the main parts of the resource**.



**2. A simple pseudo-XML syntax**.

An example of this is given above, and it is surprising easy to understand. The diagram shows the multiplicity and datatypes for each resource element, and the datatype are hyperlinked to the definition of those datatypes. Also, the simplified ‘1 line’ description for each elements is hyperlinked to the more compete definition.

**3. Vocabulary bindings**.

Where an element is bound to a particular vocabulary this section described those bindings (hyperlinked where appropriate).

**4. General notes**

These describe the purpose and scope of the resource, and any other pertinent notes.

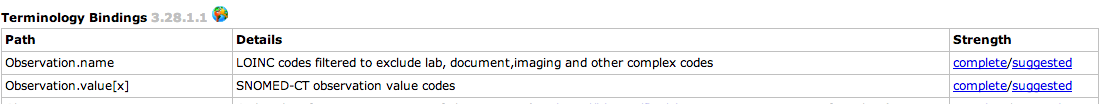
**5. Search Parameters**

Each resource defines the search parameters that ‘makes sense’ for that resource. There is no requirement that a FHIR server should support all search parameters – and it can use the conformance statement and profile to indicate what searches it does support.

Note that there is nothing stopping a FHIR server implementing any search parameter it wants to – but if a particular implementation requires a search not defined here, it is worth contacting the FHIR team to see if it’s worth including in the main specification.

**6. Terminology Bindings**

When the datatype of a resource is a Code or a CodeableConcept, then it can optionally be ‘bound’ to a particular terminology by the resource designers.



The strength of the binding has a couple of options:

* The first link is the ‘extensibility’ of the binding – whether or not you are able to add terminology codes to the specified set (think CNE / CWE)
* The second is the conformance – whether you **must** (SHALL) use this set, whether it is **suggested** (SHOULD) that you use this set, or whether the set is just an **example** (MAY).

**7. Example instance**

Actually the team aim for at least 2 examples – in both xml & json for each resource. These examples are validated during the build process to make sure that they match the specification.

The intention is that eventually there will be examples that show all the possible variations on a resource – or at least the common ones.

**8. Schema + Schematron**

Each resource has an XML schema that is specific to that resource (it is build automatically also). The schematron assertions are manually entered and displayed in the spec as constraints to cover the rules that cannot be expressed in XML schema.

### Key Parts of a resource

A resource has 3 main parts:

* Structured, defined data. Otherwise known as the ‘core dataset’ this is the list of elements that appears in the specification, and which all FHIR implementers need to understand. To be included in this dataset, the rule of thumb is that 80% of systems currently support that property.
* Any number of extensions. Extensions are described below, but allow an implementer to add an element that is needed by their implementation but is not included in the core data set. This is similar to the v2 ‘Z segment’, but there is a defined extension mechanism to avoid the issues that have occurred with Z segments in version 2.
* The narrative or text section. Described above, this allows a human to safely view a resource.

(There is actually a 4th part that a resource can have – other ‘contained’ resources – but this is an advanced topic and is not discussed further in this module).

An example of this is:

<<< all figures need to be numbered and referred to >>>



#### Narrative

The narrative of the resource is a concept borrowed from CDA, and means that the contents of a resource instance can be understood by looking at the narrative.

The specification states:

Every resource SHALL include a human readable narrative that contains a summary of the resource, and may be used to represent the content of the resource to a human. The narrative need not encode all the structured data, but is required to contain sufficient detail to make it "clinically safe" for a human to just read the narrative. Resource definitions may define what content should be represented in the narrative to ensure clinical safety.

<<<text>>>

#### Core content

The Core content of the resource are those elements that the responsible committee have determined are currently used by 80% of systems that hold data matching the resource – otherwise known as the 80% rule.

The purpose of this is to keep the size of the resource to the minimum required to meet the requirements – as opposed to the v3 (and openEhr) concept of a ‘maximal dataset’ – resulting in very bloated resources.

Needless to say, this is one of the most hotly contested aspects of resource design. However, it is mitigated to some extent by the ability to defined extensions to cover those elements that a particular project needs which aren’t in the perceived 80%.

<<<text>>>

#### Extensions

The extension mechanism is what sets FHIR apart from version 3 (and oddly enough closer to version 2). FHIR has a philosophy of the ‘80%’ – a particular property of a resource is only included in the ‘core’ resource if it is currently being used by 80% of existing systems. Not surprisingly, what is in the 80% is probably the most contentious aspect of FHIR!

Taking this approach has the advantage that the resources themselves are kept to a manageable size, and much easier to implement that if every requirement from every realm needed to be in every resource.

However doing so creates the need for a mechanism to allow a particular realm to add the properties that it needs to record within a resource. For example, in New Zealand there are the concepts for a patient of ‘iwi’ and ‘hapu’ – the tribe and sub-tribe of the native Maori population. These concepts are of no interest to a North American or European audience.

It is to accommodate these types of requirement that the extension mechanism has been defined. Extensions can be quite sophisticated, and it is worth reviewing the specification at <http://www.hl7.org/implement/standards/fhir/extensibility.htm> for details.

Extensions are defined within a profile – refer page 21.

<<< say that extensions are in profiles and refer to description>>>

**Simple Extensions: adding an element**

In its most basic form, an extension can be used to hold the value of an element that is not in the Core dataset.

Each extra property that needs to be recorded has its own extension, and there can be any number of extensions in a resource. Extensions can also be nested if required. In the example above there would be 2 extensions – one for iwi and one for hapu.

Each extension has the following properties:

|  |  |
| --- | --- |
| **Name** | **Description** |
| url | This is a reference to the profile within which the extension is defined. This means that anyone who receives a resource with an extension that they are unfamiliar with can download the definition of that extension. Refer to the discussion of the profile for further information. |
| isModifier | <<< re-write from specc >>>  isModifier is used to indicate that this value influences the interpretation, meaning or understanding of other elements in the resource. If the application processing the resource (i.e. displaying it to someone or saving the data in a datastore) does not understand an extension marked as ‘isModifier’, then it should either warn the user about the risk, or not process the resource. |
| Value[x] | The actual value of the resource. As in the Observation example, above the ‘x’ signifies the datatype of the element in exactly the same way as it does for core elements. |

In order to avoid the issue that arose with v2 Z segments where each jurisdiction defined it’s own segments without regard for what other have done, the FHIR team intend to establish a hierarchy of profile repositories that an implementer can query both to determine what a particular extension means, and also to see if there is already an extension defined for their particular need. This is made even easier because a profile is itself a resource, and so can be stored in (and queried from) a FHIR server. The hierarchy that is envisaged includes:

* An ‘official’ HL7 registry that has extensions for data elements that weren’t common enough to make the 80%, but which are nevertheless often required. For example a patient religious affiliation <<check example>>>
* Realm (or country) registries that specify extensions that are specific to that realm (and the iwi/hapu example above would fit here)
* Other registries for anything else.

There is no difference technically between the profiles stored in these registries – simply the governance that is around them.

It is also important that once an extension is in use it should not be changed – if you need to make a breaking change, then create a new extension.

**Extending a core element**

Extensions are also used when it is necessary to modify or extend a core element. For example, the patient.name element (which is a HumanName datatype) contains elements for family name, given name, prefix and suffix – what if you wanted to specify a persons middle name?

The following link (<http://www.hl7.org/implement/standards/fhir/extensibility-examples.htm>) describes in detail how this is done – first defining the extension in a profile, and then referring to the extension in the resource instance as follows:

<name>

<use value="official" />

<given value="Ian">

<extension>

<url value="http://hl7.org/fhir/profile/@iso-21090#name-qualifier" />

<valueCode value="MID" />

</extension>

</given>

</name>

This example shows how to define a middle name of ‘Ian’.

<<need to talk about extensions linking to elements>>

### Resource Metadata

There are a number of aspects to a resource that are more ‘about’ the resource than the actual contents of the resource. They are not included as elements within a resource, but a FHIR server should maintain them.

#### Identity

All resources have the concept of identity. The identity is fixed over the life time of the resource – a change to the resource does not change the identity. The identity can be set either by the client that creates the resource, or the by the server that receives a new one.

Refer to the specification (<http://www.hl7.org/implement/standards/fhir/resources.htm#metadata> ) for further information.

#### Version

The version of the resource changes each time the content of a specific resource instance changes. Combining the identity and the version leads to 2 important concepts:

* Logical id. The fixed identity on the server that hosts the resource. When you ask for a resource from a server based on the logical id, then you will get the most recent version of the resource (This is discussed further in the REST section below). Note that the id can be absolute (specifying the server) or relative (to the containing structure – like a bundle or a resource or a web page). An example of this could be <server>/FHIR/patient/@100
* Version specific id. This is the identity of a particular version of a resource – it may or may not be the most recent version. An example of this could be <server>/FHIR/patient/@100/history/@2. Again, refer to the REST section below for more details and examples.

#### Last Update

<<< text>>>

This is the dateTime that the resource was last updated, and is set by the server as it updates a resource. It is possible to retrieve resources based on this dateTime, which might be useful if you are wanting to synchronize resources between servers, or have a feed that informs you when resources change.

This is done using the history of a resource and can be at different levels of granularity. The following examples are using the REST interface (the other paradigms like messaging queries have yet to be defined).

**GET /FHIR/patient/@2/history?\_since=2012:12:01**

Return all versions of the patient whose id is 2 since December 1st 2012

**GET /FHIR/patient//history?\_since=2012:12:01**

Return all versions of all patients since December 1st 2012

**GET /FHIR/history (?\_format=mimeType) ?\_since=2012:12:01**

Return all versions of all resources on this server modified since December 1st 2012

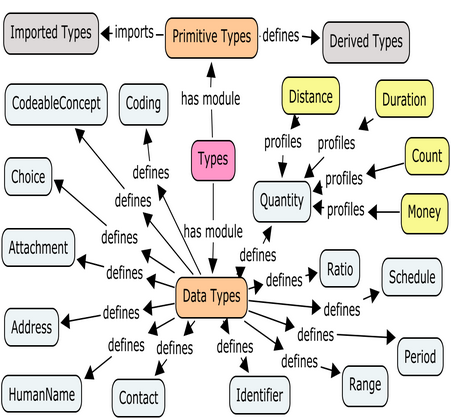
Needless to say, the resulting bundle can be very large!

Sorting and filtering on date can be tricky, particularly concerning timezones, so refer to the specification for more details if performing these types of query.

<<< clarify searching on last update>>>

## Datatypes

Each element within a resource is of a particular datatype (This is the same for all HL7 standards of course). The FHIR datatypes are a simplified version of the v3 datatypes – and are also based on the w3c schema), and the following diagram gives an overview (in the specification, each image hyperlinks to a more detailed description within the page):



The specification describes these datatypes, but some that are more common include:

### Code

A code is used when it is expected that a computer system will need to make a ‘choice’ based on the code. For example the ‘status’ of a Problem Resource might influence whether to display a problem in the patients problem list or not.

The possible values of the code will always be set in the definition of the resource itself.

### CodeableConcept

The CodeableConcept is the most commonly used datatype for coded data, as it is the most flexible. It is analogous to the CE datatype of v3 and is defined as follows:

<[**[name]**](http://www.hl7.org/implement/standards/fhir/datatypes-definitions.htm#CodeableConcept) xmlns="http://hl7.org/fhir">

<[**coding**](http://www.hl7.org/implement/standards/fhir/datatypes-definitions.htm#CodeableConcept.coding)><!-- **0..\*** [Coding](http://www.hl7.org/implement/standards/fhir/datatypes.htm#Coding) Code defined by a terminology system --></coding>

<[**text**](http://www.hl7.org/implement/standards/fhir/datatypes-definitions.htm#CodeableConcept.text) value="[[string](http://www.hl7.org/implement/standards/fhir/datatypes.htm#string)]"/><!-- **0..1** Plain text representation of the concept -->

<[**primary**](http://www.hl7.org/implement/standards/fhir/datatypes-definitions.htm#CodeableConcept.primary) value="[[idref](http://www.hl7.org/implement/standards/fhir/formats.htm#idref)]"/><!-- **0..1** Which code was chosen directly by the user </[name]>

 The key features include:

* It supports multiple codings (effectively translations) which allows common resource instances to be used in situations where different recipients might require different codes.
* It also allows for migration between code systems over time.
* It supports conveying a textual representation of the concept which is important for situations where no appropriate code exists or where the recipient doesn't recognize the code system or wishes to see the full detail of the original concept, not necessarily what was expressible by the selected code.

<<< talk about some specific datatypes here, eg:>>>

### Resource reference

The resource reference is a ‘special’ datatype, as it allows one resource to refer to another. A very common example of this is where a resource representing a clinical concept like a problem or an observation needs to refer to the patient that the resource is about.

A resource reference has the following format:

<[**[name]**](http://www.hl7.org/implement/standards/fhir/resources-definitions.htm#ResourceReference) xmlns="http://hl7.org/fhir">

<[**type**](http://www.hl7.org/implement/standards/fhir/resources-definitions.htm#ResourceReference.type) value="[[code](http://www.hl7.org/implement/standards/fhir/datatypes.htm#code)]"/><!-- **0..1** Resource Type -->

<[**url**](http://www.hl7.org/implement/standards/fhir/resources-definitions.htm#ResourceReference.url) value="[[uri](http://www.hl7.org/implement/standards/fhir/datatypes.htm#uri)]"/><!-- **0..1** relative or absolute reference -->

<[**display**](http://www.hl7.org/implement/standards/fhir/resources-definitions.htm#ResourceReference.display) value="[[string](http://www.hl7.org/implement/standards/fhir/datatypes.htm#string)]"/><!-- **0..1** Text alternative for the resource --> </[name]>

The contents are:

* [name] : the name of the reference within the resource. For example, the problem resource has a ‘subject’ reference to a patient resource.
* Type: the type of resource that is being references – eg Patient
* url: the reference to the resource. This can be a relative or an absolute reference, and can also be version specific. See the discussion on page 13 for further details on this.
* Display: this is to identify what is being referenced. For example, if the reference is to a patient then it might bame the patient name. Specifically, it is not the same as the text element of the resource.

For example

<subject>

<type value="Patient"/>

<url value="../patient/@example"/>

<display value=”patient example”/>

</subject>

would indicate that the subject of this reference is a patient whose name is ‘patient example’, has the id ‘example’ and can be found on the same server as the resource containing the reference.

### Contained Resources

A variant on the resource reference is where the resource being references is actually contained within the ‘parent’ resource. An example of this might be in a careplan where the plan is a short term one concerning a problem that is not in the patients main problem list.

The following snippet shows a problem ‘included’ in a careplan:

<contained>

<Problem id="p1">

<subject>

<type value="Patient"/>

<url value="../patient/@example"/>

<display value="Peter James Chalmers"/>

</subject>

<code>

<text value="Obesity"/>

</code>

</Problem>

</contained>

Note that the Problem has an id – this is used as an internal reference (it’s not like the id that identifies a resource) and in this example is the target of a reference from the concern element as shown in this snippet:

<concern>

<type value="Problem"/>

<url value="#p1"/>

<display value="obesity"/>

</concern>

An important aspect to contained resources is that the resource being contained should not have it’s own identity – i.e. it’s not stored on a server somewhere. If it is, then the reference should point to the resource in the usual way.

In the words of the spec:

**This should never be done when the content can be identified properly, as once identification is lost, it is extremely difficult (and context dependent) to restore it again.**

<<< talk about contained references here>>>

## Bundles

There are many situations where a collection of resources is required. These include:

* The results from a search
* The collection of versions of a particular resource
* A FHIR document
* A FHIR message
* A batch of resource to be processed.

In all these scenarios, FHIR uses the Atom Syndication format to represent the collection of resources. When used in this way, the feed is termed a ‘bundle’ of resources.

Because JSON is a first class citizen, there is a FHIR version of the atom bundle when represented in JSON.

A bundle has some header information, and then any number of resources.

The header contains:

* The title of the feed
* The time (as an instant) when it was created
* An id for the bundle. This is used if the bundle is saved.
* A number of link elements. These are used to describe the application that created the bundle, and url’s that can be used for paging large bundles (if the server supports paging)

Each entry contains:

* A title. This is required by atom, though not
* An Id. This is an absolute uri that points to the logicalId of the resource in the entry element (even if the entry in the element is not the most recent version).
* A link element that does point to the version in the bundle – it is a version specific reference in the format :
  + <server>/<resourceType>{@logicalId}/history/{@versionId}

Here is an example of a bundle in XML:

<<< example of bundle in XML & JSON – example URL >>>

## Profiles

The resources that are described by FHIR can be used in many different healthcare contexts, so it is often necessary for a particular project to be more specific on exactly how it needs to use these resources. This includes:

* Explain how a set of resources is used in a particular context
* Describe restrictions on the use of the elements defined as part of the resource(s)
* Define the extensions that are used with the resources
* Define the searches that apply in this context.
* Describe how resources are bound to terminology in a particular context

All these things are described using a Resource Profile, is which itself a resource that describes how other resources are used in a particular context. Profiles have a metadata section that describes who published the profile, and why, as well as optional lists of resources constraints, extension definitions, and vocabulary bindings.

Profiles are extremely important in FHIR, but can be complex to develop and use. It is the intention of the FHIR team to develop a tool to assist with this. Interestingly, the FHIR resources are themselves described using profiles – enter the url below into a browser to see the profile that described the patient resource:

http://hl7connect.healthintersections.com.au/svc/fhir/profile/@patient

and this will work for any FHIR resource.

## Security & Audit

The security aspects of FHIR are still in active development. FHIR itself does not define security functionality, but does depend on other services to provide that security. See <http://www.hl7.org/implement/standards/fhir/security.htm> for details. The specifics of how security concerns are addressed will vary according to the particular paradigm being used – REST, Message, Document or Service.

From an audit perspective, there are a number of resources that can be used to record activities required for auditing. These include:

* **Provenance**. This resource is used to indicate where a particular resource came from. Note that the provenance resource points to the resource that it describes and not the other way around (although it is possible to contain the provenance resource inside the resource it describes – see <http://www.hl7.org/implement/standards/fhir/resources.htm#contained> for details of how this is done)
* **SecurityEvent** resources are equivalent to the IHE ATNA Audit Record.

# Notes on specific resources

## Patient

The patient resource represents a person or animal receiving healthcare.

<<< note on identity and merging patients >>>

<<< describe the ones to use in the exercises >>>

# Interoperability Paradigms

As described above, it is intended that the same FHIR resources should be able to be used in all the interoperability paradigms required in healthcare. For example a patient resource is the same no matter how it is exchanged.

These include:

## REST

This stands for Representational State Transfer, and is used for on-line, real-time access to information using HTTP protocols – like a web browser. It can be used for updates as well as querying for information. REST has become very popular and is used by many other applications – such as Twitter or Facebook – due to its simplicity and ease of use.

FHIR fully supports REST – in fact it is the most developed so far, but is not confined to that. Most of the examples in this chapter use the REST paradigm.

There is a good tutorial on REST at <http://rest.elkstein.org/>

## Messages

A message is used when you want to send information from one system to another, and you expect the recipient system to update itself as required and then delete the message (other than any audit records of course).

HL7 v2 is all about messaging in this way.

## Documents

A document is all about recording a set of information that applies at a ‘point in time’ about a patient – such as a Discharge Summary or a Referral. CDA is all about documents, so refer to that module for further information. The FHIR take on documents is described below (p28).

## Service

A Service is also intended to be used in an on-line real-time way (usually), but the difference from REST is that a service can incorporate more complex workflow than the simple REST interface can provide. For example you might use a Service in an ordering application if you wanted the service to apply basic decision support to the order, possible modifying or rejecting it.

Use of Services is not described further in this module.

# REST basics

The REST part of FHIR is the one that has received the most attention thus far.

REST is an architectural style rather than a standard and seeks to fully utilize the HTTP protocol (<http://en.wikipedia.org/wiki/Http>) in proving access to resources – it is not simply XML over HTTP – and thus uses the verbs, headers and status codes defined by HTTP.

FHIR utilizes HTTP as faithfully as possible (as with the other standards it uses like Atom) and the page at <http://hl7.org/implement/standards/fhir/http.htm> describes this usage in detail.

Lets start with a play…

<<< how to specify xml/json>>>

## Playing with FHIR

This is the section where you can start to do some practical things with FHIR – and you are absolutely encouraged to do so!

1. install browser plugin (chrome, IE, firefox, safari)
2. sample search - name=eve
3. sample get 1 of those patients

(Note – hosted in amazon – problems try again in 5 mons & tell ius)

<<<describe what they should see – not a screen shot >>>

<<< show both xml & json >>>

<<< format & header>>>

# REST in detail

The following sections discuss how to make RESTful queries against a server, but they are by no means comprehensive. You should have the specification open and be prepared to experiment.

## Add a new resource

There are 2 distinct ways to create a new resource – and the one you choose depends on whether the client is creating the id or the server is.

### Client assigns the id

If the client is assigning the resource Id then use the PUT verb, specifying the ‘location’ of the resource (which is the server name plus the id):

PUT <host>/<resourceType>/@<id>

You can specify the format of the resource being saved (xml or json) either by setting the Content-Type header, or adding an \_format parameter to the request – eg *PUT <host>/<resource>/@<id>?\_format=json to specify json*.

If there is already a resource at that location, then it will be replaced by the resource that you supply in the body of the request. Ideally, the server will create a new version, thus maintaining a history of changes for that resource.

If the request is successful, then:

* The server will return the resource that was saved.
* If there was an existing resource at that location then the status code is 200, otherwise it is 201.
* The **Location** and **Content-Location** headers will be set to the full url (including version)
* The **Last-Modified** header will be the date of update.

The server is able to reject the update for business reasons, in which case the status code is 422.

There are a number of other possible failure codes described in the specification.

The PUT request is also used when it is intended to update an existing resource.

Note that it is only appropriate to use client assigned Ids in a trusted environment.

### Server creates the Id

If the server has the responsibility for creating the Id, then the client should simply POST the resource to the resourcetype root:

POST <host>/<resourceType>

The server will create a new resource of this type, returning the newly created resource, a status code of 210 and setting the response headers as above.

## Retrieve a resource

To retrieve a resource based on its id, issue a GET request based on the following pattern

GET <host>/<resourceType>/@<id>

Where:

* <host> is the name of the server – eg <http://hl7connect.healthintersections.com.au/svc/fhir/>
* <resource> is the name of the resource – eg ‘patient’ or ‘observation’
* <id> is the id of the resource you are after. Note the ‘@’ prefixing the id.

Thus:

GET <http://hl7connect.healthintersections.com.au/svc/fhir/patient/@1>

will return the patient resource with the id of 1 from Grahames server. The response status is important:

* 200 indicates that the resource was found
* 410 means there was a resource with this id, but it has been deleted
* 404 indicates ‘not found’.

The server will also return a number of response headers. These include:

* **Content-Location** will be the full location to the resource, including any version – ie a version specific url
* **Content-Type** will be the format of the resource – xml or json

But there will be others.

You can specify the format that the server should return the resource in in two ways:

* In the request, set the request header ‘**Accept’** to *application/json* or *text/xml* to specify json of xml respectively. This is the preferred approach.
* In the request URL, use the “\_format” parameter to specify the format:

<http://hl7connect.healthintersections.com.au/svc/fhir/patient/@1?_format=json>

The latter option is provided as a convenience for those clients that cannot (or won’t) set request headers.

You will generally get the id of a resource as a result of a search operation (see below).

## Search for a resource

In most cases, a client will not know the resource id, but rather will perform a search of some sort to find the resource or resources it wants, and can then get the resource id from the returned bundle.

Searching is one of the most complicated parts of FHIR, and is still evolving. The reader is advised to study the specification (<http://hl7.org/implement/standards/fhir/http.htm#search>) as well as reading this section. We will cover only the basics here.

Also note that the searches that can be applied to a particular resource can also be specified in the Profile resource (p 16)

In the examples below we will omit the server name to save space, thus:

**/FHIR/patient/search?name=eve**

should really be

**http://hl7connect.healthintersections.com.au/svc /FHIR/patient/search?name=eve**

The general format of a search request is:

GET <host>/<resourceType>/search?param1=&param2= …

Where the parameters are the filters to apply to the search. They will generally match a an element in the resource – though they don’t have to. The parameters have a parameter type as described in the specification. Most of these are straightforward – the tricky one being the qtoken that specifies a namespace as well as a value and is usually used for elements that are of datatypes Coding and CodeableConcept.

When multiple parameters of the same name are given, the result can return resources that contain any of the parameters (union) or only resources that contain all of the parameters (intersection). These are indicated in the type of search parameter in the resource. If the search parameter is of type single, then only 1 should be in the query.

For example:

* the ‘address’ search parameter of patient is of type ‘union’, which means you can have multiple address parameters, and the result will include all resources that have any matching values
* gender is a qtoken of type single – you should only have one

The server should also honour the format request in the Accept header or the \_format parameter.

The result of a search request is always a bundle containing the matching resources (even if there is only a single result).

There are a number of ‘standard’ searches that apply to all resources, and in addition each resource can define other search parameters that are specific to that resource. We will focus on the patient resource.

### Simple search

For simple searches – ie for a single resource – simply create a GET request with the required parameters. E.g.

* /FHIR/patient/search?name=eve
* /FHIR/patient/search?address=home
* /FHIR/patient/search?gender=M

### Searching across resources

Cross resource searching is a bit complex at present (It is due to be resolved soon).

A common scenario is where you wish to ‘de-reference’ a resourcereference datatype – for example suppose you want all the problems for a particular patient.

Examining the problem resource shows that the patient for a problem is given by the ‘subject’ element, which is a resource reference – i.e. it refers to the patient resource with a given id. So, if the patient id is ‘example’, then:

* **/FHIR/patient/@example** would return the patient resource
* **/FHIR/problem/search?subject.\_id=example**  would return all the problems where the subject was the resource whose id was ‘example’

Note that this will only work if all of the resources are on the same server. The situation gets a lot more complicated if there are problem resources on different servers, or where the problem resource on one server has a resourcereference link to a patient on another server.

as for each server:

<< actually, how would you do that??>>>

* you first need to

## Retrieve the history of a specific resource

As described above, FHIR has the concept of resource versions. A server is not obliged to provide versioning functionality, but if it does, then there is a specified way of using it.

The format for this request is:

GET <host>/<resourceType>/@<id>/history

This will return a bundle containing all the previous versions for the resource.

(As an aside, it is also possible for a server to return history at a higher level – either all the changes for a specific resource type, or all resources. This is intended for use in server synchronization scenarios).

## Retrieve a specific version of a resource

You can also return just the specific version (assuming you have both the the resource id and the version id).

The format for this request is:

GET <host>/<resourceType>/{@id}/history/{@versionId}

The result will be the specified version of the resource (unless it was deleted, in which case the server will return a 410 status code).

## Delete a resource

To remove a resource, use the DELETE verb, e.g.:

DELETE <host>/<resourceType>/@<id>

Assuming the delete was successful (ie there was no business process that prevented it occurring), then the server will return a status code of 204 (no content). If the delete cannot occur, then the returned status code is 405 (method not allowed).

A successful delete instruction means that the server will no longer return a resource to a simple GET request for that resource. If such a request is made, then the server will return a status code of 204 (no content) – note that this is different to requesting a resource that has never existed, which would return a 404.

The previous versions of the resource are not removed, and can be retrieved using a version specific request.

## Batch Updates

The batch update facility allows a client to submit a number of different resources in a single batch (atom feed) – ie a bundle. There are 2 main purposes:

* To support the notion of a transaction (as the batch processing must succeed or fail as a whole)
* To support push-based pub-sub – eg where different servers are being synchronized.

From a server perspective, batch processing is hard – especially with regard to id management, but from a client perspective it is easy – just create a bundle with the given resources and POST it to the server root.

The server processes the batch as if each resource had been submitted separately (though does need to treat the batch as a transaction and be able to roll back a failed update).

After processing, the updated batch is returned to the client (and there may be some differences between the submitted batch and the processed batch).

Note that although the batch processing uses a bundle to contain the resources to be processed, there are a couple of important differences:

* The order of resources in the batch must not be significant.
* Any one resource can only occur once in the batch.

If you do intend to use batch processing, then read the specification carefully.

# FHIR specific REST stuff

## The Binary endpoint

<<binary>>

The binary endpoint (<http://www.hl7.org/implement/standards/fhir/http.htm#binary>) allows a FHIR server to store arbitrary binary data, but reference and manipulate it in the same way as any other resource with a couple of exceptions:

* There is no provision for search (as the contents are opaque)
* The format is fixed to that of the document (and Content-Type HTTP headers are used to indicate this)

The actual end point is [service-url]/binary

A common use of the binary endpoint is in the XDS support (see next section) where the DocumentReference resource has the location element, which is a url that can point to the document stored in the binary endpoint.

The following snippet illustrates this.

...

<location value="http://example.org/xds/mhd/binary/@07a6483f-732b-461e-86b6-edb665c45510"/>

...

Another scenario could be images references by the ImagingStudy resource:

<image>

<number value="1"/>

<uid value="urn:oid:2.16.124.113543.6003.189642796.63084.16748.2599092903"/><dicomClass value="urn:oid:1.2.840.10008.5.1.4.1.1.2"/>

<url value="http://localhost/fhir/binary/@1.2.840.11361907579238403408700.3.0.14.19970327150033"/>

</image>

# REST based Use cases

## Cross server identity

<<< to do >>>

## XDS

The IHE XDS profile is designed to support the sharing of documents between systems. More detail is available at this link (<http://wiki.ihe.net/index.php?title=Cross-Enterprise_Document_Sharing>), but basically it defines an ‘affinity domain’ of co-operating systems that contains a single registry holding ‘index’ information about documents that are stored in multiple repositories within the domain. (It gets a lot more complex than this in practice, but this will do for now).

There are then a number of specific transactions that are defined, including:

1. A **document source** saves a document (along with the required metadata) to a **document repository**, which in turn registers the document/s with the **document registry.**
2. A **document consumer** queries the **document registry** for a patient and other query parameters, and receives a list of documents that match that query
3. From the list, the **document consumer** then queries the indicated **document repository** directly for the document/s that it wishes to view.

The XDS profile supports many different types of document – PDF, CDA, text – but has a fixed set of metadata about those documents.

In FHIR the registry entry is represented by the **DocumentReference** resource which contains all the required XDS metadata to enable a document consumer to query the registry. (in exactly the same way as querying the document registry would do). Thus a **document consumer** in FHIR would to a normal FHIR query against the **DocumentReference** resource to return a list of matching resources (in an atom bundle of course), and can then query the repository directly to retrieve the document.

The location of the document is given by the value of the **location** property of the resource – which is a standard HTTP URI. This could be a FHIR document (stored at the **binary** endpoint of the server) or any other valid URI. It is up to the repository to enforce any security that is required.

Expressing the above transactions in FHIR-speak, and assuming that you are saving a PDF document to a FHIR Sever, then one way you can achieve this is as follows:

1. A **document source** saving a document. This actually has two parts:
   1. The **document source** saves the PDF to the **binary** endpoint of the FHIR server. This can either be a POST (where the server will assign an id and return it in the location header) or a PUT where the document source assigns the id, and the server will replace any existing document at that location.
   2. The **document source** creates a **DocumentReference** resource, and saves it in the FHIR server.
2. Querying the registry for matching documents is a standard FHIR search against the **DocumentReference** endpoint.
3. Retrieving the document is a standard HTTP GET against the URI of the document from the search results bundle.

Although the first transaction – adding a new document – is more complex than the IHE equivalent, the big advantage is that the whole process is ‘normal’ FHIR transactions – nothing special.

There are a number of alternatives to the ‘2 phase approach’ of storing a document.

* You could place the DocumentReference and the content (as a base-64 encoded Binary resource) into a bundle, and POST it to the server root and have the server do all the work
* Place the Binary resource inside the DocumentReference resource as a contained resource. This does have the side effect of storing the content with the DocumentReference which may or may not be required
* Create a custom Service that accepts both binary and DocumentReference and performs the required actions.

Like everything else in FHIR, it is likely that the approach will be modified as real-world implementations proceed.

# FHIR Documents

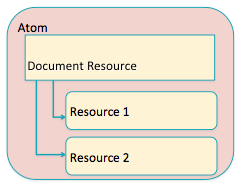
A document is all about recording a set of information that applies at a ‘point in time’ about a patient – such as a Discharge Summary or a Referral.

<<<FHIR resource very similar to CDA>>>

<<<header, sections entry = resources, text >>>

The module on CDA has more details on the overall nature of documents, but briefly a document has a context (patient, author, date, document type) and a body containing the data in one of more sections (eg medications, lab results, problems).

In FHIR, a document is constructed using a bundle (atom feed) that contains resources just like any other bundle, with the addition of a specific resource – the Document resource – that contains information equivalent to the CDA header. It must appear first in the bundle.



<<< picture should show doc resource has sections which point to resources >>>

<<< section is a container of resources >>>

<<< in most cases resource CAN just have text (like level 2 CDA) >>>

<<< each resource has the reference to the patient – i.e. not derived from patient reference in document resource (unlike CDA)

Just like any other bundle, you can either physically include the resource inside the bundle, or just include a reference to it. In either case the Id of the resource must be a version specific resource (because it might change after the document is created.)

## Key Resources for documents

Some resources are particularly useful for documents

### The Document resource

Equivalent to the CDA header.

<<< more detail >>>

<<< table comparing doc resource with CDA header >>>

<<< point out differences – eg no confidentiality >>>

### The List resource

A general resource that summarizes other resources and provides a reference to them. For example, if you wanted to represent a list of medications, then you would include the medication resources in the bundle, and then have the document section point to the list resource, which in turn points to all the medications in that list (or section). You also have the option of having the list point to a resource that is held separately – ie not physically inside the bundle.

<<< using list will be common way >>>

<<< show how it works >>>

## Document Layout options

DocumentResource

Section

List Resource (code = problem)

Detail Resources (eg problem)

Section

List resource (code = problem)

Detail Resources (eg medication)

Section

Detail resource (code=procedure)

<<< sample comparing CDA & FHIR document – ie same doc.>>>

<<< use the above as the sample >>>

<<< have 3 attached files (FHIR xml & json + CDA)

# FHIR Messages

<<< summarize spec here>>>

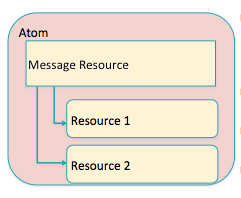
<<< 2 pages max >>>

<<< refer reader to spec >>>

## What is a message

* Collection of resources sent as a result of some real-world event intended to accomplish a particular purpose
* Event Codes & Definitions, like HL7 v2
* V2 segments broadly map to resources
* Includes a “Message” resource, similar in purpose to Message wrapper and MSH segment
* May have associated behavior
* Can be conveyed via MLLP, SOAP or other means

## The message package (bundle)



## The message resource

## Events

# Services

<<< nothing stopping using FHIR resources in SOAP, not not yet defined >>>

see page 9

What is meant by a service

When to use it (?device example)

SOAP vs REST

# Conformance Statement

As will be evident from the description above, while FHIR is simple to consume, it can be complex to provide FHIR services - particularly for searching and batch processing. Further, not all servers will support all resources - in fact it is very likely that most servers will support only a subset of FHIR resources, and only a subset of the defined searches.

A server therefore has a mechanism to describe the services that it provides in a Conformance Statement <http://hl7.org/implement/standards/fhir/conformance.htm>).

A conformance statement is actually a standard FHIR resource that is available at the ‘conformance’ endpoint – thus the url <http://hl7connect.healthintersections.com.au/svc/fhir/conformance> will a return a FHIR conformance resource that describes the capabilities of Grahames server.

The conformance resource describes:

* General information about the server and the software
* The REST resources that are supported
* The Messaging events supported
* The Document types supporte

It is envisaged that they will be useful in a number of scenarios:

* Describing an existing implementation
* Describing what a solution can do
* Describing what it is desired for a solution to do

To describe what search facilities a server supports, the server uses the Profile resource. The profile facility in FHIR is described elsewhere (<link>) and allows a system or implementation to take the base FHIR standard and describes how it should be used in a specific context – which include search capabilities.

As an example, the following URL will retrieve the profiles from the test server where the a Patient resource is constrained.

<http://hl7connect.healthintersections.com.au/svc/fhir/profile/search?resource=patient>

The result is a bundle containing a single profile, containing a structure element of type Patient that defines the elements of Patient that this sever supports, and the searchParams that are supported. An example is given below.

<searchParam>  
 <name value="name"/>  
 <type value="string"/>  
 <repeats value="union"/>  
 <documentation value="a portion of either family or given name of the patient"/>  
</searchParam>

# Migration from other versions

<<< direct to grahams blog entries >>>

# Quiz

<<<10 questions about content from this document (don’t have to go on line)>>>

<<< multi choice or single word >>>

1. Which interoperability paradigms are supported by FHIR

REST

Messaging

FTP (not)

Documents

Services

2. What is the way that more than one resource can be grouped together

3. How Does a List resource manage other resources

it refers to the them

it contains them

it doesn't do this.

# Exercises

1. GET a resource for a specific id (=5) and paste result to moodle – both xml & json

2.

we supply ‘blank resource template – for a patient (must fail)

they add data & post

they search for that based on name

they GET by id

return resource

Additional exercises

Update it to indicate that the patient has died. (Hint: you will need to look at the spec)

Get the version history

Get a specific version

Send to us

Some thoughts to get you going:

<<< too many bullet points>>>

<<< the play is simple GET’s & simple Searches >>>

<<< include specific queries against grahams server >>>

<<< then a more serious one >>>

* All of the resources have examples. You can copy these into your tool as a starting point.
* You can also access the Subversion repository at <http://gforge.hl7.org/gf/project/fhir> and download the sample files (and the whole FHIR specification for that matter) to your local machine. The specification is open for reading/downloading by any user (though edit rights are more closely regulated!).
* Use the Test Servers as validating tools as well. There is nothing like updating a resource on-line and then seeing your updates in the next search!
* Use a browser and a rest plugin (like postman for chrome or fiddler) to send requests to the test servers. These plugins will allow you to set request headers, and view the response headers and status codes back from the servers.
* Use the search queries below to get resources from the test servers, then copy them into your xml tool, change them, validate them and send them back.
* Use the Test servers to convert between XML and JSON by setting the appropriate request headers (or \_format parameters) as discussed below.
* Get hold of a validating XML tool (like Oxygen or XMLSpy) and download the FHIR schema from <http://hl7.org/documentcenter/public/standards/FHIR/fhir-all-xsd.zip>. The xml schema and schematron (and there is one schema set per resource) will help in constructing valid FHIR instances.