22 September 2021
EMA/286879/2021
European Medicines Agency

Electronic Product Information (ePI) Standard and API Specification v1

Document version 1.2

Table of contents

1. Document purpose

The purpose of this document is to present the description of the intended ePI Standard and its Application Programming Interface (API) version 1.

This ePI is a standard based on FHIR (Fast Healthcare Interoperability Resources), an open international data exchange standard for healthcare information, http://hl7.org/fhir

2. Scope

The scope refers exclusively to the ePI standard and its API.

This standard and API specification are based on the latest version of FHIR, available at http://build.fhir.org/resourcelist.html, built on top of R5 publication. For details on versioning of the ePI API and FHIR, see 4.1. See http://hl7.org/fhir/directory.html for a list of all FHIR versions.

3. Introduction

3.1. FHIR

FHIR is a recent standard from HL7 that makes it easy and quick to build REST based APIs for healthcare applications. FHIR solutions are built from a set of modular components called "Resources".

A general introduction to FHIR can be found here:

http://hl7.org/fhir/summary.html

Developers and architects may wish to read these more technical overviews:

http://hl7.org/fhir/overview-dev.html

http://hl7.org/fhir/overview-arch.html

Those with a clinical background can start here:

http://hl7.org/fhir/overview-clinical.html

In general, this specification will not copy or repeat information that is available in the FHIR standard (http://hl7.org/fhir). Instead, references to the relevant parts of FHIR are given. A general knowledge of FHIR, gained from reading the introductions above and further reading at http://hl7.org/fhir will be necessary to fully interpret this specification.

Although references to FHIR are given, FHIR is a wide and flexible system, and not every aspect of FHIR specification will be supported. This document shows which parts of FHIR do apply to the ePI API. The parts that are supported shall conform to the FHIR specification rules.

FHIR is an emerging standard and is being actively tested in many live implementations around the world. Therefore it is possible that changes to FHIR may require changes to this specification. This API will be up-versioned if and when changes are necessary, in a controlled and communicated manner. For details on versioning of the ePI API and FHIR, see 4.1.

FHIR documentation references in this API refer to the current major release of FHIR called R4 (at http://hl7.org/fhir) for general FHIR information that is not subject to change. References to the resources used in this API are given to the build server draft version (http://build.fhir.org/resourcelist.html).

3.2. Definitions

An API can be defined in various ways; the definitions below form a good start for this[1]:

1. "It is a set of **routines, protocols**, and tools for building software applications."
2. "It expresses **a software component** in terms of its **operations, inputs, outputs**, and **underlying types**."

If we take the second definition, we can expand on the terms used to make it more particular to the situation at hand:

Element	Description
Software component	System hosted at EMA
Operations	Create, read, update, and delete
Inputs	Search terms, documents, metadata attributes
Outputs	Documents, metadata attributes
Underlying types	e.g. Composition, MedicinalProductDefinition, Bundle, etc. (see 6.)

3.3. What the API is not

It should also be noted that there are misconceptions and fallacies about APIs. An API is not:

- A software component that you install on a computer
- A process that automates human activities
- An end-to-end system between the NCAs and EMA

3.4. Flexibility and constraints

The definition of the API must be such that it addresses concerns of all the stakeholders as opposed to a small number of stakeholders. This is the trade-off between genericity and specificity, and to be able to specify an API, the following points must be taken into account:

- The API must meet the requirements.
- The stakeholders have different needs as they have different business processes, IT infrastructures and budgets.
- There will only be one API for all stakeholders.
- It is important to draw the line between generic features, usable by all stakeholders, versus specific features, usable just by one or a few stakeholders only.
- Features that appear to be specific to one or a few stakeholders must be implemented on the client side and are out of the scope of the API definition.

3.5. Spelling

The resources and attributes in this specification are defined as per FHIR convention, which has standardised on US spellings. However, EMA uses British spellings, which have been used within this

[1] Based on Wikipedia: http://en.wikipedia.org/wiki/Application_programming_interface

specification in the descriptions and textual explanations. This generates a certain mismatch that is however unavoidable.

4. Specification

The specification for the API is based on the RESTful style API. The same style of API was adopted for the SPOR API 1.x and 2.0, PSUR API and for the Common Repository. This is for the sake of consistency but also for its clarity, ease of use with minimal infrastructure and its clear separation between resources and the operations that can be applied on those resources.

4.1. Versioning

4.1.1. ePI API versioning

The ePI API is based on the FHIR specification. Both ePI API and FHIR will keep separate versioning schemes and evolve at their own pace. Each version of the ePI API will be based on a particular version number of FHIR, as recorded in this document.

This API will be up-versioned (for example, to become v2) if and when changes are necessary, in a controlled and communicated manner and taking into account dependencies on other systems using FHIR. See http://hl7.org/fhir/directory.html for a list of version numbers of FHIR releases. FHIR servers communicate their supported version as part of their conformance statement (see http://hl7.org/fhir/capabilitystatement.html).

4.1.2. XML schemas versioning

FHIR APIs are not versioned at schema level. A given FHIR server shows its version and properties using its CapabilityStatement resource (see http://hl7.org/fhir/capabilitystatement.html).

FHIR schemas are issued with a release of the FHIR standard, via the downloads page of the relevant release of FHIR (http://hl7.org/fhir/downloads.html). These schemas will not be altered, although future releases of the API may adopt newer iterations of the standard. Variation is accommodated by supporting different subsets of the elements in the FHIR models and schemas, and the use of extensions (see below). FHIR allows for validation beyond what is possible with XML schema, including schematron and FHIR profile-based validation. ePI API will come with its own profiles to validate FHIR resources, and these profiles will be specific for the version of the ePI API specification.

XML document instances should not include a schemaLocation attribute, since this can be file system dependent and not transportable between systems. Modern XML tools support schema validation without the use of schemaLocation in instances.

4.1.3. Service versioning

Each endpoint URL will be prefixed with /v{version}, where version is the service version number. The service URL is case sensitive.

i.e. `GET /v1/Bundle`

Where a breaking change is required, a new versioned endpoint will be released. The previous version will be supported for a specific duration.

4.2. Authentication and authorisation

All services require authentication, unless explicitly stated otherwise in the service definition. Authentication is based on an API subscription key, provided in HTTP headers over SSL. The system uses Role-Based Access Control to determine the level of access. The roles are assigned during user registration.

4.3. FHIR extensions

FHIR deliberately does not cover every localised detail of every healthcare domain. Specifically accommodating every last information point for the world's diverse healthcare data items would make the FHIR core unmanageably large and complex. Instead FHIR defines the most commonly used subset of data items and lets individual implementation extend this, in a controlled, enforceable and well documented manner. For more details, see http://hl7.org/fhir/extensibility.html. Some data items within this API can use FHIR extensions, and these are documented within the individual specifications for those resources as used in this API.

4.4. HTTP methods

The API makes use of the standard HTTP methods such as GET and POST to read and write respectively from and to the servers.

These are described in detail as part of the standard FHIR specification (see http://www.hl7.org/fhir/http.html, with a summary at http://www.hl7.org/fhir/http.html#summary).

4.5. HTTP errors and status

The API will make use of a number of HTTP status codes where applicable (see: http://www.hl7.org/fhir/http.html#2.21.0.4 and http://www.hl7.org/fhir/http.html#summary).

Not all of the above referenced HTTP codes are used in this API.

Those that are used:

Name	Code	Description	Comment
Read Update	200	OK	
Create	201	Created	
Create Update	202	Accepted	For an asynchronous operation, indicates initial basic success, with more work ongoing
Delete	204	Success and No Content	Success - no data needs to be returned in the body. Compare to 200, which usually returns the created data. Deleting a resource that doesn't exist gives a 204, not a 404.
Search Update Create	400	Bad Request	Resource update failed basic validation or search parameters failed basic validation, or no id provided for update.
All	401	Not Authorized	Operation needs authorization and no authorization was attempted

Name	Code	Description	Comment
All	403	Forbidden	Operation needs authorization and authorization failed
Read Search Update Create	404	Not Found	Unknown resource or unknown resource type (for Search, Update)
Update Delete	405	Method Not Allowed	Can't update a resource that didn't exist. Or not permitted to delete
Update Create	422	Unprocessable Entity	The proposed resource (while basically valid) violated applicable FHIR profiles or server business rules.

Note that updates will never create a record that did not exist before.

Apart from targeting specific individual resources, updates can also be achieved with Bundles of type transaction (see 4.7.). These create or update multiple resources, and return a Bundle of transaction results, each having an HTTP result code (see 4.7.1.).

Whenever there is any sort of failure, in addition to an appropriate HTTP response code, extra information will be returned in an FHIR OperationOutcome resource (see http://hl7.org/fhir/operationoutcome.html).

The server may also issue an OperationOutcome whenever the HTTP response code is a success. This would indicate that there are warnings or hints found during the business rules validation.

4.5.1. Asynchronous updates

Asynchronous operations are not considered in this ePI API specification. Details will be added in future, when applicable.

4.6. FHIR references and identifiers

FHIR uses two separate types of identifiers, known respectively as the "id" and the "identifier". These sound similar but are significantly different and it is important to distinguish their purpose and use. The id, of which there is only one, is how the resource is accessed on a technical level (record read, write, location), and is specific to the FHIR interface. The identifier(s) are human readable strings that are used as working numbers for the day-to-day identification of ePI and exist outside of FHIR. Both ids and identifiers are strings and can be numeric or alphanumeric if desired.

The two types are needed because the uses are very different. Ids are only for internal software use of the API. Identifiers are for human users of the software system. It is possible in theory for the id to be the same string as the identifier. This is an attractive idea but has problems in practice. Every resource has to have one persistent id that never changes, from the moment the resource is first created.

4.6.1. FHIR resource id (1..1)

This is the RESTful id of a resource, which corresponds to its location on the server, and can be used to directly access the resource. This is not a business identifier. It is a technical identifier and should never be exposed to a standard user. The digits/letters usually have no "real world" meaning, and do not correspond to any another number. There can only ever be one id per resource, and in normal operation it never changes. It is only ever used in the context of the FHIR interface and is always generated by the server, never the client or a user.

FHIR data consists of a set of resources, normally on a "RESTful", web-based server. Each resource can be thought of as a web page, and it has an id that can be considered as its location.

e.g. `/server/Bundle/4be6d0b5-9d39-4367-9c6d-ed030790db01`

where 4be6d0b5-9d39-4367-9c6d-ed030790db01 is the FHIR RESTful id. (The id is shown here as a UUID, but it need not be – other formats are equally possible.)

A software system that accesses the URL above will directly see the data for that single resource (or an error if the id is not recognised). This is a direct read access, with no searching or retrieval of other connected resources. Note that standard users will not normally see or interact with these ids and will never see the URLs that software uses internally. All URLs and ids will normally be hidden within the GUI software that provides the working screens to the business user. Hence the length of these ids will not be an issue in day-to-day use.

The id can be considered as metadata, because it is not part of the ePI data itself – it is just a record id, or a database id. In FHIR terms, the id is not defined on a per-resource basis but is inherited from the base class of all resources: Resource. It is therefore documented separately from each resource (and can be easy to overlook). For example it is not shown in the list of elements here: http://hl7.org/fhir/documentreference.html#tabs-struc, but instead is covered here: http://hl7.org/fhir/resource.html#tabs-struc.

4.6.2. References

FHIR resources can be thought of as pages, and these pages have "references" between them that act like hyperlinks (see http://www.hl7.org/fhir/references.html).

It is usually necessary to use more than one FHIR resource type to represent some useful collection of data items. This involves having several resource types, and using the RESTful id of one as a reference in another. In the following example, Section is directly a part of the Composition resource, but the subject is not (it is a reference).

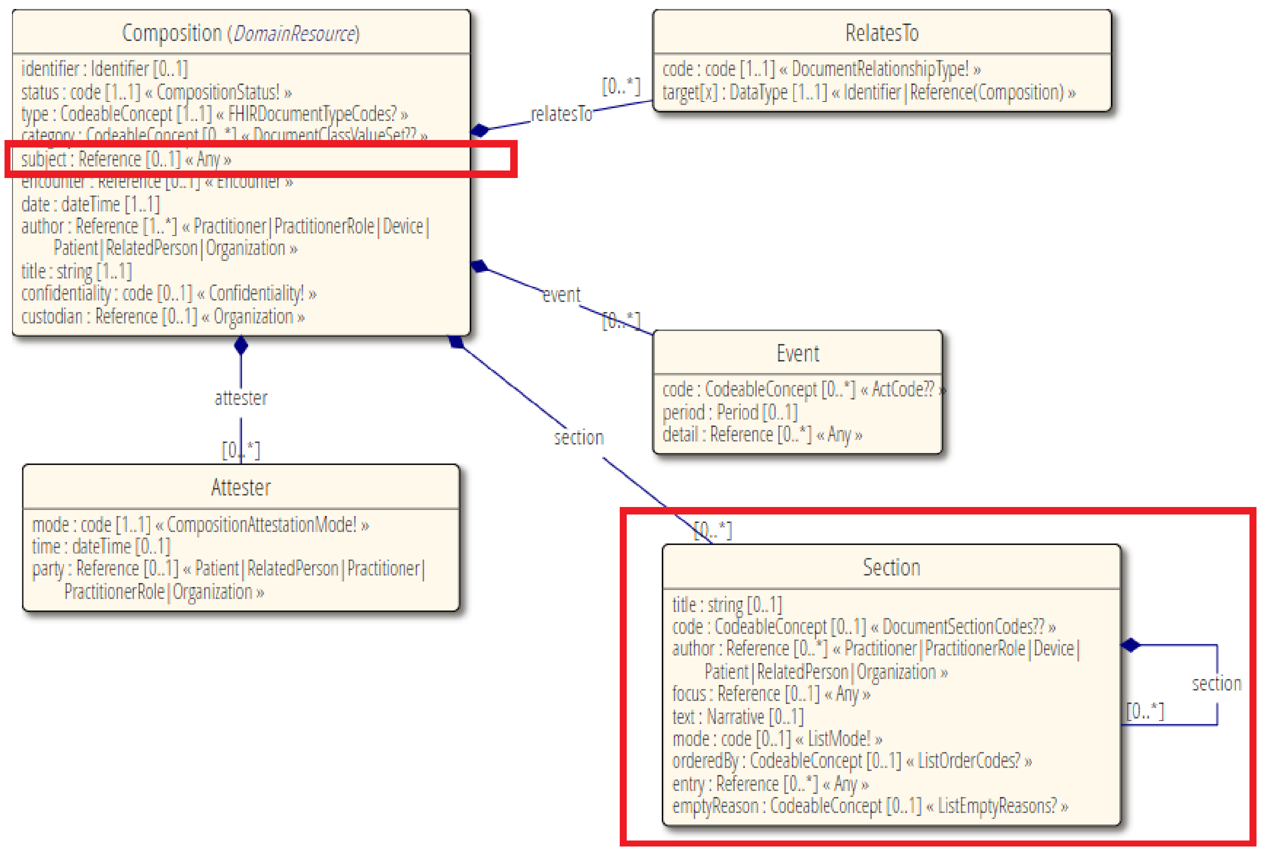
Composition (DomainResource)
identifier : Identifier [0..1]
status : code [1..1] « CompositionStatus! »
type : CodeableConcept [1..1] « FHIRDocumentTypeCodes? »
category : CodeableConcept [0..*] « DocumentClassValueSet?? »
subject : Reference [0..1] « Any »
encounter : Reference [0..1] « Encounter »
date : dateTime [1..1]
author : Reference [1..*] « Practitioner|PractitionerRole|Device|Patient|RelatedPerson|Organization »
title : string [1..1]
confidentiality : code [0..1] « Confidentiality! »
custodian : Reference [0..1] « Organization »
RelatesTo
code : code [1..1] « DocumentRelationshipType! »
target[x] : DataType [1..1] « Identifier|Reference(Composition) »
relatesTo
[0..*]
event
[0..*]
Event
code : CodeableConcept [0..*] « ActCode?? »
period : Period [0..1]
detail : Reference [0..*] « Any »
attester
[0..*]
Attester
mode : code [1..1] « CompositionAttestationMode! »
time : dateTime [0..1]
party : Reference [0..1] « Patient|RelatedPerson|Practitioner|PractitionerRole|Organization »
section
[0..*]
Section
title : string [0..1]
code : CodeableConcept [0..1] « DocumentSectionCodes?? »
author : Reference [0..*] « Practitioner|PractitionerRole|Device|Patient|RelatedPerson|Organization »
focus : Reference [0..1] « Any »
text : Narrative [0..1]
mode : code [0..1] « ListMode! »
orderedBy : CodeableConcept [0..1] « ListOrderCodes? »
entry : Reference [0..*] « Any »
emptyReason : CodeableConcept [0..1] « ListEmptyReasons? »
section
[0..*]

Figure 1. Example showing Section and subject.

This resource-oriented or page-oriented view of data has implications, because the full relevant data set is split over a series of resources, connected by references. Subject to business constraints of what makes sense and is allowed, these can be treated either as separate resources, retrieved and updated individually, or as a group of resources accessed together. In ePI, this is in use by combining Compositions with other elements such as Lists or MedicinalProductDefinition to produce some higher value outcome.

4.7. Bundles

The ePI API makes use of the FHIR Bundle resource (see http://www.hl7.org/fhir/bundle.html).

Bundle is used in many APIs but has particular significance for ePI, as a document-oriented API, since it is used as the basis for FHIR documents.

A Bundle is a container resource that is used whenever a group of more than one resource is needed.

Each Bundle has some basic header information, including its type (searchset, transaction, transaction-response, batch or document), and a total number of "hits", if it is a set of search results. It then consists only of a repeating "entry" structure, each of which contains one resource (of any type), and then possibly a request or result section, for use with transactions.

Some examples of the use of Bundles are:

- Documents

FHIR documents consist of a Composition resource for the headings, sections and text, and other resources for the supporting structured information. For ePI, this information will be List resources, containing references to MedicinalProductDefinitions (for the relevant products), and binary resources for images. These are all wrapped in a Bundle, of type "document". All this represents one single document, and these documents are the core entities of ePI (see 5. on representation of an ePI with multiple Document Bundles). Note that because Bundles have other uses in FHIR, and this API can refer to multiple documents at once, it makes use of Bundles of (document) Bundles.

- Search results

Used when receiving a Bundle of 0 or more resources, of the type requested, and possibly other types that are linked to that type (requested using "_include"). This is a Bundle with type "searchset". Also used when multiple resources are retrieved using an operation such as $everything.

- Transaction

Used when a linked set of resources must be created or updated. It acts like a repeated RESTful call, all in one call. This allows for "atomicity" and referential integrity (if any parts fail, every part is rolled back). Also provides a way to link different resources correctly when creating a set that must reference each other by ids. These ids are normally server assigned, and the client does not know them in advance. Bundles that have items linked via temporary ids get these automatically replaced with the real ids when the data is saved.

- Transaction-response

A transaction is a way of performing several http calls at once. After this, several sets of http results are needed together, and this uses a transaction-response Bundle.

- Batch

In some cases a service may accept a set of resources to be processed, but with no requirement for transactional behaviour or link resolving.

Bundle general schematic:

```
Bundle
    type= transaction|transaction-response|searchset|batch|document
    total=N (for searchset)
    [entry
        {resource}
        [request (used in a transaction to give http commands)
            method=POST|PUT|GET|DELETE
        ]
        [response (used in transaction response to give http results)
            location={URL of resource, including id}
        ]
    ] *
```

4.7.1. Document Bundles

Detail schematic of a Document Bundle, as used in this API:

```
Bundle
    type=document
    entry
        Composition <!-- first resource must be a Composition -->
             [contained
                    Binary <!-- images can be contained and referenced -->
]
             section
                    text “The HTML text of this section”
    entry
        List <!-- for this API, second resource is a List of product references -->
             entry
                    reference MedicinalProductDefinition/{product-id}
             (repeats)
```

4.7.2. Transaction Bundles

Detail schematic of a Transaction Bundle, showing how linkages work:

```
Bundle
    type=transaction
    entry
        {Parent Resource Type}
            {attribute of child resource type}
                reference=tempUuid (temp local id of child, gets replaced by server)
        request
            method=POST
    entry
        fullUrl=tempUuid (matching temporary local id)
        {Child Resource Type}
        request
            method=POST
```

The result would be in this form:

```
Bundle
    type=transaction-response
    entry (one per created resource, same order as incoming transaction)
        response
            location={ParentResourceType/parentid} (URL says type and id of created
resource}
    entry
        response
            location={ChildResourceType/childid}
```

The created parent resource will have a reference in it that points to `{ChildResourceType/childid}`

4.7.3. Bundle endpoints

Bundles can be of a mixed set of resource types. For this reason, Bundles being sent to the server are posted to the root of the server (e.g. /v{version}), rather than to a specific resource type endpoint.

Bundles can also be retrieved from the other, resource-specific, endpoints however. For example, a search would return a set of resources in a Bundle of type “searchset”.

4.8. Searching

Search capabilities are offered on every resource based on GET operations, using a number of query parameters.

e.g. `GET /v1/Bundle?status=pending`

Each resource search endpoint will list a number of recognised query parameters that can be used to filter the results of a search. Out of all the possible query parameters, a maximum of 10 can be provided in a single search, in no particular order.

It is also worth noting that searches will be performed against the latest version of the resource that the caller is authorised to view. No match against historical information will be considered, but history can be accessed via the "version" operations.

4.9. Paging and sorting

All FHIR results from a server are subject to paging. This is described here: http://hl7.org/fhir/http.html#paging

This only affects results where there is more than one result (i.e. searches). It is possible to override the default page size, by asking the server to supply more records per page using the "_count=N" search parameter modifier.

e.g. `GET /v1/Bundle?composition.type=100000155532&_count=100`

As documented in FHIR, paging works by each "page" of search results (a Bundle), having links to the first, last, next and previous pages. Implementations only need to use these supplied links, from the Bundle header, to navigate the entire search results. In technical terms, this operates by the caller using the appropriate URL, which contains a search token that is unique to this search result set, and a page number.

e.g. `GET /v1/Bundle?composition.type=100000155532&page=3`

Knowing this allows a client to construct the URL for any page in the search results. However there is no requirement to be able to parse and use the token, because the necessary URLs provided can be used to reach each page in turn.

Searches can be sorted using the "_sort" parameter, as described here: http://hl7.org/fhir/search.html#_sort

4.10. Resources and representations

For an API with a RESTful style, a resource is anything that can be identified and manipulated by a set of HTTP verbs. Resources are defined by FHIR and referenced in the services in the rest of this document (in particular, see 7).

Not all of those resource types will be directly exposed as RESTful endpoints – some are only used embedded within others. Resources can be expressed using various representations depending on the need of the user and the nature of the resource. In the context of this API, the representations for resources are, according to their media type defined by IANA:

- **`application/fhir+xml`** – used to indicate that the resource is represented by xml data.

- **`application/fhir+json`** – used to indicate that data is represented using the JavaScript Object Notation, which is a programming language independent data format, expressing information in the form of key-value pairs.

The default resource representation is application/fhir+xml and it is the client's responsibility to indicate if application/fhir+json is required. For this purpose, the client must make use of the `Accept` header field in the HTTP request.

If the representation requested is not supported by the server, then an appropriate error is returned by the server to the client (see 4.5.)

Examples:

- Request for a resource representation in xml format (may be omitted as default):

  `Accept: application/fhir+xml`

- Request for a resource representation in JSON format:

  `Accept: application/fhir+json`

4.11. Encoding

All resources are UTF-8 encoded, unless explicitly stated otherwise in the service definition.

4.12. Request parameters and searches

For this API specification, the parameters for a request can be provided in a number of ways to the server:

- **Path**: `/v1/[type]/{id}`

where the single parameter is the resource's FHIR id. [type] represents the name of a type of resource e.g. Bundle. Note that resource names in FHIR are always case sensitive and in upper camel case.

- **Query string**:

  `/v1/[type]?{param}={[op]value[,value]}[&{param}={value}]`

e.g.: `/v1/[type]?name=example,exampletwo&_count=100`

where the resource type is followed by a name-based query and a request for up to 100 records per page.

`[op]` represents possible use of other operators than "=". (see http://hl7.org/fhir/search.html#prefix)

`[,value]` represents possible use of comma separated values for "or"ed criteria.

`[&{param}={value}]` represents use of multiple query phrases, which are logically "and"ed together, or the use of extra query modifiers such as _count, _format, _sort.

The actual parameters that can be used are defined for each part of the API. (See also http://www.hl7.org/fhir/http.html#search and http://www.hl7.org/fhir/search.html)

- **Header of the request**: for example: `Accept: application/fhir+json` which is used by the server to determine which representation will be returned to the client (in this case overriding the default of XML).

All of the above can be used jointly in the same request to the server. The service URL is case sensitive.

4.12.1. Parameter characteristics

In the definitions below, all endpoint path parameters are mandatory unless shown in square brackets ([]).

String based searches in FHIR are by default case and accent insensitive, and a field matches a search string if the value of the field equals or starts with the supplied parameter value. In other words, "starts with" is assumed.

The :contains modifier can always be added to allow full substring searching. :exact can be used to restrict to exact matches in terms of string position and case sensitivity.

For full details of how query parameters work in FHIR, see http://www.hl7.org/fhir/search.html.

To prevent excess server load, for this API the number of search parameters per URL is limited to 10.

4.12.2. Full text search

The FHIR API supports searching on the text of multiple fields using the "_content" parameter. See http://www.hl7.org/fhir/search.html#content.

4.12.3. Chained searches

The ePI data is stored in FHIR as multiple resources, linked by FHIR references. A Document Bundle resource will contain a Composition and a Product List, which references products as MedicinalProductDefinitions (via the PMS id), in a set of referenced PackagedProductDefinition resources. Compositions are never used on their own and API has no specific endpoint for them. Endpoints exist only for Bundle and for List.

Each endpoint can be queried. It is possible, and often necessary, in FHIR to use search parameters from child resources even when querying on the parent resource. This is known as a chained search and is described here: http://www.hl7.org/fhir/search.html#chaining

An example in schematic form would be:

```
GET /v{version}/Bundle?{parent-param}={value}&{child-attribute-name}.{child-
param}={value}
```

In the above example, child-attribute-name is the name of the child resource when used as an attribute in the parent resource. For example, Bundle.composition is a reference to the Composition within the Document Bundle, and can be used to access its search properties e.g. composition.title. Note that only search parameters can be accessed this way, not every attribute of the resource.

A chained query could be:

```
GET /v{version}/Bundle?composition.title:contains=acmedrug
```

or

```
GET /v{version}/List?item:Bundle.composition.title:contains=acmedrug
```

Note that although chained queries can ask questions about data in a linked child resource, this is still a query exclusively on the List resource and so will only return List resources and not the Bundle resources that are queried, see also "_include" below.

4.12.4. Including other resources in search results

Every FHIR resource's endpoint can be queried, as described elsewhere, and using the specific parameters defined in this API. But each resource endpoint normally only fetches query results for that particular resource type, not any others that may be linked to that data.

However, when searching it is possible for the results to include extra resources that are linked to the parent. This uses the "_include" or "_revinclude" parameters. See

http://www.hl7.org/fhir/search.html#include

An example would be:

```
GET /List?item:Bundle.composition.title:contains=acmedrug&_include=List:item
```

Note that this includes the child resource by using the attribute name of it as used by the parent. It does not use the name of the child resource type (which would be Bundle). The result will be a List with references to Bundles, as well as the actual Bundles themselves.

4.13. Metadata

The metadata associated with resources is documented here: http://hl7.org/fhir/resource.html#metadata.

Metadata includes the resource "versionId" and "lastUpdated" date. Both are available on every resource.

"versionId" is the number of the FHIR history version. This is incremented with each save of a resource. It cannot be queried directly, but is instead accessed by the "_history/{version-number}" method.

"lastUpdated" is the server date of the last change to any data item in the resource and is also therefore the date of the last "save". It can be queried using a special query parameter called "_lastUpdated". This works in the same way as any other query parameter and is documented with examples here: http://www.hl7.org/fhir/search.html#all.

These items are defined for every FHIR resource, because they are in the resource class, which is the base type of all resources. They appear in the full XML or JSON representation, when the resource is returned from a server, although, being server assigned, they are usually omitted when sending data to a server.

Example:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Bundle xmlns="http://hl7.org/fhir">
    <id value="4be6d0b5-9d39-4367-9c6d-ed030790db01"/>
    <!-- metadata is near top of resource -->
    <meta>
       <versionId value="2"/>
       <lastUpdated value="2018-10-27T18:40:21Z"/>
    </meta>
       etc.
```

4.14. Standards

- All dates/times returned or provided as path parameters must be expressed in the timezone UTC and comply with the formatting of the allowed formats of the ISO-8601 standard.
- The API supports a maximum URL size of 2048 characters – including the hostname, resource path and query parameters. This limit is subject to ongoing technical investigations.

5. ePI standard resource structure

This standard makes use of multiple resources to represent single business concepts.

5.1. Summary of use of resources by the ePI standard

The table below summarises how the ePI standard uses the FHIR resources Bundle and List. The names in the table are not FHIR naming, but are created for the purpose of describing the use of the resources by this API. Further details are provided in the following sections and figures. (For a summary of FHIR resources used in this API, including Bundle and List, see 6.1.)

Name	Description
Document Bundle	A Document Bundle is a Bundle with type=document. A Document Bundle contains a single PI document (i.e., an SmPC or a package leaflet or labelling or another document) represented using a Composition resource and optional Binary resources. A Document Bundle also contains a Product List. See Figure 2.
Envelope Bundle	An Envelope Bundle is a 'Bundle of Bundles', and has type=transaction. It transports Document Bundles for a medicine, when going to and from the API server. An Envelope Bundle also contains a PI List and may contain a Translation List. See Figure 4. These Envelope Bundles are only for temporary transport. The contents are extracted, and the envelope is not retained.
Product List	Each Document Bundle contains a Product List. The Product List carries the PMS IDs, referencing MedicinalProductDefinition resources for the relevant products.
PI List	Each Envelope Bundle contains the PI List. The PI List has references to the entire set of Document Bundles relating to the product (whether or not the actual Document Bundles are present in that particular Envelope Bundle). The server uses this PI List to know what Document Bundles make up the set for this product, including all types and any translations of them (see below). To add or remove Document Bundles from the product set, send a new copy of the PI List (in an Envelope Bundle) with the relevant changes, and include any Document Bundles not previously sent.
Translation List	Each Envelope Bundle may contain a Translation List. A Translation List states the Document Bundles that are translations of each other (i.e. a list of all the translations of a given document such as an SmPC, including the original). Every document in the Translation List must also be present in the most recent PI List, which always has the full set.

5.2. *Groups of resources*

ePI is composed out of multiple documents (summary of product characteristics [SmPC], package leaflet, etc.). Every document is represented in the same way in FHIR. This diagram represents the structure of any document:

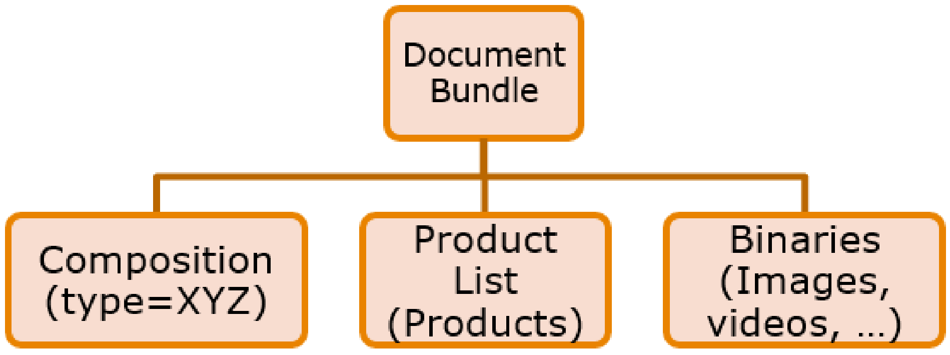

Figure 2. Structure of a Document Bundle.

The type of the Composition identifies the type of document.

We call these "Document Bundles" to differentiate them from Bundles used for other purposes (envelopes and transactions) (see also 4.7.1.).

A document can be linked to other documents using a List of Document Bundles (a PI List or a Translation List).

The Composition resource represents the structure of the textual parts of the document. Nested Sections represent the structure of the document, shown bottom right of Figure 3.

The Section.text item, with type "Narrative", contains HTML and references to binaries (which are shown in Figure 2 as Binary resources). SPOR is used as master for controlled vocabularies, where OMS provides the link to the owner of the document (mapped in the author attribute) and RMS provides all of the terms of type Coding (for example document type, section type, etc.).

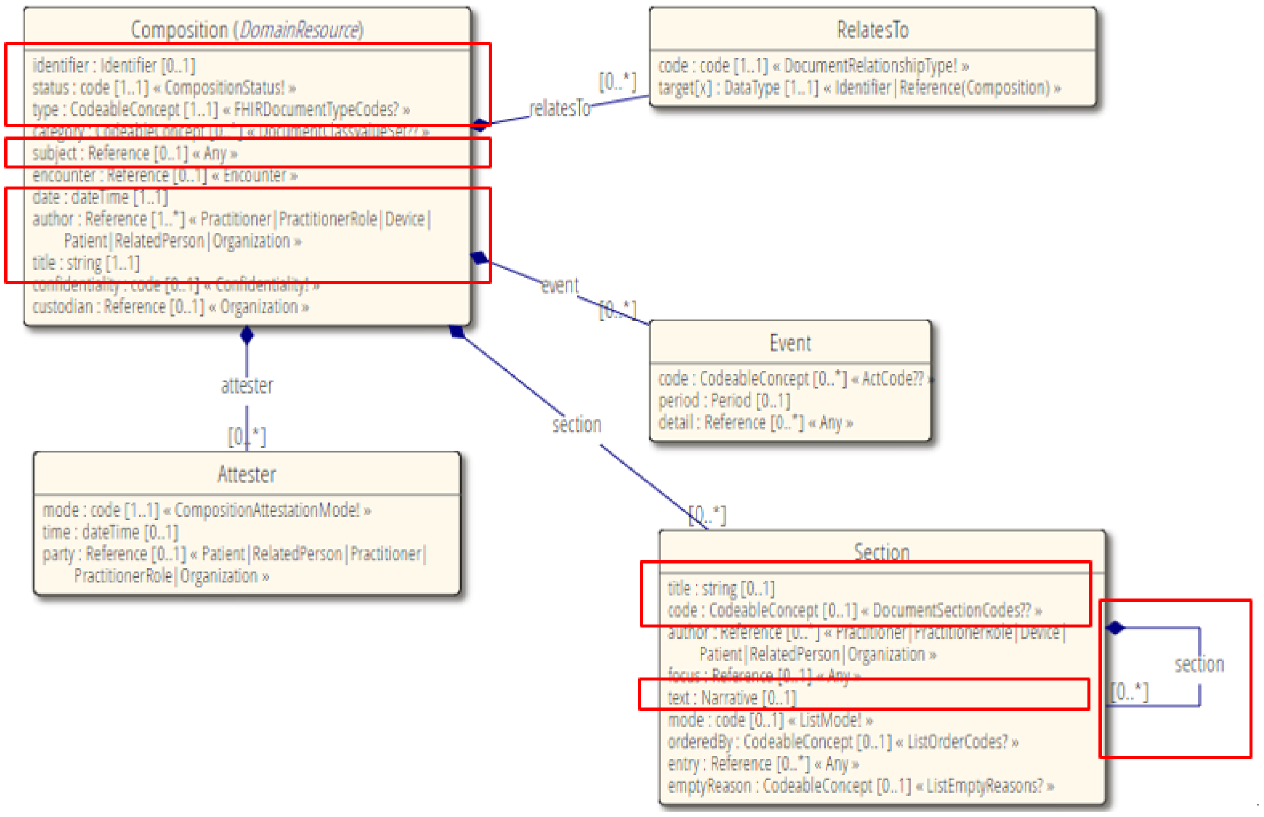
Composition (DomainResource)
identifier : Identifier [0..1]
status : code [1..1] « CompositionStatus! »
type : CodeableConcept [1..1] « FHIRDocumentTypeCodes? »
category : CodeableConcept [0..*] « DocumentClassValueSet?? »
subject : Reference [0..1] « Any »
encounter : Reference [0..1] « Encounter »
date : dateTime [1..1]
author : Reference [1..*] « Practitioner|PractitionerRole|Device|Patient|RelatedPerson|Organization »
title : string [1..1]
confidentiality : code [0..1] « Confidentiality! »
custodian : Reference [0..1] « Organization »
relatesTo
[0..*]
RelatesTo
code : code [1..1] « DocumentRelationshipType! »
target[x] : DataType [1..1] « Identifier|Reference(Composition) »
event
[0..*]
Event
code : CodeableConcept [0..*] « ActCode?? »
period : Period [0..1]
detail : Reference [0..*] « Any »
attester
[0..*]
Attester
mode : code [1..1] « CompositionAttestationMode! »
time : dateTime [0..1]
party : Reference [0..1] « Patient|RelatedPerson|Practitioner|PractitionerRole|Organization »
section
[0..*]
Section
title : string [0..1]
code : CodeableConcept [0..1] « DocumentSectionCodes?? »
author : Reference [0..*] « Practitioner|PractitionerRole|Device|Patient|RelatedPerson|Organization »
focus : Reference [0..1] « Any »
text : Narrative [0..1]
mode : code [0..1] « ListMode! »
orderedBy : CodeableConcept [0..1] « ListOrderCodes? »
entry : Reference [0..*] « Any »
emptyReason : CodeableConcept [0..1] « ListEmptyReasons? »
section
[0..*]

Figure 3. FHIR Composition resource.

An ePI set of documents is represented as shown in Figure 4.

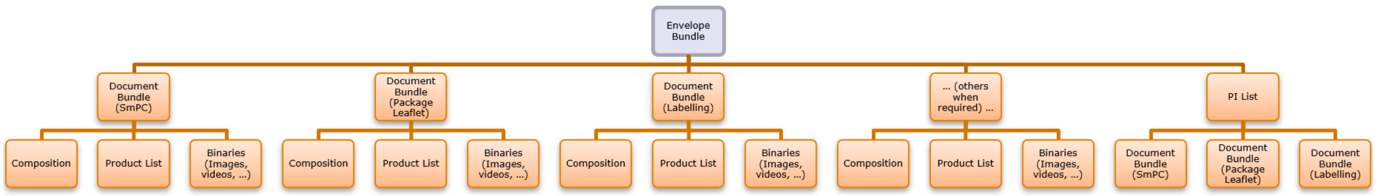
Envelope Bundle
Document Bundle (SmPC)
Document Bundle (Package Leaflet)
Document Bundle (Labelling)
... (others when required) ...
PI List
Composition
Product List
Binaries (Images, videos, ...)
Composition
Product List
Binaries (Images, videos, ...)
Composition
Product List
Binaries (Images, videos, ...)
Composition
Product List
Binaries (Images, videos, ...)
Document Bundle (SmPC)
Document Bundle (Package Leaflet)
Document Bundle (Labelling)

Figure 4. An ePI set of Document Bundles, in an Envelope Bundle.

A set of Document Bundles is shown (SmPC, package leaflet etc.), each with its own components. Note that the grey "Envelope" Bundle at the top is only used for the transport of the data to or from the server and it has no semantic meaning. It is not preserved after the data arrives.

The List resource at the bottom right is a PI List and acts as an index that groups the Document Bundles together into the set that makes up a single ePI.

5.3. Scenarios and lifecycle

This section describes scenarios of the ePI lifecycle and how they would be implemented with FHIR resources.

5.3.1. Scenario: creation with SmPC, package leaflet and 2 labelling entries

In Figure 5., each orange Bundle represents one whole document, whose internal structure is described earlier (see Figure 2).

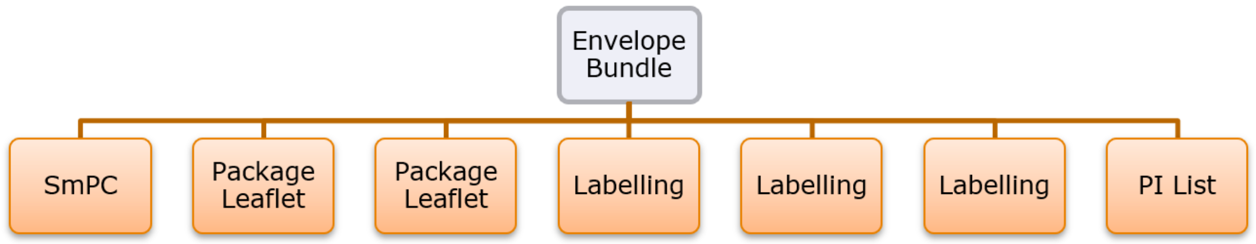

Figure 5. An ePI set of documents being created.

When creating a new set of ePI documents on the server, an Envelope Bundle is used to contain all the Document Bundles in the set, as well as the PI List (listing all the Document Bundles in the Envelope Bundle).

Points to note for this scenario:

- The Envelope Bundle disappears upon reception at the server, and plays no further part.
- SmPCs include a List resource of MedicinalProductDefinition resources called the Product List. (This is different to the PI List).
- Package Leaflets and Labelling documents also can be linked to their own List of MedicinalProductDefinitions (Product List).
- The submitter has to submit the PI List too in all cases (shown on the far right, Figure 5). The receiving system can validate that this is present and correct.
  - The server will use profiles to check the content: ePI creation and ePI update.
- Creation by submitting a single document is allowed (e.g. to begin creating an ePI where more documents would be added later). The Document Bundle must be accompanied by a PI List in an Envelope Bundle.
- Different strengths of the same "product" could be in one single SmPC or multiple SmPCs. The standard allows that one ePI can have either one or multiple SmPCs. The same applies for the package leaflet.
- Business rules will enforce consistency of MedicinalProductDefinition linking in the different artefacts, according to the business process.

5.3.2. Scenario: creation of individual list entries

When it is not desirable to create all parts of the ePI at once, the previous scenario can be broken down into several steps:

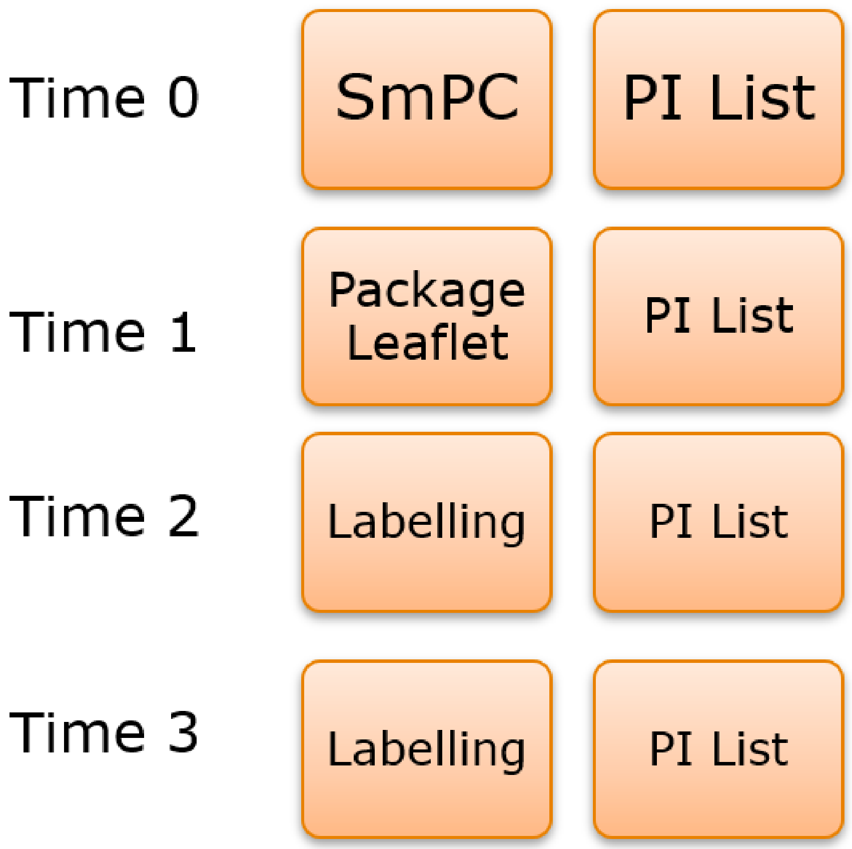

Figure 6. Creating an ePI set of documents in stages.

Here, each document is being created independently. One document per step is shown above, but each step can have one or more documents. The submitter has to submit the PI List together with every submission, and the pair will need to be in an Envelope Bundle. The PI List contains all the documents currently in the set plus any new ones – it always lists the full set of documents at every step.

5.3.3. Scenario: adding a translation for the SmPC and labelling

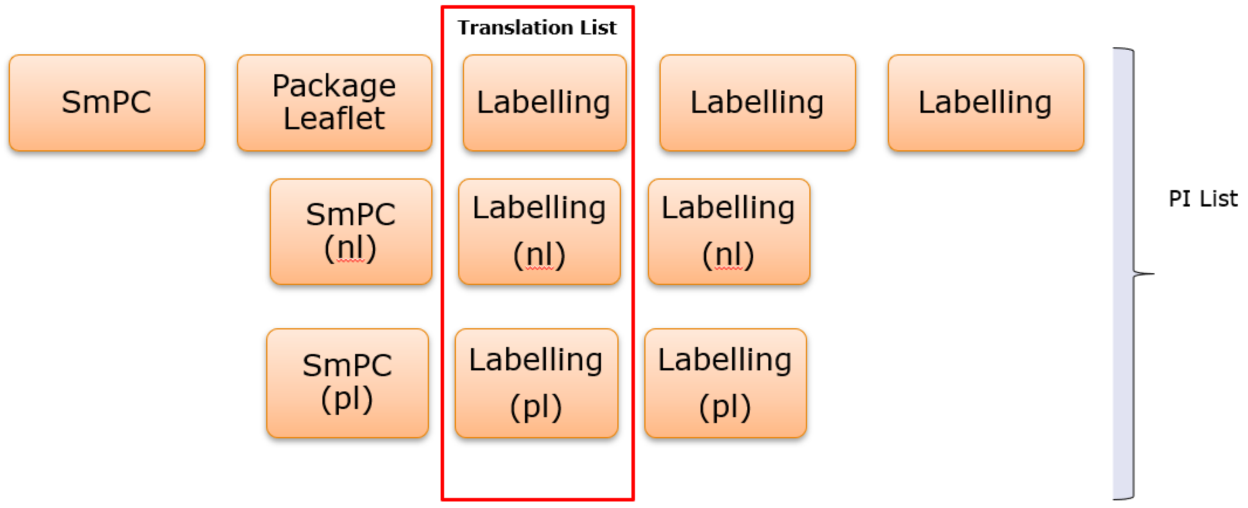

Figure 7. Adding document translations.

In this scenario, an ePI set of documents includes translations of the labelling. This requires an extra List resource, represented by the red outline, called a Translation List. This is in addition to the PI List represented above by the grey grouping at the right-hand side. The diagram shows the "after" state of adding two sets of translations, nl and pl, possibly at different times.

The submitter is responsible for updating both the Translation List and the PI List with references to the Document Bundles. (The scenario assumes that the ePI exists on the server before the translations are added.) The Translation List is a sibling of the PI List – in effect it is a translation index – and is separate from the PI List. It is normal for Document Bundles to be in the PI List and in a Translations List as well.

Technical Note: FHIR Composition resources have a "relatesTo" element, that could be used to link a main document to its translations. However, this API avoids using that because of bi-directional issues when one document is not strictly the master, and also so that all relationships (ePI set of documents index, translations index) are modelled in one homogeneous and portable way: using Lists.

5.3.4. Scenario: updating a package leaflet

An ePI set of documents is created as below, with a POST and with an Envelope Bundle, and gets assigned ids by the server (shown on the two leftmost Document Bundles, but present on all). The mandatory List resource that links these together, the PI List, is shown as the grouping arrow on the right.

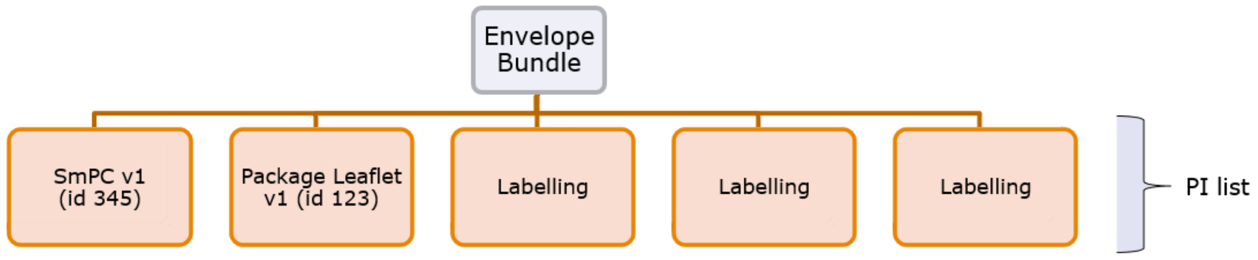

Figure 8. Initial ids are assigned by the server.

In the above, a package leaflet is being sent to the server, as part of an Envelope Bundle of multiple documents (Document Bundles). All of the above items can be sent together. The package leaflet gets an id assigned, in this case: 123.

Note that doing this in one step implies use of a FHIR transaction, because the PI List must know the ids of the Document Bundles, and they are not yet known when the PI List is being sent (see 4.7.2.).

To update a package leaflet (and the same applies to any other document type), a PUT is used. At this point the Document Bundles, including package leaflet 123 already exist on the server:

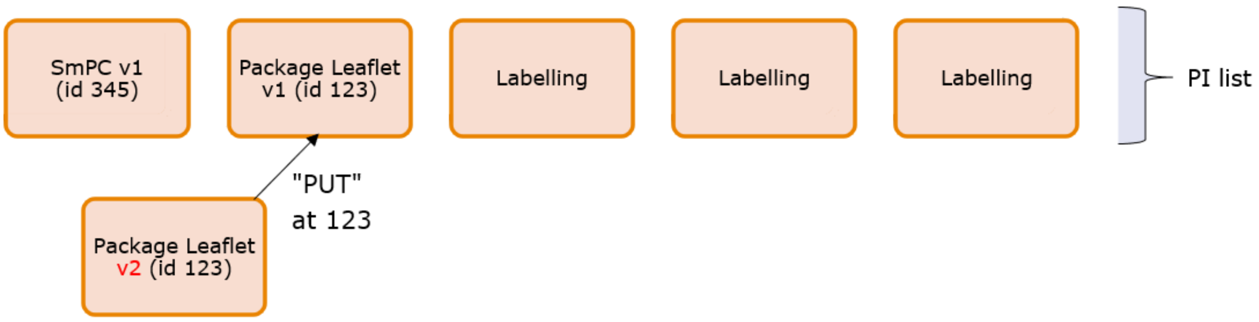

Figure 9. Update process.

The PUT performs an update on item id 123, and replaces the content with what is sent. The package leaflet stays at id 123, but becomes version 2. Since item 123 is already in the PI List, that aspect is unchanged. No update to the PI List is needed. All external links to id 123 will now point automatically to the new version of 123. This can happen again and again. There is no need to group changes in a Bundle or to submit any of the other documents in the original Bundle. The result is this:

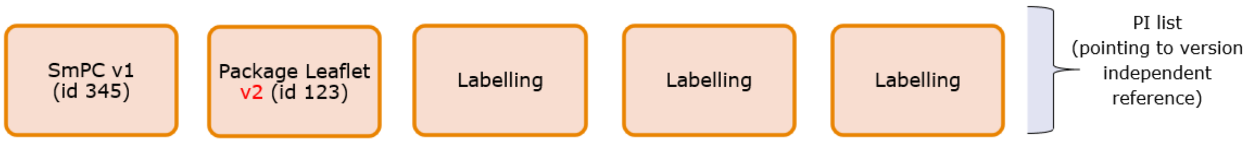
SmPC v1 (id 345)
Package Leaflet v2 (id 123)
Labelling
Labelling
Labelling
PI list (pointing to version independent reference)

Figure 10. After the update.

The SmPC can be updated similarly:

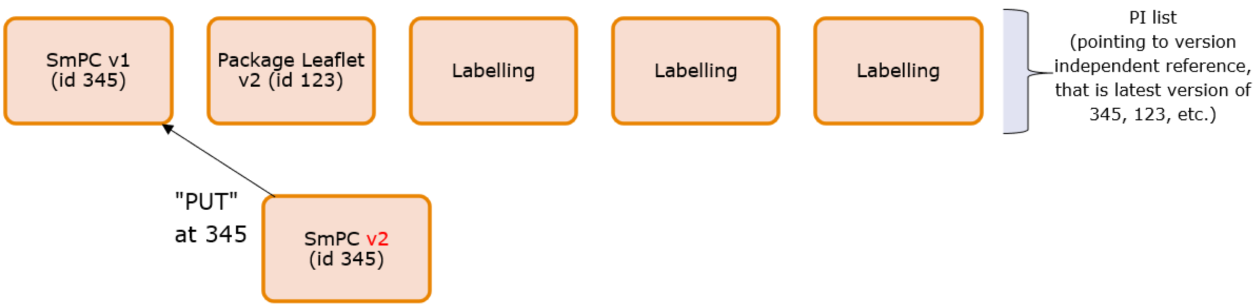
SmPC v1 (id 345)
Package Leaflet v2 (id 123)
Labelling
Labelling
Labelling
PI list (pointing to version independent reference, that is latest version of 345, 123, etc.)
"PUT" at 345
SmPC v2 (id 345)

Figure 11. Update the SmPC.

With this result:

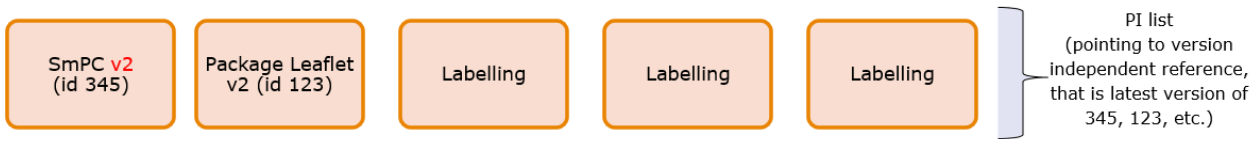

Figure 12. The SmPC has been updated.

5.3.5. Scenario: a new package leaflet and a labelling are added

A set of new documents can be added to an existing one. Here an extra package leaflet and labelling are being sent (POST new elements + PUT of the PI List):

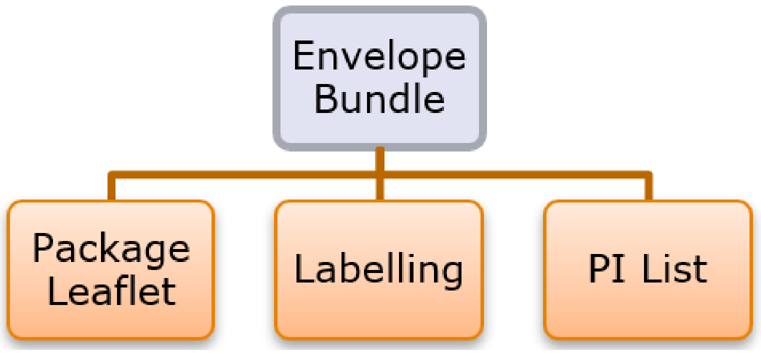

Figure 13. Adding new items to an existing ePI set.

Based on the set from the previous section, the result would be:

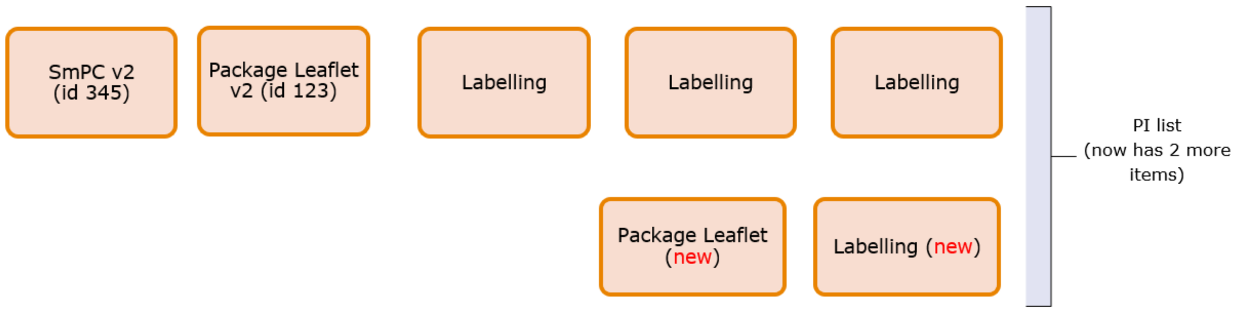

Figure 14. The ePI set with the additions.

5.3.6. Scenario: a labelling is deleted

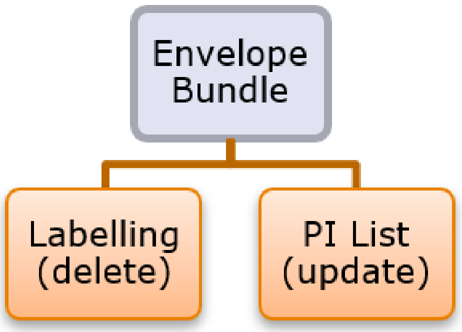

Figure 15. Delete an item.

To delete an item the DELETE http request is used to remove that resource, and the PI List must also be adjusted to remove it from the set. These can both be done in one step using an Envelope Bundle, as a transaction. The updated PI List references the old elements minus the deleted one.

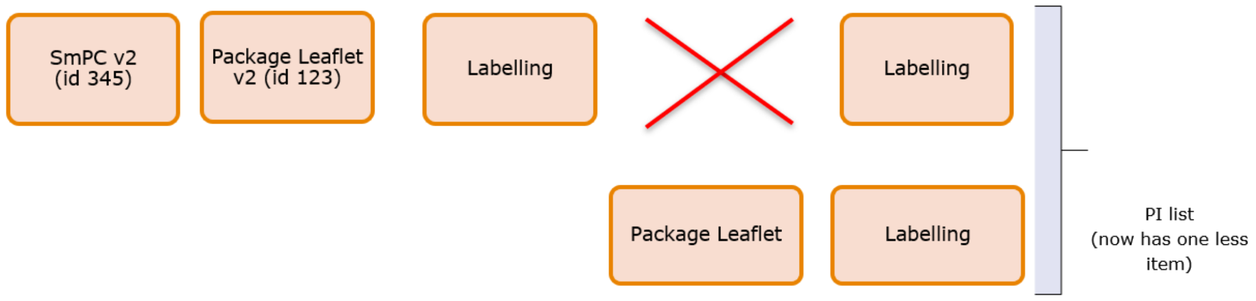

Figure 16. After the deletion.

5.3.7. Scenario: parallel distribution

The FHIR standard and resource structure for ePI offers great flexibility for implementation in varied business processes in the future. Use of ePI for parallel distributed medicines could possibly be accommodated as described here, however further consideration of the particular business process would be required. Parallel distribution involves different PI documentation for the same products. This can be accommodated by different ePI sets pointing at the same products. There is no other link between the two ePI sets.

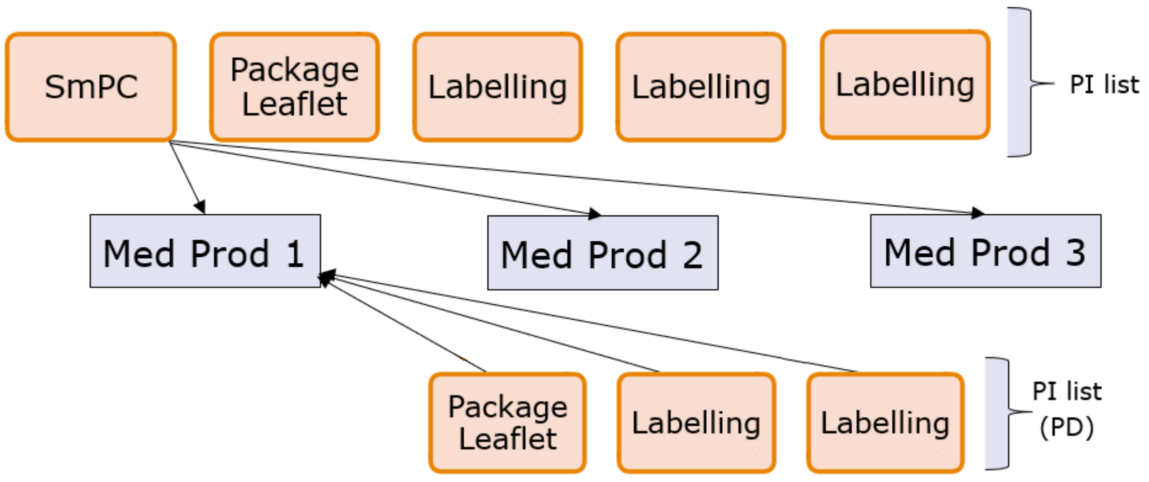

Figure 17. A parallel distribution ePI set points at the same products.

6. REST services

6.1. Resource summary

A summary list of FHIR resources for this API is below (for full description, see 7.). For a summary of how Bundles and Lists are used in this API, see 5.1.

Resource	Description
Composition	Composition is the resource that represents the outline of a document, with metadata - such as the author - and repeating nested sections of HTML text. Sections can optionally refer to other resources for structured detail. Compositions are always contained in Bundles (of type "document"), and it is the Bundle that corresponds to the document as a whole.
Binary	Used for images, that can be embedded in the Composition and referenced from specific places in the HTML text.
List	A set of other resources (as resource references), with a specific coded type or purpose. For use in ePI, see 5.1.
MedicinalProductDefinition	Detailed definition of a medicinal product, typically for uses other than direct patient care (e.g. regulatory use). These appear as references in ePI, but not as full resources.
Bundle	A container for a collection of resources. Several uses in this API (see 5.1). Bundles group together multiple resources - including other Bundles - for search results, and when submitting multiple entities at once.

Other infrastructural resources such as CapabilityStatement and OperationOutcome may also be encountered and are covered in section 7. Resources.

6.2. *Service summary*

The service list is documented in a separate catalogue.

7. Resources

Except where stated, all resources are as modelled as a FHIR resources as documented here: http://build.fhir.org/resourcelist.html

7.1. *Bundle*

http://build.fhir.org/bundle.html and see 4.7.

7.2. *List*

http://build.fhir.org/list.html

Extension:

Canonical URL	http://ema.europa.eu/fhir/extension/domain
Type	Identifier
Extends	List.subject

Example:

```xml
<subject>
      <extension url="http://ema.europa.eu/fhir/extension/domain">
            <valueCoding>
                  <system value="https://spor.ema.europa.eu/v1/100000000004" />
                  <code value="100000000012" />
                  <display value="H" />
            </valueCoding>
      </extension>
      <reference value="List/812a430c-0eda-4615-a30b-a760edaa3598" />
      <display value="Elocta" />
</subject>
```

See next extension.

Extension:

Canonical URL	http://ema.europa.eu/fhir/extension/documentType
Type	Coding
Extends	List.entry.item

Example:

See next extension.

Extension:

Canonical URL	http://ema.europa.eu/fhir/extension/language
Type	Coding
Extends	List.entry.item

Example:

```xml
<item>
      <extension url="http://ema.europa.eu/fhir/extension/documentType">
            <valueCoding>
                  <system value="https://spor.ema.europa.eu/v1/100000155531" />
                  <code value="100000155532" />
                  <display value="SmPC" />
            </valueCoding>
      </extension>
      <extension url="http://ema.europa.eu/fhir/extension/language">
            <valueCoding>
                  <system value="https://spor.ema.europa.eu/v1/100000072057" />
                  <code value="100000072178" />
                  <display value="German" />
            </valueCoding>
      </extension>
      <reference value="Bundle/a156a584-2423-44a7-981b-c83c12e2ee7b" />
</item>
```

7.3. Composition

http://build.fhir.org/composition.html

7.4. Binary

http://build.fhir.org/binary.html

7.5. MedicinalProductDefinition

http://build.fhir.org/medicinalproductdefinition.html

7.6. OperationOutcome

Used in the return from HTTP calls to document errors or warnings.

http://build.fhir.org/operationoutcome.html

7.7. CapabilityStatement

A Capability Statement documents a set of capabilities (behaviours) of a FHIR Server. These will not be exchanged, but the server will expose a read-only CapabilityStatement describing its properties.

http://build.fhir.org/capabilitystatement.html

8. About this document

8.1. Definitions, acronyms, and abbreviations

Acronym/Abbreviation	Description
API	Application Programming Interface
ePI	Electronic Product Information
FHIR	Fast Healthcare Interoperability Resources
GUI	Graphical user interface
HTTP	Hypertext Transfer Protocol
IANA	Internet Assigned Numbers Authority
JSON	JavaScript Object Notation
MAH	Marketing Authorisation Holder
NCA	National Competent Authority
PMS	Products Management System
PSUR	Periodic Safety Update Report
SmPC	Summary of product characteristics
SPOR	Substances, Products, Organisations, Referentials
SSL	Secure Sockets Layer
OMS	Organisations Management System
REST	Representational State Transfer
RMS	Referentials Management System
URL	Uniform Resource Locator
UTF-8	Unicode Transformation Format – 8-bit
UUID	Universal Unique Identifier

Acronym/Abbreviation	Description
XML	Extensible Mark-up Language

8.2. Open issues

None.

8.3. Document approval

Date	Version	Submitted by	Approved by	Approve role
22 September 2021	1.2	ePI set-up project to EU NDB	EU NDB	EU NDB

8.4. Document history

Version	Who	What
1.0	EMA	Creation for proof-of-concept and consultation in ePI set-up project
1.1	EMA	Extension domain moved from List.entry.item to List.subject. Extension medicine-name added. Typo fixed in Resource summary. Fictional product used in the examples instead of commercial brand.
1.2	EMA	Updates following public consultation.