XML schemas

Explanatory Notes

On this page we find the XML schemas of the lists that are made available by *MedMij Management* to *Source* and *Publisher* for a range of purposes. The XML schemas are an implementation of the logical models of the lists in XML syntax and accordingly fulfil the role of technical model. XML fits the hierarchical structuring that has already been deployed in the logical models. In addition, XML schemas and XML files are serial in nature. In other words, the translation from the logical model requires the classes to be placed behind each other without allowing their diagrammatical organisation in the logical model to disappear. A composition relationship in the logical model becomes a nesting in the XML schema. XML tags are used in order to be able to mutually separate the model elements that have been placed behind each other, both in the XML schema and in the XML instance, and to provide the elements with meta information.

Just like they do on the conceptual level of the meta model and on the logical level of the logical model, invariants appear on the technical level too. XML is even able to check some of these invariants automatically. In such XML validation, it is checked whether a certain XML file complies with the structure of a certain XML schema. The MedMij Framework utilises this feature too by requiring the recipient of the four lists to carry out such a validation. The XML schemas for this are made available as part of the MedMij Framework. This validation provides additional certainty about the correctness of the distributed lists and in this way helps to improve the reliability of the MedMij network's operation.

All the same, there are still various ways to translate the lists' logical model into their XML schemas. In the MedMij Framework, the following considerations have been used in this regard:

- All types and elements that are used for one of the lists are defined in the XML schema of the list in question. In other words, no use has been made of a basic common sheet. In this way, the dependency between the XML schemas is limited and it becomes easier to modify one of the schemas without having to modify the other schemas too. However, the definitions must continue to fit the meta model and the logical model; a modification in one of these models makes it necessary to modify all XML schemas that are affected by this change.
- There are four technical components that are associated with the lists' logical model. The highest class of each component becomes the root element of the relevant XML schema. The attributes of the abstract class above it (*MedMijManagementlist*) are distributed across the technical models of these four. In other words, for each list there is a separate XML schema. This means that the homonymy of Dataservice is no longer a problem and that the suffixes _CPL and _DSNL can be omitted from the names.

- Just like in the step from the meta model to the logical models, the granularity of the classes remains the same: no classes are taken together to make a sheet more compact.
- Each of the logical classes, apart from the class that serves as the ‘root’, are defined individually as `complexType` in the XML schema, so that they can be reused within the XML schema.
- Each of the basic classes is defined individually as `simpleType` in XML schema, so that they can be reused within the XML schema.
- All classes and attributes from the logical model are modelled as elements in the XML schema. This enables there to be a clear translation of the logical model; no distinction between elements and attributes needs to be applied. Elements provide more options than attributes do, which is why (as a generic choice) they are preferred.
- Where reference is made in the logical model to ‘identifiers’, a 'uniqueness constraint' has been included in the XML schema.

Sheets

List	Filename	Release	Version of file
Care Providers List	MedMij_Careproviderslist.xsd	2	5
Whitelist	MedMij_Whitelist.xsd	2	9
OAuthclientlist	MedMij_OAuthclientlist.xsd	2	5
Data Service Names List	MedMij_Dataservicenameslist.xsd	1	7

Only the above-mentioned files, with the stated release and version number, may be used in this release of the MedMij Framework.

Sample files (XML)

A sample file of each list is available. This file is not part of the official specifications of the MedMij Framework.

List	Filename	Version of sample file	Belongs to XML schema of the list with release number
Care Providers List	MedMij_Careproviderslist_example.xml	2	2
Whitelist	MedMij_Whitelist_example.xml	5	2
OAuthclientlist	MedMij_OAuthclientlist_example.xml	2	2

Data Service Names List	MedMij_Dataservicenameslist_example.xml	2	1

Explanatory Notes

Time aspect

The meta model and the logical models, with their invariants, work “over time”. They describe how the classes are related at each moment. However, the XML files for the lists are specific snapshots in time of the classes’ instances. This is why a time element must be added to those lists that are generated at different moments, in order to be able to distinguish between them and in order to be able to retrospectively establish a list’s validity duration.

- Each XML file has an indication of the version it is. The combination of a Serial number and a Timestamp is used for this. This fulfils three information requirements:
    - When two lists (of the same type) with successive Serial numbers are available then the validity duration of the older list can be established. This helps in the interpretation of audit logs and in error tracking.
    - Lists can be uniquely identified. This can be done using Serial number or Timestamp, whereby human users often find the Serial number to be the most intuitive to use.
    - It can be verified for each list as to when the last change was made. This will usually be a 'functional' change, not an error recovery. By comparing successive versions, this can be used to deduce when the current list was most recently amended; this can be useful when assessing the effects of changes or when error tracking.
- Timestamp consists of Date, Time and Timezone indication, based on xs:dateTime type. Opting for a native XML datatype simplifies the implementation. However, there is a restriction on the element, which forces a Timezone indication to always be provided.

Release Management

The filenames of the XML schemas and XML sample files have been chosen such that they do not change if the content of the XML schema changes. This makes it easier to implement changes. It is customary to include meta-information not in the filename but in the XML files themselves (especially in the header). This is why it is not necessary to use - in addition to the information in the file - the filename for version indication too.

Each of the XML schemas has its own release numbering. This enables them to be modified independently of each other. This prevents implementation being an unnecessary burden when a change is made. The release number is a whole number, for reasons of simplicity. Always - but only if - a XML schema is changed is the release number increased by one.

The XML schemas are an integral part of the framework. As a result, a change in the XML schemas leads to a new release of the Framework. However, in the reverse situation it does not need to be the case that a change in the other agreements within the Framework makes it necessary to change the XML schema.

Since a change in a XML schema quickly leads to incompatibility with other versions (note that XML files that are based on different versions of the XML schema will not be validated by the 'other' XML schema), it has been decided to include the release number in the indication of the namespace. This is why a XML file's reference to the namespace also includes the release number. In this way it is ensured that XML files are not validated using a wrong version of the XML schema.

The XML schemas and the sample XML files are also given a version number. The version number is a whole number that is increased by one upon each change in the file. Version numbering is used to distinguish between file versions during the development process. The number is also present in production versions; this makes it unnecessary to modify the XML products when there is a status change to a release of the MedMij Framework, even if their content has not changed. The version number is included in the file as a comment, because it does not need to be machine-readable and because this creates a clear system for the XML schemas and the XML sample files. The comment has the form: <!--File version: [version number]--> and can be found on the second line of a file. For reasons of simplicity and clarity, the version numbering is independent of the release numbering for the XML schemas.

Namespaces

An URL is used to indicate namespaces. This is the easiest option, because this - unlike with an URN - does not require any namespace registration with IANA. The namespace URL has the following structure: xmlns://afsprakenstelsel.medmij.nl/[nameList]/release[releasenumber] .

- A namespace URL uses xmlns:// as the sheet indication. This makes it clear that this is merely an identification and that the URL is not intended to be used for dereferencing (for example to download the XML schema).
- The domain afsprakenstelsel.medmij.nl is a unique hostname on the Internet. Using it provides both sufficient recognisablity and uniqueness
- The nameList has one of the following values: Whitelist, OAuthclientlist, Care Providers List or Data Service Names List.
- The indication release is added to make it easier for people to read and hence to improve clarity.

Where the meta model has not defined any names, for reasons of consistency and elegance we opt to use lowercase for the URL's structure. The following is used here: elementFormDefault = "qualified". This makes the XML schemas easier to read, because no prefixes are needed to define the elements and because it does not impair any functionality. The prefixes for the namespaces are kept as short as possible in order to make the XML schemas easier to read, and always consists of three letters and are entirely in lowercase. The following table shows which prefix is used for which component (list).

Component	Prefix
Data Service Names List	dnl
OAuthclientlist	ocl
Whitelist	whl
Care Providers List	cpl

Syntactic options

The XML schemas are based on XML 1.0 and XML Schema 1.0 (built up from specifications relating to structure and data types). These versions provide sufficient functionality and enjoy widespread implementation and support.

The filename of a XML schema has the structure MedMij_[nameList].xsd. The variable nameList relates to one of the following values: Whitelist, OAuthclientlist, Care Providers List or Data Service Names List.

The XML schemas contain the XML Declaration <?xml version="1.0" encoding="UTF-8"?>. The presence of a declaration is recommended by XML 1.0. When using UTF-8, the encoding is optional. However, the encoding is explicit because it prevents potential uncertainty about the intention of or the correct compliance with the specifications. No use is made of the pseudo-attribute standalone, because XML schemas are used instead of DTDs.

To make them easier to read, the XML schemas are pretty-printed; readability is further improved by using line breaks and indents. Furthermore, each XML schema uses a standard sequence in its structure:

- The root element, preceded by the comment text <!--Rootelement-->.
- The definition of the logical classes, preceded by the comment text <!–Logical classes-->.
- The definition of the basic classes, preceded by the comment text <!--Basic classes-->.

There is leeway within them as to the order in which the classes are defined.

The uniqueness constraints use <xs:unique>. The (mandatory or other) name of uniqueness constraints in XML is built up in line with Unique_[nameClass]. In this way, the characteristic of the attribute Hostname of the class MedMijNode from the logical model to which the whitelist belongs translates into a uniqueness constraint with the name Unique_MedMijNode. It is sufficient to have the name of the class (without the hierarchical context), because class names based on the logical model are unique. The name of the attribute does not need to be quoted. The attributes that between them form the identity of a class’s instance are depicted in the logical model . Within <xs:unique>, only <xs:selector> is used for the XPath expression; <xs:field> is included (in accordance with the XML specification) but left empty (has the entry . (full stop)). This is an easier option than having to provide a criterion for the splitting of the XPath expression between <xs:selector> and <xs:field>.

Use is made of <xs:sequence> within all complexTypes, with <xs:all> not being used here, because in this way elements can be used more than once. This is a characteristic that is used a great deal; it is inherent to the nature of the lists and is relevant to many of the composition relationships (who do not have any maximum scope for the compilation).

The XML schemas do not contain a Byte Order Mark. According to XML 1.0, using a Byte Order mark is optional for UTF-8. RFC 3629, chapter 6, argues that the Byte Order Mark must be prohibited where UTF-8 is made mandatory.

Basic classes

The definition of the basic classes in the logical model is translated into simpleTypes in the XML schema, which builds on a native XML data type and which sometimes attaches additional restrictions to it.

Basic class	Basis (XML data type)	minLength	maxLength	pattern
Backchanneluri	xs:string			https://(([a-z0-9])([a-z0-9-])*(\.))+([a-z0-9])([a-z0-9-])*([a-z0-9])(:(\d){1,5})?(/[^?#/]+)*
DateTime	xs:dateTime			.{20,}
Frontchanneluri	xs:string			https://(([a-z0-9])([a-z0-9-])*(\.))+([a-z0-9])([a-z0-9-])*([a-z0-9])?(/[^?#/]+)*
DataserviceId	xs:string	1	30	
Hostname	xs:string			(([a-z0-9])([a-z0-9-])*(\.))+([a-z0-9])([a-z0-9-])*([a-z0-9])
OAuthclientOrganisationname	xs:string	3	50	
Positive number	xs:positiveInteger			
System role code	xs:string	1	30	
Depiction name	xs:string	3	50	

Care Provider Name	xs:string	10	57	([a-z])+@medmij