# 2MVG2-2RIM2

# XML Interface Format

# 1 Introduction, scope & objectives

We refer to the general project results (see document “guidelines”) to correctly understand the scope and role this message format plays in the overall project guidelines.

This document defines the XML Message Format as follows:

- The syntax of the data in the XML Message Format is formally defined by an XML Schema document, named “rimvg2v3.xsd”.
- The semantics of the data in the XML Message Format is defined by the Conceptual Model. We do not repeat the semantics in this document, but refer to “UML Conceptual Model – Description”.
- The behavioral aspects of the XML messages are described further in this document.

# 2 Message format

The formal syntax of the data in the XML Message Format is defined by the XML Schema named “rimvg2v3.xsd”.

This format is illustrated by the UML representation of the message format, found in “XML Message Format - Diagram”.

The semantics of the fields in the message are described in “UML Conceptual Model – Description – en”.

# 3 Message principles

The XML messages respect the following principles.

## 3.1 A single message format

The XML Message Format defines exactly one message that can be used to exchange all the required information.

The message can contain clinical data (Acts & Facts) as well as administrative data (Patient & Stays). The contained elements are optional, so that is perfectly legal to create a message that only contains administrative data and another message that contains clinical data.

The main entities of a Message are:

- Patient
- HospitalStay
- Act
- Fact
- Plan

## 3.2 Idempotent messages

Messages were designed to be idempotent. This means that the meaning of the transferred information is NOT dependent on the order in which the messages that convey the information are processed.

This means that you can send the messages in any order without influencing the final result.

In order to process the messages in an idempotent way, you should respect the following:

- The main entities have a Sys, Ref and RefVersion. The Ref uniquely defines the entity in the sending system Sys. The RefVersion makes the version of this entity unique: it is indeed possible that the first name of Patient "Dupont" is updated in the sending system. A single Ref will exist, with two RefVersions. Hence:
    - Make sure that Ref is unique in the sending system for every entity
    - Make sure that the RefVersion timestamp is updated every time the entity changes in the sending system.
- Some main entities have Uid and UidVersion. Sys is meaningless to these main entities. The same principles apply, ignoring the sending system. Those entities (Patient, HospitalStay, …) have identifiers that are unique over all sending systems, i.e. unique for the institution.

## *3.3 Transactions*

By definition the processing of every main entity in the message should be atomic, i.e. be part of a single (logical) transaction.

The state of the processing of a complete message, containing several main entities, may then be:

- NotProcessed. No single main entity has been processed completely, i.e. no transaction in which a main entity is processed, has completed.
- PartiallyProcessed. Some, not all main entities have been processed completely.
- CompletelyProcessed. All main entities have been processed completely.

Note that partially processed messages can exist for two reasons:

- During the processing of a message, some work has already been done; other work is still waiting.
- Some main entities cannot be processed completely because of errors. In this case, correct the message and replay it; messages are idempotent.

To cope with the latter case, a Message also has the Sys, Ref and CreationDate attribute. This allows to recreate the same message (a message with the same Ref), a second time (with other CreationDate).

## *3.4 Minimal and complete messages*

Messages were designed to be "minimal". Most entities and attributes are optional. Leaving this information out of a message strongly reduces the message volume.

The default value for the entities and attributes is chosen, such that it is the most common value.

On the other hand, we have defined that main entities are always updated completely. It is not possible by a message to update some aspects of a main entity and leaving other aspects a their old values.

Messages containing null values for optional attributes, if not defaulted, update the optional attributes: they erase the existing value.

## *3.5 Patient merges*

In general a mechanism is foreseen to cope with patient merges or merges of hospital stays in an easy way. We illustrate the basic principle by example here, and describe the details later in the document.

When a patient is admitted to the hospital, an HospitalStay is associated with the Patient. When it turns out later that this patient already exists, the HospitalStay can be reassigned the correct Patient. This reassignment can be:

- Immediate, by updating the HospitalStay.
- Delayed. It is possible to “delete” the first Patient. The corresponding HospitalStay’s are not deleted, but orphaned: they refer to a Patient that no longer exists, which – by definition – means that they do not refer to a Patient any more, but still exist. Afterwards, the new Patient can be assigned.

# 4 Message details

## 4.1 Patient

### 4.1.1 Basic operations

#### 4.1.1.1 Insert

When a Patient main entity is processed and the Uid does not yet exists, the Patient is inserted.

All attribute values are assigned a value. Missing attribute values lead to NULL values.

#### 4.1.1.2 Update

When a Patient main entity is processed and the Uid already exists, and isDeleted=”false”, the Patient is updated, if the existing information is older than the Patient.UidVersion timestamp in the message.

All attributes values are updated with the new values, specified in the message. If a value is not specified in the message, it is updated to NULL.

#### 4.1.1.3 Delete

When a Patient main entity is processed and the Uid already exists, and isDeleted=”true”, the Patient is deleted, if the existing information is older than the Patient.UidVersion timestamp in the message.

When a Patient is deleted, it is still possible that ContextPatient entities reference the deleted Uid. The meaning of the ContextPatient entities is as if they were NULL. We say these ContextPatient entities are orphaned.

### 4.1.2 Other operations

Intentionally empty.

### 4.1.3 Operations that lead to orphans

Intentionally empty.

## 4.2 HospitalStay

#### 4.2.1.1 Insert

When a HospitalStay main entity is processed and the Uid does not yet exists, the HospitalStay is inserted.

All attribute values are assigned a value. Missing attribute values lead to NULL values.

The ContextPatient entity is inserted as well, if present.

All depending UnitStay entities are inserted as well.

#### 4.2.1.2 Update

When a HospitalStay main entity is processed and the Uid already exists, and isDeleted=”false”, the HospitalStay is updated, if the existing information is older than the HospitalStay.UidVersion timestamp in the message.

All attributes values are updated with the new values, specified in the message. If a value is not specified in the message, it is updated to NULL.

The ContextPatient entity is updated as well. If not present is it set to NULL.

The existing UnitStay entities are deleted and all depending UnitStay entities are inserted.

#### 4.2.1.3 Delete

When a HospitalStay main entity is processed and the Uid already exists, and isDeleted=”true”, the HospitalStay is deleted, if the existing information is older than the HospitalStay.UidVersion timestamp in the message.

When a HospitalStay is deleted, it is still possible that ContextHospitalStay entities reference the deleted Uid. The meaning of the ContextHospitalStay entities is as if they were NULL. We say these ContextHospitalStay entities are orphaned.

The following main entities may become orphans, when a HospitalStay entity is deleted:

- Act
- Plan

### 4.2.2 Other operations

#### 4.2.2.1 Reassign Patient

Act entities are logically associated to HospitalStay entities via a ContextHospitalStay. Indirectly, are thus Act entities associated with Patient entities: the HospitalStay entity is associated to a PatientEntity via a ContextPatient entity.

To assign all Act entities of a single hospitalization, simply update the HospitalStay entity to another ContextPatient entity. Do not forget in this case to provide also all attributes of the HospitalStay entity and all UnitStay entities.

## *4.3 Act*

#### 4.3.1.1 Insert

When an Act main entity is processed and the Sys, Ref combination does not yet exists, the Act is inserted.

All attribute values are assigned a value. Missing attribute values lead to NULL values.

The ContextHospitalStay entity is inserted as well, if present.

All depending Aspect, Fact, ContextPlan and ContextAct entities are inserted as well.

#### 4.3.1.2 Update
When a Act main entity is processed and the Sys, Ref combination already exists, and isDeleted=”false”, the Act is updated, if the existing information is older than the Act.RefVersion timestamp in the message.

All attributes values are updated with the new values, specified in the message. If a value is not specified in the message, it is updated to NULL.

The ContextHospitalStay entity is updated as well. If not present is it set to NULL.

All depending Aspect, ContextPlan and ContextAct entities replace the existing values.

The depending Fact entities are updated, if the Act.RefVersion is more recent that the Fact.LastlyReportedDuring.RefVersion. See later how a Fact is updated.

Note that it is possible that Act main entity is not updated, but the depending Facts are. This would be the case if the existing information in the Act main entity is more recent than Act.RefVersion, but there exists depending Facts whose Fact.LastlyReportedDuring.RefVersion is less recent. This scenario is rare and will only occur if a Fact is sent in the context of different Act.Refs.

#### 4.3.1.3 Delete
When an Act main entity is processed and the Sys, Ref combination already exists, and isDeleted=”true”, the Act is deleted, if the existing information is older than the Act.RefVersion timestamp in the message.

When an Act is deleted, it is still possible that ContextAct entities reference the deleted Ref. The meaning of the ContextAct entities is as if they were NULL. We say these ContextAct entities are orphaned.

The following main entities may become orphans, when an Act entity is deleted:
- Act

### 4.3.2 Other operations
Intentionally empty.

## *4.4 Fact*

#### 4.4.1.1 Insert
When a Fact main entity is processed and the Sys, Ref combination does not yet exists, the Fact is inserted.

All attribute values are assigned a value. Missing attribute values lead to NULL values.

All depending Aspect entities are inserted as well.

#### 4.4.1.2 Update
When a Fact main entity is processed and the Sys, Ref combination already exists, and isDeleted=”false”, the Fact is updated, if the existing information is older than the Fact.LastlyReportedDuring.RefVersion timestamp in the message.

All attributes values are updated with the new values, specified in the message. If a value is not specified in the message, it is updated to NULL.

All depending Aspect entities replace the existing values.

#### 4.4.1.3 Delete

When a Fact main entity is processed and the Sys, Ref combination already exists, and isDeleted=”true”, the Fact is deleted, if the existing information is older than the Fact.LastlyReportedDuring.RefVersion timestamp in the message.

### 4.4.2 Other operations

Intentionally empty.

### 4.4.3 Notices

Facts always occur in the context of an Act during which the Fact was lastly reported (see Fact.LastlyReportedDuring relationship).

Facts however exist as independent entities. This means that,

- In order to delete a previously reported Fact, it is not sufficient to “delete” the Act during which this Fact was reported. The Fact itself must be “deleted”.
- The same Fact can be reported by several Acts. Only the Act during which the fact was lastly reported will eventually be remembered (this rule guarantees idempotent message semantics).

## *4.5 Plan*

#### 4.5.1.1 Insert

When an Plan main entity is processed and the Sys, Ref combination does not yet exists, the Plan is inserted.

All attribute values are assigned a value. Missing attribute values lead to NULL values.

The ContextHospitalStay entity is inserted as well, if present.

#### 4.5.1.2 Update

When a Plan main entity is processed and the Sys, Ref combination already exists, and isDeleted=”false”, the Plan is updated, if the existing information is older than the Plan.RefVersion timestamp in the message.

All attributes values are updated with the new values, specified in the message. If a value is not specified in the message, it is updated to NULL.

The ContextHospitalStay entity is updated as well. If not present is it set to NULL.

#### 4.5.1.3 Delete

When a Plan main entity is processed and the Sys, Ref combination already exists, and isDeleted=”true”, the Plan is deleted, if the existing information is older than the Plan.RefVersion timestamp in the message.

When an Plan is deleted, it is still possible that ContextPlan entities reference the deleted Ref. The meaning of the ContextPlan entities is as if they were NULL. We say these ContextPlan entities are orphaned.

The following main entities may become orphans, when a Plan entity is deleted:

- Act

## 4.6 CaregiverAssignment

#### 4.6.1.1 Insert

When a CaregiverAssignment main entity is processed and the (CaregiverUid,At) does not yet exists, the CaregiverAssignment is inserted.

All attribute values are assigned a value. Missing attribute values lead to NULL values.

All depending CaregiverUnitAssignment entities are inserted as well.

#### 4.6.1.2 Update

When a CaregiverAssignment main entity is processed and the (CaregiverUid,At) already exists, and isDeleted=”false”, the CaregiverAssignment is updated, if the existing information is older than the CaregiverAssignment.UidVersion timestamp in the message.

All attributes values are updated with the new values, specified in the message. If a value is not specified in the message, it is updated to NULL.

The existing CaregiverUnitAssignment entities are deleted and all depending CaregiverUnitAssignment entities are inserted.

#### 4.6.1.3 Delete

When a CaregiverAssignment main entity is processed and the (CaregiverUid,At) already exists, and isDeleted=”true”, the CaregiverAssignment is deleted, if the existing information is older than the CaregiverAssignment.UidVersion timestamp in the message.

# 5 Change Log

This section lists the most important changes that have been applied, since the initial publication of the XML interface format. They are based on feedback from pilots and sector.

## 5.1 Order of elements in <message>

The order of the <Patient>, <HospitalStay>, <Act> and <Plan> elements in the <Message> element, is now free. Before there was a strict order defined. There is in fact no need for a fixed order as no semantics are associated with the order. When the order is free, it is easier to generate the <Message> file.

## 5.2 <DataTypeCode> and <ArchetypeId> are no longer mandatory in the <Aspect> and <Fact> elements

<DataTypeCode> and <ArchetypeId> convey the data type information of the value in <Aspect> and <Fact>.

Originally, we assumed that this data type information should be repeated in each message.

Now, we have relaxed this assumption. The receiving system (provided it is properly configured) can also know the data type information based on the <AspectCode> and Fact<Code>. Every <AspectCode> and Fact<Code> must always receive values of the same data type. The data type information can then be stored beforehand in the receiving system and need not be in the message itself.

Data type information can still be conveyed in the message. Then the receiving system can do a type check when importing a message.

## *5.3 <FactTypeId> is no longer mandatory in the <Fact> element*

The same arguments hold as for 5.2.

## *5.4 <Plan> is updated*

The element <Plan> has been updated according to the changes in the conceptual model. Plan is now adapted to the requirements of version 1.3 of the coding guide.

## *5.5 Registration manual (version November 2006)*

The elements are modified according to the needs of this manual.