Schema Versioning Proposal

Purpose

This document is a proposal for the design and implementation of schema versioning for the ALMA project. It is the first deliverable of the ALMA Schema Versioning Function Based Team (VFBT hereafter). After its release the document has to be reviewed and approved by all subsystem leads, since the propsed solution affects all ALMA subsystems and at least some of the proposed solutions have to be implemented by each of the subsystems. After approval this document will be turned into a design and implementation guideline for the implementation of schema versioning.

Scope

Schema versioning is a, although very common, quite complex problem for almost any project dealing with longer term data interfaces and persistent storage. It applies to classical relational database schemas as well as interface data formats and probably most prominently in the usage of XML as a descriptive data transport and document markup language. XML is regarded as a flexible, yet descriptive way to describe contents, both type-safe but also with some semantic outline. Since neither data nor semantic descriptions and relations are stable over a longer period of time we need to find solutions to be able to deal with the differences of the XML schemas and the derived instance documents. In ALMA XML schemas are used to describe observing projects and all the related entities, as well as the project status. This structure of XML schemas is derived from a UML model which is called the ALMA Project Data Model (APDM). Similarly the data collected during the execution of an observing project is described using XML schemas derived from a UML model called the ALMA Science Data Model (ASDM). In addition XML is used for the

transfer of the ACS wide logging information and to describe the processed data entities from the pipeline. This proposal describes a system to enable ALMA to develop and use different versions of XML schemas and more importantly instance documents throughout the ALMA data flow by marking both the schemas and the instance documents with version information and using this information to switch to the appropriate software or to transform older instance documents to comply to a more recent version or at the very least to allow the software to easily detect whether it would be able to deal with a certain schema version by just analyzing the root element of an instance document.

This proposal does not cover schemas or instance documents which are created on-the-fly or ad-hoc like the result sets from queries, in particular XQuery operations. The proposal does not cover either the orthogonal problem of versioning (i.e. updating) of instance documents during the lifecycle of the instance document, although we are proposing to add the `documentVersion` attribute to every root element of instance documents in order to be able to trace such changes.

Reference Documents

There is a set of document links listed in the literature section of the SchemaVersioning FBT twiki page.

Definitions and Conventions

- Schema documents or in short schemas: These are the XML documents defining the structure and semantics of **instance documents**. In ALMA there are approximately 100 different schemas in use where some of them are just defining types rather than **instance documents** or XML elements. Both the ASDM and the APDM are defining a whole set of inter-related schemas which will in general be versioning as a whole.
- Instance documents: these are XML documents containing the actual data. They have to be compliant to the **schemas** defining them. Standard XML validation should not result in errors. This also implies that the instance documents have to be well formed according to the XML standard.
- Elements and attributes: Just short for XML elements and XML attributes, both of them are written like `SchemaVersion` in this proposal.

Requirements

1. The version information shall be propagated automatically from the root of the data model into the XML schema documents and to the instance documents.
2. The version information shall be accessible in the archive. This means that it shall be possible to request instance documents of a certain schema version only.
3. It shall be possible to perform instance document transformations either on-the-fly or persistently to a newer schema version by software as long as a transformation can be described in either software directly or XML stylesheets. Transformations will exist as long as the changes introduced in a newer version are backwards compatible.
4. The versioning introduced here shall not render the already existing instance documents invalid and it shall be possible to still use them after the full implementation of the schema versioning.

Proposed Solution

The proposed solution is a mixture of adopted practices in the XML community, custom attributes and usage (contents) conventions to be followed by the schema developers. In broader terms all ALMA schemas shall comply to the following requirements:

1. a namespace is assigned to the schema and it must remain unchanged whatever the schema version.
2. the schema must be tagged by a major version number.
3. the schema must be tagged by a “revision” identifier, the CVS number returned by a commit.
4. the instance documents must carry the schema version number they comply to.
5. the instance documents must carry all of what is required to know the URL of the schema.

The general rule is that the major version number, an integer, must be incremented by one every time the schema has been modified and if this revision requires a transformation in order to validate older instance documents against the new schema. One of the roles of the CVS identifier, a model comparable object, is to allow to discriminate between backward compatibilities, its value ’increasing’ as the schema is revised. To make the versioning model effective, it must be possible to compare these two numbers in the instance documents to those in the schema. This requires some pre-processing prior to parsing the schema and validating the instance document against that schema. For this reason these numbers are mandatory, not only in the schema itself but also in the instance documents. This versioning model implies that the author of the schema must identify any potential elements which could be root elements at the instance level, this to assign to them the versioning attributes in their definitions in the schema. For example

in the the SDM-BDF there are two useful stand-alone XML instance documents which can be extracted, a) a document with the `sdmDataObject` root element and b) a document with the `sdmDataHeader` root element. For this reason not only the `sdmDataObject` element but also the `sdmDataHeader` element must have the versioning attributes. The versioning model defines 3 attributes, `schemaVersion`, `revision` and `documentVersion`. They are defined in the namespace of that model. In the case of the SDM-BDF only the two first attributes are effectively used.

Schema Versioning in Practice

Attribute usage, convention and dependencies

The ALMA schema versioning proposed here essentially is implemented as a combination between custom attributes and usage convetions of existing attributes. The following table shows the used attributes and their mandatory/optional status in comparison with general XML. In effect the versioning imposes some restrictions on the attribute usage. In addition to being mandatory this proposal also imposes some conventions on the contents and formatting of these attributes. These are secribed in the subsequent sections below.

Attribute usage pattern

attribute	General XML	XML with versioning
xs:version	optional	mandatory
xs:targetNamespace	optional	mandatory
xvers:schemaVersion		mandatory
xvers:revision		mandatory
xvers:documentVersion		optional
xsi:schemaLocation	optional	mandatory

Convention in the context of versioning

The value taken by the xsi:schemaLocation at the instance document level is at least one pair of URIs, an absolute URI for the namespace, the first member, and a relative URI for the schema location, the second member.

Would this attribute have a set of pairs it is the second member of the pair which is the key for the set.

The convention is that when there are several pairs with the same namespace, the first one appearing in the sequence must correspond to the schema where is defined the instance document element. The schemaVersion and revision numbers present in the instance documents are meaningful in the context of that schema only.

Schema dependencies

In general it is not required to have more than a single pair for this xsi:schemaLocation attribute in the document element, the one corresponding to that schema where is defined the document element. The versioning model implies that when there are schema dependencies these must be in a hierarchy relative to that primary schema. With the proposed design the dependency on the versioned schemas necessarily propagates up to the signature of the version of the primary schema.

Using the xsi:schemaLocation attribute with several pairs is beyond the scope of the proposed versioning model. This use-case is mostly useful when using the xs:import in a schema when only its namespace attribute is specified. To support that case would require for the processors to compare versioning numbers for non-root element in the instances with those in the corresponding schemas. In any cases XMLSchema vers. 1.0 does not offer the possibility to formulate the predicates which are needed to assess the versioning constraints.

Version numbers: the attributes `version`, `schemaVersion` and `revision`

The `version` attribute

Every schema has a pair of version numbers assigned via the `version` attribute in the root element of a schema. Since the usage of this pre-defined attribute is not implied by any XML standard, nor is the format constraint at all (it is simply a string), we are free to define our own convention for ALMA:

```
<?xml version="1.0" encoding="UTF-8"?>
<schema ... version="1 1.12">
  ...
</schema>
```

The first member of the pair, an integer number, here 1, must be modified every time the schema is modified with non-backward compatibility. This is the major version number. It is the responsibility of the author of the schema to state whether a modification made to the schema is backward compatible or not. The second member of the pair is the CVS number, here 1.12. This number is set automatically when there is a commit after schema revision. It is recommended to associate comments to this version using the standard environment immediately after this `schema` tag. This could also include a full revision tag from CVS.

The `schemaVersion` and `revision` attributes

All potential root elements of a schema are defined with the pair of attributes `schemaVersion` and `revision`; they are mandatory to enforce the possibility to assess backward compatibility when resolving the schema location based on the information given in the SDM binary data object at the instance level.

Thus at the instance level one must have e.g.:

```
<?xml version="1.0" encoding="UTF-8"?>
<sdmDataHeader ... xvers:schemaVersion="1"
xvers:revision="1.12">
...
</sdmDataHeader>
```

xvers is an alias to the namespace of the versioning model. When creating XML documents with a root element the structure in this case looks like

```
<?xml version="1.0" encoding="UTF-8"?>
<sdmDataObject ... xvers:schemaVersion="1"
xvers:revision="1.12">
<sdmDataHeader ... xvers:schemaVersion="1"
xvers:revision="1.12">
...
</sdmDataHeader>
...
</sdmDataObject>
```

Note that the constraint set by the versioning model leads to a duplication! This is necessary to fulfill the standard practices in XML by having the document element with the `schemaVersion` attribute. These instances with `schemaVersion="1" revision="1.12"` must be valid against the schema with `version="1" revision="1.12"` **and** `version="1" revision="1.13"`, but may not be necessarily valid against a schema neither with `version="1" revision="1.11"` nor with `version="2" revision="1.18"` (**NOTE**: Version and revision numbers are not connected in any way, mainly because we don't want to impose any restrictions on the way the CVS system is used. Since most of the people are always just checkin a new revision we will end up in a sequence of CVS numbers of the form 1.x.).

The namespace of the sdmDataObject schema is a fixed value and will not change from one version to the next. The mapping contained in the `xsi:schemaLocation` attribute allows keeping the namespace fixed while changing the URL and/or the actual schema file name to reflect the version. In addition the major version number is contained in the `version` attribute of the root element.

Namespaces and Schema Locations

The XML world defines and implements a quite extensive and powerful way of mapping namespaces to uniform resource identifiers (URIs) and from there to actual uniform resource locators (URLs). This scheme has mostly been inherited from the SGML world and uses as one way of namespace resolution a catalogue file which in turn can define several possible locations for any schema used in the system. There is also the possibility to define directly a valid URL in the `schemaLocation` attribute of the root element of the instance document or the schema document, as well as in any `import` and `include` element in schemas and `include` in instance documents. In this case there is no need to have a catalog file at all. If a catalog file is used, and this is what is recommended for the ALMA internal environment, the verifying software has to implement the OASIS standard for namespace resolution. In Java for the xerces parser this can be done by using the *resolver.jar* through the convenience class *XMLCatalogResolver* contained in the xerces

package. To use XMLCatalogResolver as an XNI EntityResolver you need to do something like this:

```
import org.apache.xerces.util.XMLCatalogResolver;
import org.xml.sax.*;

...

XMLReader reader;
String [] catalogs =
  {"file:///C:/catalog/cat1.xml", "file:///C:/catalog/cat2.xml"};

...

// Create catalog resolver and set a catalog list.
XMLCatalogResolver resolver = new XMLCatalogResolver();
resolver.setPreferPublic(true);
resolver.setCatalogList(catalogs);

// Set the resolver on the parser.
reader.setProperty(
  "http://apache.org/xml/properties/internal/entity-resolver",
  resolver);

...
```

Note that most of the standard XML tools are directly supporting XML catalog files already and usually come with quite a number of standard schemas already configured and adding additional custom schemas in different locations is also supported. The correct resolution of the schema location in all situations is very important, but also fairly complex as it is implemented to various levels of compliance by different parsers and tools.

Access to Schema Version

Schema access is an important issue to be solved. The URL in the second part of the `xsi:schemaLocation` attribute needs to be accessible and it shall return the requested schema. In full operations this needs to be ensured even for external users somewhere on the network in order to be able to verify XML entities using standard tools. The idea here is to use an official `alma.cl` URL which exposes this part of the Oracle XML DB to the outside world. The problem of access to schemas without a working network connection can be resolved using the XML catalog support of existing parsers and the builtin fall-back mechanisms to access local schemas on the file system. This has been demonstrated already by FrancoisViallefond. The XML catalog is a OASIS standard and has been implemented in many tools and parsers but requires additional attention and maintenance. This is acceptable for the ACS installation, but certainly not for external users of ALMA data.

Loading of Schemas into the Archive

The XML schemas are the interface definitions for the **content** of the data transported between the various subsystems and to external applications. These are the descriptions of the persistent legacy of the ALMA observatory and as such are the only way to understand the data structures and meaning of the measured quantities collected during the observations. Therefore we have to be very careful to design a consistent and complete set of schemas and a rigid system to keep these definitions accessible along with the derived instances and the data. The schema lifecycle in ALMA can be summarized in a few bullets:

1. Data model design or change
2. Schema generation (automatic from the design or manual)
3. Binding class generation (automatic or manual)
4. **Schema registration (this generates the database binding automatically)**
5. **Instance verification using schema**
6. **Whenever a new schema version is registered the instances derived from the current version have to be migrated, that marks the end of a schema lifecycle. We will most likely keep both the original schema version and the instances at least for some time in order to be able to recover from migration errors.**
7. If changes are required go to 1

The points in bold are the critical parts of the schema lifecycle where instances are actually generated and thus the schema can't be dropped anymore. Thus it is proposed here to register all schemas in the ALMA archive, more specifically in the Oracle database

and expose the already existing service of the database to the outside world. The actual URL of this service should be proxied by a persistent web server which is accessible through a very stable URL (i.e. this URL should ideally never change and should always be accessible). This URL could start with `http://www.alma.cl/xml` and point on the backend to the actual Oracle service. In the example the full namespace would then be called `http://www.alma.cl/xml/XSDM/sdmbin`

Instance documents

The XML standard allows for an incredible flexibility. It is possible to construct instance documents containing the same elements compliant to different versions of a single schema. Although this might be a desired feature for XML in general, it is quite questionable within the framework of an integrated project like ALMA, and thus it is not supported by the schema versioning model described here.

As a consequence, for this document it is assumed that for a specific build of the ALMA software there is only one version of any given schema (set) used to create new instance documents. This would be a perfect situation if no instance documents of other versions could come from either different builds of the ALMA software or even other sources. Unfortunately this is the case for instance documents coming from the archive. Since it is foreseen that in particular the (A)SDM could serve as a standard for the interferometric community, even data from multiple observatories could be combined. In general there are two ways of dealing with this: either the software has to be able to understand all versions older than the current one as well, or there must be a mechanism to translate older instance documents to comply to the schema currently used by the software. Here we are sticking to the second option and that requires that all instance documents have to comply to the same version of the underlying schema set. That does not mean that we can not version schemas individually, but it does mean that we have to implement a much more strict configuration control on schemas, very much the same way as we are doing for the subsystem interface definitions (IDL files). Thus it is proposed here to put **the whole set** of schemas (or alternatively the model used to create them) used in a given release of the ALMA software under strict revision control. Although this is mostly done already as a consequence we also propose to register them in the archive during initialization of the archive already and thus disable the dynamic loading of schemas in the **operational** archive. We would still allow full flexibility on the installations of the test archive.

Schema Migration

Such a translation process is called schema migration, though it is not so much about migrating the schemas, but rather the instance documents. Here we have to distinguish among three different classes of schema changes:

- Minor changes where no translation is necessary (e.g. change of a comment in the schema document)
- Major changes where a translation is required, but exists. That means that it is possible to write a XSLT stylesheet to perform the automtic batch translation
- Major changes where a translation is required, but there is no XSLT for automatic batch translation. One example is adding a new mandatory element, where the content of the element can not be derived from the information contained in existing instance documents, e.g. adding a home e-mail address as a mandatory element.

Obviously the last case should be avoided if at all possible. Adding or removing mandatory elements quite often (not always) leads to incompatible changes, which can't be captured by a translation process and leads to a quite complex manual intervention on a potentially very big number of instance documents. Thus here we will deal only with the second case, since the first one is trivial. In general a translation process could span more than one version. For instance the source version could be 2 and target version could be 6. Thus the adopted mechanism should support chaining of translations. For ALMA we propose also the restriction to support only forward translations, i.e. from lower versions to higher ones. Schema migration for XML instance documents is supported by many standard XML tools through the application of XML stylesheets (XSLT). Provided that the XSLTs exist, the migration process can be applied in various places in the ALMA dataflow. Since also Oracle supports this internally we propose for the standard ALMA dataflow to use this built-in functionality to do both on-the-fly migration and static migration. The former is initiated through a special query, while the latter is initiated as a batch job for all documents of a certain schema version. In addition we should provide guidelines and a small tool to drive one of the standard xslt processors on instance documents in the file system.

Schema Verification

The main access and the main source of XML instance documents in the ALMA data flow is the archive. It is also in the archive where all versions of all instance documents have to be stored. Up to now the archive has been completely 'blind' to both schema versions and schema violations. That means that in practice it is currently possible to store whatever instance document in the archive by just referring to an existing schema. Essentially the schema documents are not used, but just stored. With full schema versioning in place this approach has to be adjusted and handling of schemas must be a lot stricter than it is at present. For the instance documents this means that they have to comply to the schemas referenced internally, i.e. we have to switch to full schema verification. As a side effect it will not be possible anymore to store invalid documents. If the schema verification is switched on only

in the archive this could lead to situations where the whole execution of observations went fine but then the final archiving failed. Due to operational pressure this will almost certainly lead to the request to switch the verification off completely. With full schema versioning and migration in place this is not an option anymore, because it could render whole document instance tables inaccessible. Thus schema verification should be switched on very early already during the first creation of an instance and probably done again after major changes. The proper places in the data flow have to be identified since the verification might be an expensive operation and should therefore not be carried out too often. We propose to carry out schema verification every time an instance document is changed substantially (**NOTE**: This is a different version of a single instance document marked by the documentVersion attribute.) and again when the document is ingested into the archive. The performance penalty of this approach has to be verified. In a number of cases schema verification is not sufficient to ensure consistency of the information contained in the instance documents. This includes cases where the allowed values of an element are dependent on the value of another element. Such cases are well known in general in the XML community and are usually covered by additional rules written in Schematron.

Access to Instance Documents

When accessing instance documents we have to be able to allow for the following scenarios:

- Instance documents are accessed either locally on the file system or through the network.
- There are persistent and transient instance documents in the ALMA data flow. Some transient instance documents are made persistent only at the end of their lifecycle (e.g, events).
- By means of more advanced access methods (e.g. XQuery or XSLT) it is possible to generate instance documents on-the-fly. In principle this could lead to instance documents compliant to different versions of the same schema.

In ALMA a lot of code parsing and interpreting instance documents is automatically generated using the schemas and thus any change in the schema in general will also lead to a code change. With code generation the schema binding is usually very strict and even small variations in the schema can't be captured without manual tuning anymore. Thus the schema, the instance documents and the binding code are closely coupled and the translations of instance documents to a common schema (set) has to be done very carefully.

Compliance with Standard XML Tools

In order to be able to reuse existing code both within ALMA and externally the XML schemas, the instance documents and XML techniques used to assign schemas to instance documents have to comply to the W3C standards, recommendations and practices. In order to ensure this we have implemented two demonstrator suites, one as a stand-alone software package, the other one to ensure that we can use our proposed solution together with the XML technologies available in the Oracle database. We have also used some other tools (Oxygen, libxslt, expat, Xerces) to generate XML instances from the schemas or to parse and transform the instance documents. Schema versioning is not very well standardized in the XML community and even best practices vary depending on the authors. The XML tools thus don't support versioning as a fundamental function, but rather the developers of an XML based system have to choose their way of solving this problem. In particular there is no common solution for an automatic versioning of either schemas or instance documents and even less of a solution for keeping every single document of a big collection of instance documents in an operational system connected to the correct schema and on top of that allow interoperability between the instance documents.

Implementation Guidelines

In general implementers shall make sure that they are using the version/revision information contained in the instance documents and if necessary also in the schemas in order to verify whether their code is able to deal with that particular version/revision. Since all what's necessary is contained in the root element of every instance document this does not even require to parse the whole document, but just finding the `schemaVersion` and `revision` attributes in the root elements and compare them to the values acceptable for the software. Obviously the software developers can implement support for more than one schema version in the same software if so required, but in particular generated code will in general not be able to deal with more than one version.

Developers should also be cautious when working in an environment where there is a potential of receiving a whole set of instance documents which may be compliant to different versions, e.g. as a result of an archive query. In this case the compliance of the software with the returned versions of instance documents should be checked individually and warnings should be logged for the non-compliant ones only, but the compliant ones should still be treated correctly.

The archive will implement additional interfaces to allow to query for instance documents of a certain version only, we will also work on the implementation of an on-the-fly transformation of instance documents, provided that we will receive the required XML stylesheets for the conversion from the maintainer of the schema. In addition we will support batch transformations, i.e. it will be possible to do a full schema migration of all instance documents from one version to the next.

Schema Creation

Schemas in ALMA are either generated from a model or they are written using an XML modeling tool or completely manually. The proposed changes affect only the schema header section and the possible root elements. Thus the implementation of the required changes should be fairly straightforward in either of the cases described below.

Generated Schemas

Schemas are generated using the (UML) model driven open architecture ware (OAW). The definition of attributes and elements is mostly done directly in the UML model, while the actual logic of the generation process is contained either in template code or in Java. The current way of generating the APDM schemas is shown in the appendix. The required changes can be deducted from the attached APDM set.

Hand Written Schemas

For the cases where schemas are written by hand the requirements above have to be applied directly in the schema file. The necessary changes are quite minor and are mainly to introduce the `version` attribute with compliant contents into the root element of the schema and to define the `schemaVersion` and `revision` attributes in each potential instance root element.

Generated Binding Code

The code used to parse and interpret the instance documents in Java mostly is generated automatically using Castor. The classes produced by Castor are very tight mappings between the schema and the Java object model. What is relevant for schema versioning in this respect is that the versioning information has to be maintained and also analysed by the code in order to be able to distinguish between supported and un-supported versions of the schema.

Hand Written Code

In hand written binding code the developers also have to make sure that the version information is maintained and used to distinguish between supported and unsupported versions of the schema. In principle this could be done by maintaining a list of supported schema versions inside the software.

Error Handling

If the binding code detects an unsupported version of an instance document it should throw an exception, but never crash. Likely the code surrounding the binding code should make sure that if a whole set of instance documents is treated that it propagates the exception(s) from the binding code, but does not stop processing the supported instance documents. Of course a warning should be raised to the upper level and finally to the user, that not all instance documents have been treated.

CVS revision numbers

Schemas and instance documents refer to CVS revision numbers, which can be inserted either by CVS keyword substitution or by accessing CVS files in the code.

CVS keyword substitution

Please refer to http://ximbiot.com/cvs/wiki/CVS--Concurrent%20Versions%20System%20v1.12.12.1%3A%20Keyword%20substitution

Access local CVS files

In Java

The following example for Accessing CVS revision numbers in Java was provided by JosephSchwarz. It finds the revision numbers by searching the file name the `Entries` file in the `CVS` subdirectory of the local directory:

```
// input parameter is the file, of which we want to retrieve the CVS revision
// number
public void setApdmFile(String cf) {
        this.ApdmFile = cf;
        File f = new File(cf);
        File entries = new File(f.getParent()+File.separator
          +"CVS"+File.separator+"Entries");  // CVS management file
        String modelName = f.getName();
        BufferedReader rdr;
        try {
            rdr = new BufferedReader(new FileReader(entries));
        } catch (FileNotFoundException e) {
            CvsVersion = "-1";
            System.out.println("CVS Entries file not found");
            return;
        }
        String line;
        try {
            // find line corresponding to our file
            while ((line = rdr.readLine()) != null) {
                if (line.contains(modelName)) {
                    String[] fields = line.split("/");
                    CvsVersion = fields[2];
                    System.out.println("APDM CVS Version is:
                      "+CvsVersion);
                    return;
                }
            }
            CvsVersion = "-1";
            System.out.println("Couldn't find filename "+modelName+"
              in CVS/Entries");
            return;
        } catch (IOException e) {
            CvsVersion = "-1";
            System.out.println("Error in readLine from CVS/Entries");
            return;
        }
```

In C++

Here a C++ example provided by MichelCaillat:

```
/*
 * $Id: SchemaVersioningProposal.txt,v 1.20 2008/01/18 14:37:57 AndreasWicenec Exp http $
 * */

 #include <iostream>
 #include <sstream>
 #include <vector>
 using namespace std;

 int main (int argC, char *argV[]) {

 vector<string> tokens;
 istringstream iss("$Id: TestCVSRevision.cpp,v 1.3 2007/12/03
    11:13:20 mcaillat Exp $");

 string s;

 while (!iss.eof()) {
 iss >> s;
 tokens.push_back(s);
 }

 cout << "Hello world, my revision number is " << tokens.at(2) << endl;
 }
```

Attachments

- APDMexample.V1.tar.gz: APDM example with full versioning

-- AndreasWicenec, AlanBridger, HolgerMeuss, JosephSchwarz, FrancoisViallefond - 2008-01-15

I	Attachment	Action	Size	Date	Who	Comment
	APDMexample.V1.tar.gz	manage	19.9 K	2008-01-18 - 14:36	AndreasWicenec	APDM example with full versioning

This topic: ZLegacy/Archive > SchemaVersioningProposal
Topic revision: r20 - 2008-01-18 - AndreasWicenec