# ADDML

*Archival Data Description Markup Language*

English edition of version 1.0:  2014 -08-08 OHS
English edition of version 1.0: updated 2014-09-02 KB
English edition of version 1.0: updated 2014-09-05 TPD
English edition of version 1.0 updated 2014-09-08 KB
English edition of version 1.0: updated 2014-11-19 OHS and TPD
English edition of version 1.0: updated 2014-11-21 TPD
Last update of original version: 2014-07-25 TPD

## *Chapter 1, Introduction*

This document describes the standard for Technical Metadata - Archival Data Description Mark-up Language (ADDML) version 8.2.

The document is divided into these main parts:

- A general introduction (Chapter 1)
- A part describing ADDML and the options embedded in the standard (Chapter 2)
- A part describing creation of profiles for use of ADDML (Chapter 3)
- And finally a part describing submitting change requests (Chapter 4)

ADDML is a standard describing a collection of data files. Such a collection is defined as a dataset, and a file containing the description of a dataset is called a dataset description.

### *Scope*

ADDML is a standard describing a collection of data files organised as flat files. A flat file in this context signifies a file existing as plain text, internally organised either by fixed positioning or delimiter separation. Such a collection of files is called a dataset. A file containing the description of a dataset is called a dataset description.

Additionally it is also possible to describe other types of files, but not in detail. This can be used to describe relations between files and metadata about them.

### *Résumé*

The standard ADDML was originally developed by the National Archives of Norway (NAN) and has existed in several different versions. Only two of the earliest versions were implemented for use by the NAN: versions 7.2 and 7.3. These versions were used in relation to the development of the tools Arkadukt 1.0 and Arkade 1.0[1]. The earliest versions of ADDML were developed simultaneously with the development of the tools, but were subsequently omitted from practical implementation.

When the development of the tools reached the test phase, the ADDML standard had already entered version 7.2. The version were implemented, but due to the discovery of deficiencies, a new version – 7.3, was developed. This version was in use after 2000.

Through new versions of the Arkadukt and Arkade software, decisions were made which resulted in a simplification of the ADDML-standard. The degree of re-working resulted in alterations, thereby changing the version number from the 7-series to version 8. The first versions in the 8-series were again of a temporary nature, until a more constant form was reached with 8.2, the present de facto standard. A new version containing minor alterations (8.3) will presumably be introduced by the end of 2014.

[1] Arkadukt and Arkade are tools developed by The National Archives of Norway (NNA), facilitating dataset descriptions according to ADDML as well as compliance testing of submitted archival materials to the Archives. The tools are only available for use in the NAN.

Along with version 8.2, a consistent approach to the development of the standard was introduced. Prior to the introduction of this version, a mutual agreement on joint development between The National Archives of Norway and Sweden was made, on the basis of both countries having applied the standard. Since then, The National Archives of Finland also has joined the cooperative framework.

The first versions of the standard were developed in order to describe flat file structures. In version 7.x, an additional option made it possible to describe xml-files. In version 8.x this has been omitted, limiting the structure to flat files since in normal cases an XML-document always is accompanied with a XML-schema (.xsd) which gives the detailed description of the structure.

## *How to use ADDML*

ADDML serves several purposes. Its main task is to describe the technical structure of a dataset designated for repository submissions. Today's standard sees an extension of its original purpose, but the technical structure remains, making it possible to describe a flat file structure when it is to be exchanged from one system to another (and not only for archival purposes).

Version 8.x additionally facilitates the description of other types of files, but not in detail, since other standards available are already handling this kind of descriptions. Emphases in the implementation of describing other files than flat files in ADDML has been put on the option of describing the file types, the relation between them and so forth.

Both the *reference* part and the *dataObjects* part are generic, making an expansion possible according to individual needs. In addition, the option of including properties has been developed and implemented from version 8.0 and on.

The implementation of ADDML requires limitations. The use of the generic parts of the standard depends on individual definitions. It is advisable to make a written outline describing your organisation's use of the standard. Read more about this in Chapter 3.

# *Chapter 2, ADDML on a General Level*

## *What ADDML 8.2 describes - Main structure*

ADDML consists of three main parts. One part describes related information (catalogue information), i.e. information related to the data of the datasets – *reference*. The second part gives detailed description of the flat files itself – *flatFiles*. The third part provides the option of relating other files than flat files to the description – *dataObjects.*

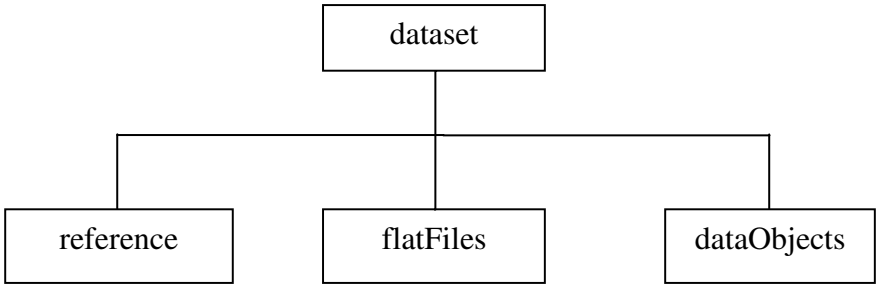


Figure 1. The main elements of ADDML.

- In *reference,* preservation metadata may be registered on a contextual and content-related level.
- In *flatFiles*, a detailed description of the data file structure may be registered, if they are given as fixed values or comma separated values.
- In *dataObjects* all files not given as fixed values or comma separated values will be described. On this level, it is nevertheless not possible to give a detailed description.
-

In an addml-file, the top levels will have the following form:

```
<addml name="addmlname">
        <dataset name="datasetname">
                <reference name="refname">
                        …..
                </reference>
                <flatFiles>
                        …..
                </flatFiles>
                <dataObjects>
                        …..
                </dataObjects>
        </dataset>
</addml>
```

Please note that *flatFiles* and *dataObjects* are not mandatory, while *reference* is.

## *Additional information – dataset information*

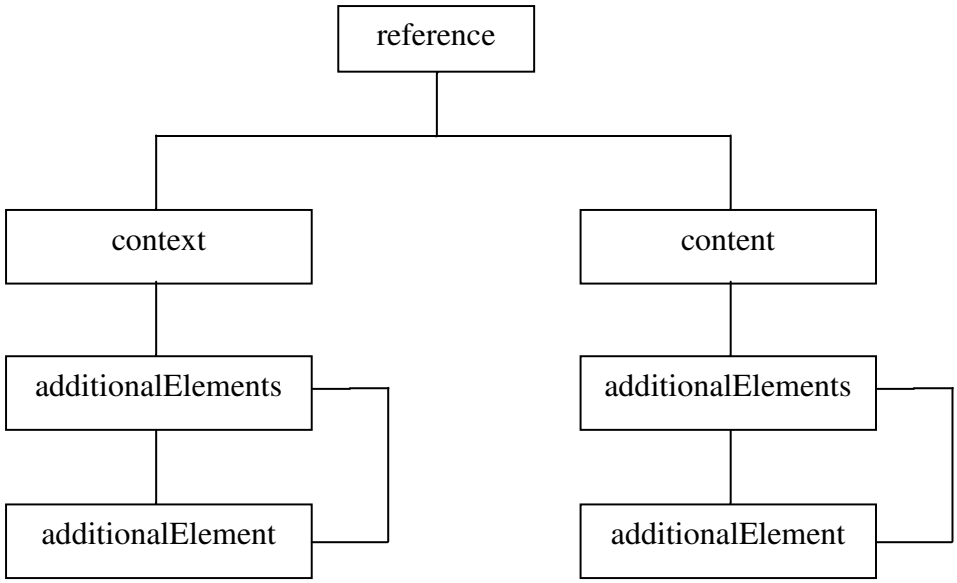


Figure 2. An overview of elements in *reference*.

Additional information is information about the dataset. This may relate to details regarding the creator of the dataset, originating system, the origins of the data in the dataset etc. This is not descriptive information of the dataset itself.

In ADDML, all additional information is gathered in the element *reference*. The element *reference* has two sub elements, *context* and *content* which describes:

- *context* - contextual information
- *content* - content-related information

The groupings have no fixed parameters, but may be defined as unique parameters for the possible inclusion in a dataset in the user profile (Chapter 3). Please refer to the section specifying generic elements to read more about defining own parameters.

As an example, *reference* including sub elements may have the following outline:

<reference>

```
<context>
    <additionalElements>
        <additionalElement name = recordCreators>
            <additionalElements>
                <additionalElement name=recordCreator>
                    <value>TheNationalArchives</value>
                </additionalElement>
            </additionalElements>
        </additionalElement>
    </additionalElements>
</context>
```

```
        <content>
            <additionalElements>
                <additionalElement name = archivalPeriod>
                    <additionalElements>
                        <additionalElement name=startDate>
                            <value>20040401</value>
                        </additionalElement>
                        <additionalElement name=endDate>
                            <value>20100331</value>
                        </additionalElement>
                        <additionalElement name=period>
                            <additionalElements >
                                <additionalElement name=retentionOutline>
                                    <value>Inactive</value>
                                </additionalElement>
                                <additionalElement name=nonCurrent>
                                    <value>Disposition</value>
                                </additionalElement>
                            </additionalElements>
                        </additionalElement>
                    </additionalElements>
                </additionalElement>
            </additionalElements>
        </content>
</reference>
```

## *The structure describing flat files*

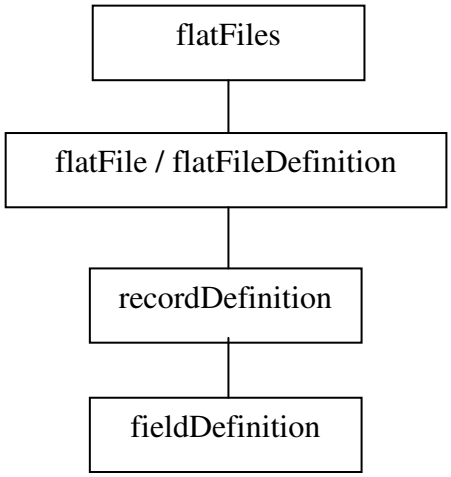


Figure 3. Overview of elements in *flatFiles* (simplified form).

The part of ADDML related to the structure, describes files in the category flat files. Flat files in this context is defined as a file having either fixed format (table format) – where all elements begins in the same position – or delimiter separated format – where the elements are separated by a specified delimiter (csv-files is an example of this format).

Figure 3 depicts a simplified version of the model. The structure has an element *flatFiles* as top level. The element *flatFiles* may contain one or more elements named *flatFile*. Equally, a flatFile may contain one or more elements *recordDefinition*. Subsequently, *recordDefinition* will contain one or more elements named *fieldDefinition*.

Under normal circumstances, a common relational database may not have any need for the record-level. This signifies a merging together of the file-level and the record-level: a table in such a database. In other circumstances, a file may nevertheless contain several record types which for some reason or another would want to split in individual tables. As a result, the connection to a database will be to that of the record-level, and not to the file-level. For this reason, all information regarding keys is placed at the record-level.

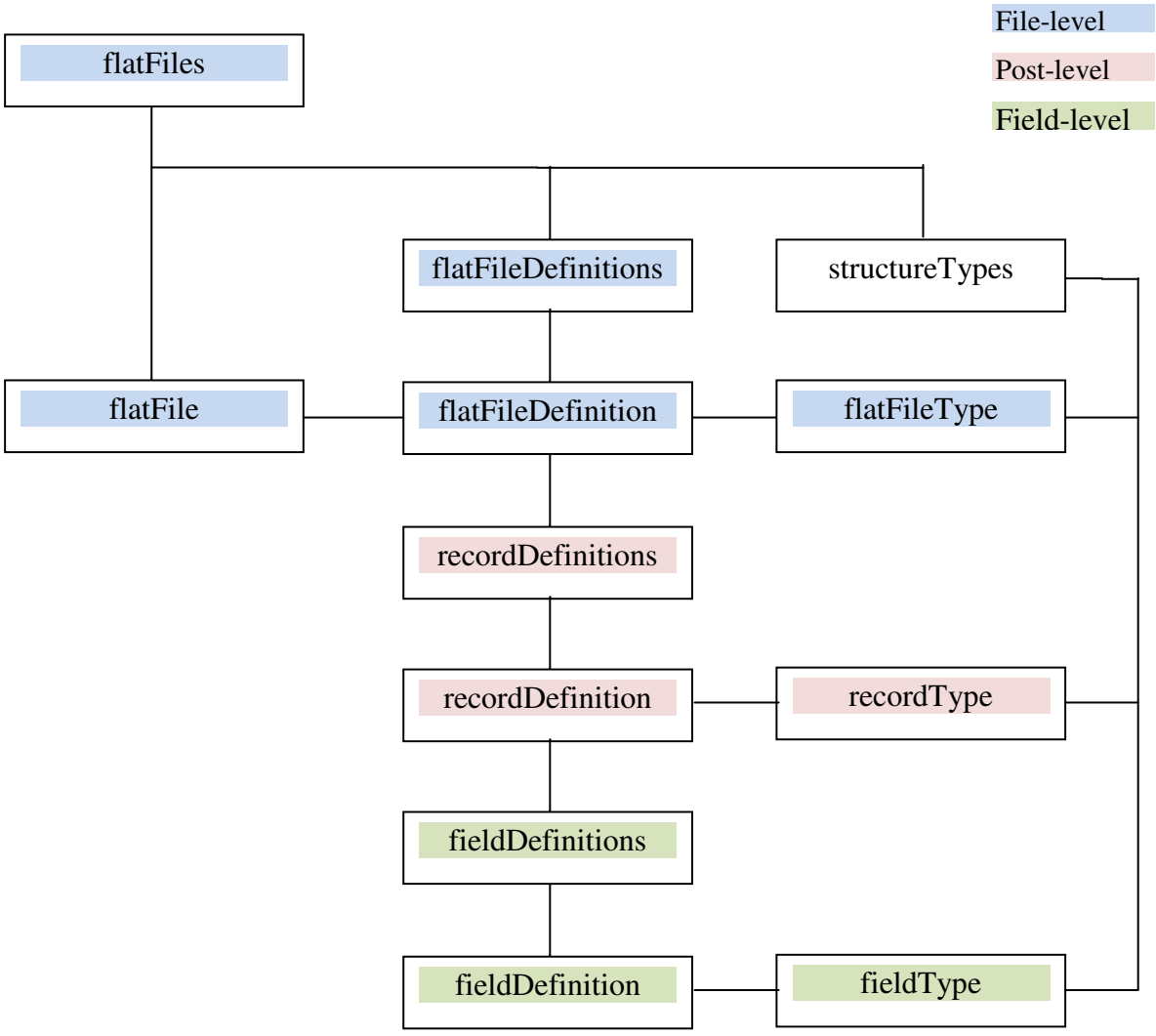


Figure 4. Overview of elements in *flatFiles* (complete form).

The complete model contains three main parts. In addition, each level contains a multiple level and a detailed level, with the exception of *flatFiles*. The three levels consistently contain information regarding the physical representation of the level, in terms of definition and in terms of overall type-definition. At present, the need for physical representation has not been deemed necessary outside the file-definition[2].

This signifies an increase in the structural complexity compared to previous versions, while at the same time increasing its flexibility and general usability.

As a result, the top levels in flatFiles may appear like this:

```
<flatFiles>
    <flatFile name="filname" definitionReference="fildef1">
    </flatFile>
    <flatFileDefinitions>
        <flatFileDefinition name="fildef1" typeReference="typefildef1">
            <recordDefinitions>
                <recordDefinition name="recorddef1" typeReference="typerecorddef1">
                    ....
                    <fieldDefinitions>
                        <fieldDefinition name="identity" typeReference="typefeltdef1">
                            ....
                        </fieldDefinition>
                    </fieldDefinitions>
                </recordDefinition>
            </recordDefinitions>
        </flatFileDefinition>
    </flatFileDefinitions>
    <structureTypes>
        <flatFileTypes>
            <flatFileType name="typefildef1">
                ....
            </flatFileType>
        </flatFileTypes>
        <recordTypes>
            <recordType name="typerecorddef1"/>
        </recordTypes>
        <fieldTypes>
            <fieldType name="typefeltdef1">
                ....
            </fieldType>
        </fieldTypes>
    </structureTypes>
</flatFiles>
```

[2] . Should the need arise, a revision of the standard may be initiated. For more about submitting change requests, see Chapter 3

Through the use of coloring, the manner in which references run between the physical level to the definition level by the use of ***fieldef1*** is shown. Equally, by the use of ***typefiledef1*** is depicted how references run between the definition level and the type level for file, and by the use of ***typerecorddef1*** for record and by ***typefeltdef1*** for field.

## *Additional files*

As previously mentioned, the structure merely contains definition of flat files with fixed or delimiter separated format. Detailed descriptions of other file types may not be performed with the use of ADDML 8.2 elements. It is nevertheless possible to relate other categories of files or information objects towards the dataset defined in an ADDML-file. Such an action is performed by defining these files and/or information objects as logical objects by using *dataObjects* and *dataObject.* To a logical object, properties may be related in order to explain the type of file and/or information related to it. The files may be data files in xml-formal, dtd or xml-schema, document files, image files, audio files, video files etc.

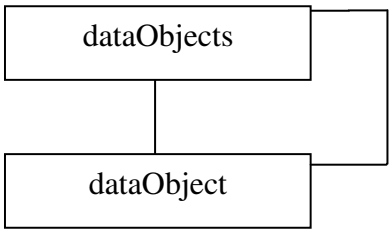


Figure 5. Overview of elements in *dataObjects*.

The following outline may be an example of the use of dataObjects:

```
<dataObjects>
    <dataObject name="Reports">
        <dataObjects>
            <dataObject name="rerecordfiles">
                ....
            </dataObject>
        </dataObjects>
    </dataObject>
</dataObjects>
```

## *Generic elements*

In the previous descriptions, numerous instances of elements forming a loop have been shown. This relates to both *additionalElements – additionalElements* and *dataObjects – dataObject*. In these instances, the structure in question is generic, and the user may construct a hierarchical structure with the previously mentioned elements.

## *Properties*

For several of the elements, there is the additional option of embedding properties in the form of the attribute *properties*. Properties also have additional elements in a generic structure through *properties – property*.

Below is given an example of the use of properties:

```
<dataObject name=”reportfile”>
    <properties>
        <property name=”filename”>
            <value>rapport.xml</value>
        </property>
        <property name=”checksum”>
            <properties>
                <property name=”algorithm”>
                    <value>SHA-256</value>
                </property>
                <property name=”value”>
                    <value>
F13CED809E4AD36198352495397FABB54DCECCBD5A33BEEDB50BBDD5C9A09232<
/value>
                </property>
            </properties>
        </property>
    </properties>
</dataObject>
```

## *Processes*

The ADDML standard provides the option of defining actions to be executed on the information. The element *processes* is used for this purpose, an element which may contain a set of the element *process*, which defines each single action. Such actions to be executed are for instance controls (where controls of codes are one example), conversions (for instance unpacking packed fields and alterations from EBCDIC to ASCII or UTF-8).

The controlling aspects of codes for the element «profession» in the record-type recorddef1 in the file filedef1 may appear like this:

```
<flatFileProcesses flatFileReference="filedef1">
  <recordProcesses definitionReference="recorddef1">
    <fieldProcesses definitionReference="profession">
      <processes>
        <process name="Control_Codes"/>
      </processes>
    </fieldProcesses>
  </recordProcesses>
</flatFileProcesses>
```

In regards to generic elements, properties and processes, it's up to the user to define individual criteria. The standard provides the options of defining these as generic elements and/or attributes.

## *A step by step description of the ADDML 8.2 elements*

Below is a description of the most important elements. They do not necessarily follow the order of the elements in the XML schema[3].

A remark regarding this list of elements is that element with names ending with the letter "s" is a grouping element always containing one or more of its sub element with the name without the letter "s".

### addml

*addml* is the top level of the structure. This element is to exist once, and once only, as defined by the rules of XML.

### dataset

*dataset* is the main level of the description. This equals a Submission Information Package. Nevertheless, one and the same description may contain numerous *dataset*. This is done in order to enable the gathering of descriptions when they are to be jointly used, for instance in a user setting.

### reference

*reference* is a joint level for the administrative information related to the dataset. This level is mandatory for every dataset.

### context and content

*context* contains contextual information about the submission. *content* contains content-related information about the submission. Both *context* and *content* contains options for individual user definitions.

### flatFiles and flatFile

*flatFiles* is the superstructure of the file structure. Each single flat file is found in *flatFile*, while *flatFiles* gather them in one unit. *flatFile* contains information regarding one individual file on the physical level. There is a reference to *flatFileDefinition*, a reference that might imply many to one.

### flatFileDefinitions and flatFileDefinition

As with the description of physical aspects, on the overall level is a splitting of the definition level into a grouping element and a defined element for each individual file. These elements are *flatFileDefinitions* and *flatFileDefinition* respectively. From *flatFileDefinition,* there is a reference to *flatFileType*.

[3] The schema can be found at http://schema.arkivverket.no/ADDML/v8.2/addml.xsd

## structureTypes

In order to reduce the degree of redundant information in the dataset description -including a simplification of the registration of it, types are introduced on the three main levels: file, record and field. These types are gathered under the element *structureTypes*. A type may be defined in a manner allowing file-, record- or fielddefinition to use identical file-, record- or fieldtype respectively. Thereby, all information on type level will be registered only once.

## flatFileTypes and flatFileType

For the types on the file level, the elements *flatFileTypes* and *flatFileType* are used as one customary grouping level and one detailed level. On a file level one may state a name of the type, a description (optional) as well as use of character set, and whether the file is in fixed or delimiter separated format.

## recordTypes and recordType

For types on the record level, the elements *recordTypes* and *recordType* are used with a grouping level and a detailed level. On a record level, a name of the type may be stated as well as a description (optional). In addition, a description of whether the fields have been trimmed may be given - that is, if the leading zeroes and subsequent blank fields are removed in the fields.

## fieldTypes and fieldType

For the types on field level, the elements *fieldTypes* and *fieldType* are used, normally with a grouping level and a detailed level. On the field level a name of the type may be stated, in addition to a description (optional), data type for the field, field format, what kind of adjustment the field contains (if packed) and whether special characters has been employed for zeroes values.

## queries and query

The idea behind these elements is to facilitate queries on the data files, for instance in the form of SQL-sentences. Such queries may document how the information has been used, and how the actual extraction has been executed. Additionally, the future option of letting other applications use these queries is thereby provided.

## processes and process

As is the case with queries, processes may be defined in an ADDML-file. Processes designate the executions one or more applications are to perform. As an option, processes may be employed in a different manner. The element processes is a grouping level, while process is a detailed level, consistent with the general approach of the ADDML-standard.

## flatFileProcesses, recordProcesses and fieldProcesses

These elements represent processes in the various parts of the structure of an ADDML-file describing flat files. The process-structure follows the same structure as flat-file-, record- and field definitions. The stating of processes relates to the designated place in the process-structure.

## recordDefinitions and recordDefinition

As with files, there is in a similar structure on the definition level for records. There is a splitting of the definition level in one grouping level and one defined element for each record. These elements are *recordDefinitions* and *recordDefinition,* respectively. From *recordDefinition* there is a reference to *recordType*.

## keys and key

As with the other elements, keys are defined with a grouping level and a detailed level. For each key will be stated the definition of key type – primary, secondary or foreign key.

## fieldDefinitions and fieldDefinition

As with files and records, a similar structure exists on the definition level of fields, with a splitting of the definition in a grouping level and a defined element for each field. These elements are *fieldDefinitions* and *fieldDefinition* respectively. From *fieldDefinition* there is a reference to *fieldType*

## fieldParts

In some instances, a splitting of an element into smaller parts may be required, while simultaneously enabling a general reference to it. The element *fieldParts* is meant to cover this requirement.

## codes and code

By the use of the grouping element *codes* and the detailed element *code*, codes which may appear in the field in the data element may be included as part of the individual field definitions. The element *code* will state the code value and a description of it.

## external and incomplete

The use of the element external is meant to create a reference to a file which is not in the dataset. In this case one will define only field-elements of interest to create a reference. Hence the definition will also be marked incomplete. Normally this kind of feature will be used towards general code tables as national postal-codes etc.

## dataObjects and dataObject

These are elements allowing the creation of a hierarchical structure for files that are not flat files. The structure has to be created in an adequate manner, dependant on an organisation's requirements.

## *Chapter 3, User profiles*

When the use of ADDML is decided a user profile is needed to be created to explain the use and also own defined elements. This document may be created using tables or in a written format. The main issue is that the document needs to exist. An accompanying document will be an version of the ADDML XML-schema with the restrictions implemented. An example of this can be found looking at the Norweigan National Archives use document and XML-schema found at: http://www.arkivverket.no/arkivverket/Arkivbevaring/Elektronisk-arkivmateriale/Standarder/ADDML

## *Chapter 4, Change request*

When the need for a change is required a change request can be submitted to the mail-address addml@arkivverket.no. The request needs to contain detailed information about what change that are needed, why and from who see the following list.

Proposal requirements:

- Statement of change
- Description of solution, if any
- Reasons/use case for the request
- Name of requestor / organization and email contact

## *Appendix 1, the full example*

This is the whole example used in the document. Also available as an XML-document at: http://www.arkivverket.no/arkivverket/Arkivbevaring/Elektronisk-arkivmateriale/Standarder/ADDML

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<addml xmlns="http://www.thenationalarchives.no/standards/addml" name ="addmlname">
  <dataset name="datasetname">
    <reference name="refname">
      <context>
        <additionalElements>
          <additionalElement name ="recordCreators">
            <additionalElements>
              <additionalElement name="recordCreator">
                <value>Thenationalarchives</value>
              </additionalElement>
            </additionalElements>
          </additionalElement>
        </additionalElements>
      </context>
      <content>
        <additionalElements>
          <additionalElement name ="archivalPeriod">
            <additionalElements>
              <additionalElement name="startDate">
                <value>20040401</value>
              </additionalElement>
              <additionalElement name="endDate">
                <value>20100331</value>
              </additionalElement>
              <additionalElement name="period">
                <additionalElements >
                  <additionalElement name="retention">
                    <value>Disposition</value>
                  </additionalElement>
                  <additionalElement name="inactivePart">
                    <value>Classified</value>
                  </additionalElement>
                </additionalElements>
              </additionalElement>
            </additionalElements>
          </additionalElement>
        </additionalElements>
      </content>
    </reference>
    <flatFiles>
      <flatFile name="filename" definitionReference="filedef1">
      </flatFile>
      <flatFileDefinitions>
        <flatFileDefinition name="filedef1" typeReference="typefiledef1">
```

```
        <recordDefinitions>
          <recordDefinition name="recorddef1" typeReference="typerecorddef1">
            <keys>
              <key name="primarykey">
                <primaryKey/>
                <fieldDefinitionReferences>
                  <fieldDefinitionReference name="identity"/>
                </fieldDefinitionReferences>
              </key>
            </keys>
            <fieldDefinitions>
              <fieldDefinition name="identity" typeReference="typefielddef1">
                <startPos>1</startPos>
                <endPos>11</endPos>
                <unique/>
              </fieldDefinition>
              <fieldDefinition name="name" typeReference="typefielddef1">
                <startPos>12</startPos>
                <endPos>41</endPos>
                <notNull/>
              </fieldDefinition>
              <fieldDefinition name="profession" typeReference="typefielddef1">
                <startPos>42</startPos>
                <endPos>61</endPos>
                <codes>
                  <code codeValue="governmentalemployee"/>
                  <code codeValue="municialemployee"/>
                  <code codeValue="privatesectoremployee"/>
                  <code codeValue="unemployed"/>
                  <code codeValue="pensioener"/>
                </codes>
              </fieldDefinition>
            </fieldDefinitions>
          </recordDefinition>
        </recordDefinitions>
      </flatFileDefinition>
    </flatFileDefinitions>
    <structureTypes>
      <flatFileTypes>
        <flatFileType name="typefiledef1">
          <charset>utf-8</charset>
          <fixedFileFormat/>
        </flatFileType>
      </flatFileTypes>
      <recordTypes>
        <recordType name="typerecorddef1"/>
      </recordTypes>
      <fieldTypes>
        <fieldType name="typefielddef1">
          <dataType>string</dataType>
```

```
            </fieldType>
          </fieldTypes>
        </structureTypes>
        <flatFileProcesses flatFileReference="filedef1">
          <recordProcesses definitionReference="recorddef1">
            <fieldProcesses definitionReference="profession">
              <processes>
                <process name="Control_Codes"/>
              </processes>
            </fieldProcesses>
          </recordProcesses>
        </flatFileProcesses>
      </flatFiles>
      <dataObjects>
        <dataObject name="Report">
          <dataObjects>
            <dataObject name="reportfile">
              <properties>
                <property name="filename">
                  <value>report.xml</value>
                </property>
                <property name="checksum">
                  <properties>
                    <property name="algorithm">
                      <value>SHA-256</value>
                    </property>
                    <property name="value">
                      <value>
F13CED809E4AD36198352495397FABB54DCECCBD5A33BEEDB50BBDD5C9A09232
                      </value>
                    </property>
                  </properties>
                </property>
              </properties>
            </dataObject>
            <dataObject name="schema">
              <properties>
                <property name="filename">
                  <value>report.xsd</value>
                </property>
                <property name="checksum">
                  <properties>
                    <property name="algorithm">
                      <value>SHA-256</value>
                    </property>
                    <property name="value">
                      <value>
F13CED809E4AD36198352495397FABB54DCECCBD5A33BEEDB50BBDD5C9A09232
                      </value>
                    </property>
```

```
                    </properties>
                </property>
            </properties>
        </dataObject>
    </dataObjects>
</dataObject>
</dataObjects>
</dataset>
</addml>
```