

## How to write an mzXML 2.0 conformant file

### Index

1. Introduction
2. Overview
3. Beginning the file
4. msRun
  - 4.1 parentFile
  - 4.2 msInstrument
    - 4.2.1 msManufacturer
    - 4.2.2 msModel
    - 4.2.3 msIonisation
    - 4.2.4 msMassAnalyzer
    - 4.2.5 msDetector
    - 4.2.6 software
    - 4.2.7 msResolution
    - 4.2.8 operator
  - 4.3 dataProcessing
    - 4.3.1 software
    - 4.3.2 processingOperation
  - 4.4 separation
  - 4.5 spotting
    - 4.5.1 plate
      - 4.5.1.1 plateManufacturer
      - 4.5.1.2 plateModel
      - 4.5.1.3 pattern
        - 4.5.1.3.1 spottingPattern
        - 4.5.1.3.2 orientation
      - 4.5.1.4 spot
        - 4.5.1.4.1 maldiMatrix
    - 4.5.2 robot
      - 4.5.2.1 robotManufacturer
      - 4.5.2.2 robotModel
  - 4.6 scan
    - 4.6.1 scansIntegration
    - 4.6.2 precursorMz
    - 4.6.3 maldi
    - 4.6.4 peaks
    - 4.6.5 nameValue
  - 4.7 sha1
5. index
  - 5.1 offset
6. indexOffset
7. sha1
8. file naming convention

### Appendix A: mzXML general types

- A.1 ontologyEntryType
- B.2 namevalueType

### Appendix B: code examples

- B.1 peaks encoding and decoding

### Appendix C: graphic conventions

## 1. Introduction

In this tutorial we show how to create an mzXML instance document that conforms to the Schema hosted at "[http://sashimi.sourceforge.net/schema\\_revision/mzXML\\_2.0/mzXML\\_idx\\_2.0.xsd](http://sashimi.sourceforge.net/schema_revision/mzXML_2.0/mzXML_idx_2.0.xsd)".

Additional documentation can be found at

"[http://sashimi.sourceforge.net/schema\\_revision/mzXML\\_2.0/Doc/mzXML\\_2.0.html](http://sashimi.sourceforge.net/schema_revision/mzXML_2.0/Doc/mzXML_2.0.html)".

It might be useful to have an example of an instance document to look at while reading this file.

You will find plenty of examples at "<http://sashimi.sourceforge.net/software.html#xmlrepository>".

The intended audience for this document is developers that are planning to write applications that write or read mzXML 2.0 conformant instance documents.

## 2. Overview

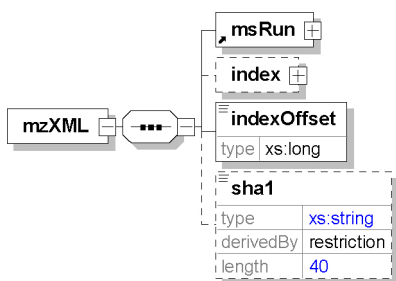


Figure 1: main components of the mzXML Schema.

The root element (*mzXML*) has four children elements:

- *msRun*: the actual MS data is stored in this element. This element has an XML Schema of its own ("[http://sashimi.sourceforge.net/schema\\_revision/mzXML\\_2.0/mzXML\\_2.0.xsd](http://sashimi.sourceforge.net/schema_revision/mzXML_2.0/mzXML_2.0.xsd)")
- *index*: the byte offsets of each scan in the instance document
- *indexOffset*: the byte offset of the index element
- *sha1*: sha1-sum calculated for the current instance document

We have decided to use two XML Schemas to separate implementation specific solutions (the indexing approach) from the actual data.

## 3. Beginning the file

The root element of our schema is called *mzXML*. Therefore the beginning of our document should look like this:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<mzXML
  xmlns="http://sashimi.sourceforge.net/schema\_revision/mzXML\_2.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://sashimi.sourceforge.net/schema\_revision/mzXML\_2.0
http://sashimi.sourceforge.net/schema\_revision/mzXML\_2.0/mzXML\_idx\_2.0.xsd">
```

## 4. msRun

This element contains the actual MS data. This element has an XML Schema of its own ("[http://sashimi.sourceforge.net/schema\\_revision/mzXML\\_2.0/mzXML\\_2.0.xsd](http://sashimi.sourceforge.net/schema_revision/mzXML_2.0/mzXML_2.0.xsd)").

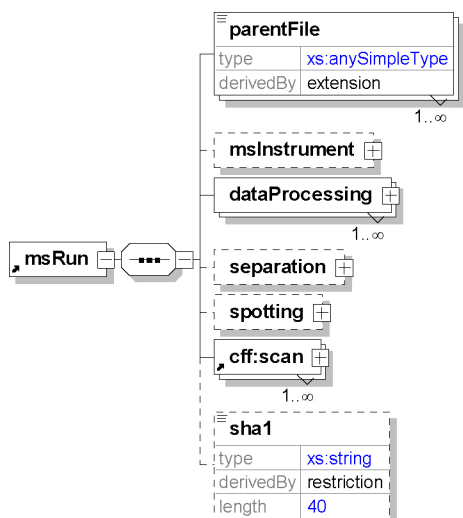


Figure 2: the *msRun* element.

The following attributes are defined for the *msRun* element:

- *scanCount* (optional): represents the total number of scans contained in the XML instance document. This number should be equal to the total number of scans in the original RAW data, unless some were not converted (e.g.: signal below threshold).
- *startTime* (optional): the time (using xs:duration type) at which the run was started.
- *endTime* (optional): the time (using xs:duration type) at which the run was finished.

This how the *msRun* element should look like in an mzXML instance document:

```
<msRun scanCount="3897"
      startTime="PT1203.57S"
      endTime="PT12001.4S">
```

#### 4.1 parentFile

This element stores a chronological list of all files used to generate a given instance document. For example, if a native output file is converted to a first mzXML document, from which a second mzXML file is created, the second mzXML document will have two parentFile elements. The first parentFile element will represent the URI (Universal Resource Identifier) of the native data, the second the URI of the first mzXML document. Each URI is also associated with a sha1-sum calculated for the corresponding file.

The following attributes are defined for the *parentFile* element:

- *fileName* (required): the URI of the file used to generate the following document or any previous mzXML document.
- *fileType* (required): this attribute has two legal values (RAWData and processedData). They describe the data content of the file referenced by the URI given in the *fileName* attribute.
- *fileSha1* (required): sha1-sum calculated for the document referenced by the URI in the *fileName* attribute. This attribute is used as a key by the *parentFileID* attribute of the *scansIntegration* element (see 4.6.1 scansIntegration).

This is how the *parentFile* element would look like for an mzXML instance document that has been created directly from a native acquisition file.

```
<parentFile   fileName="file://Regis/data2/search/jimmy2/xml-sample-data/LCQ/10mix2.RAW"
              fileType="RAWData"
              fileSha1="be3d0b00e80df8a021294dcea8e4a84e5c289f70"/>
```

If more than one native acquisition file has been used to create the current mzXML instance document another *parentFile* element can be added.

```
<parentFile fileName="17mix_test2.raw/_extern.inf"
  fileType="RAWData"
  fileSha1="c09a1cd7634251fbabb86ccb68dfb57631be1d57"/>
<parentFile fileName="17mix_test2.raw/_func001.dat"
  fileType="RAWData"
  fileSha1="34c3e2b935a0cfc633c884148f6b4c174effe151"/>
<parentFile fileName="17mix_test2.raw/_func001.sts"
  fileType="RAWData"
  fileSha1="35c93c137b6e20a5eec617c36c38c845406482d9"/>
<parentFile fileName="17mix_test2.raw/_func001.idx"
  fileType="RAWData"
  fileSha1="f3969d892fb1f0d939148c6cfd85a9c2dc52e79"/>
```

If an mzXML instance document is created from an already existing mzXML instance document the URI for the first mzXML instance document will be added to a *parentFile* element as well. Remember that files are represented in a cronological order!

```
<parentFile   fileName="file://Regis/data2/search/jimmy2/xml-sample-data/LCQ/10mix2.RAW"
  fileType="RAWData"
  fileSha1="be3d0b00e80df8a021294dcea8e4a84e5c289f70"/>
<parentFile   fileName="file://Regis/data2/search/jimmy2/xml-sample-data/LCQ/10mix2.mzXML"
  fileType="processedData"
  fileSha1="a7b3034ac48e25f223d102c8adf492696ed9065"/>
```

## 4.2 msInstrument

This element stores the specifications of the MS instrument (e.g. resolution, manufacturer, model, ionization type, mass analyzer type, detector type) and acquisition software used to generate the data. A *nameValue* element (see *namevalueType* element A.2) provides a way to store modifications to the instrument specific to a given laboratory. Even in a vendor-neutral representation it is important to preserve this information since different instruments possess different strengths and weaknesses, which ideally should be taken into consideration by the analytical software.

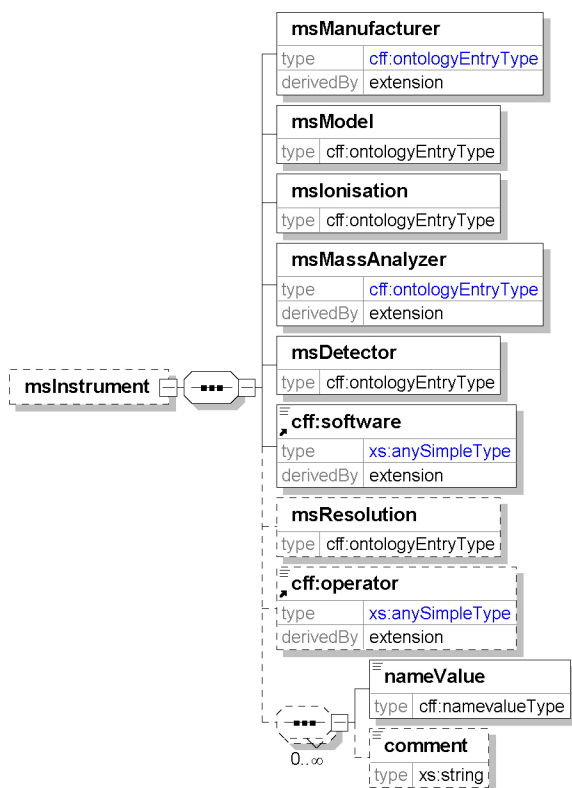


Figure 3: the *msInstrument* element

This is how the *msInstrument* element might look like an mzXML instance document:

```

<msInstrument>
  <msManufacturer category="msManufacturer" value="ThermoFinnigan"/>
  <msModel category="msModel" value="LCQ Deca"/>
  <msIonisation category="msIonisation" value="ESI"/>
  <msMassAnalyzer category="msMassAnalyzer" value="Ion Trap"/>
  <msDetector category="msDetector" value="EMT"/>
  <software type="acquisition"
    name="Xcalibur"
    version="1.3 alpha 6"/>
  <msResolution category="msResolution" value=""/>
  <operator first="Patrick"
    last="Pedrioli"/>
</msInstrument>

```

#### 4.2.1 msManufacturer

Element of type *ontologyEntryType* (see A.1 *ontologyEntryType*).

The company manufacturing the MS instrument used to generate the data in the current mzXML instance document.

Category should be of object\_term\_name *msManufacturer*.

#### 4.2.2 msModel

Element of type *ontologyEntryType* (see A.1 *ontologyEntryType*).

The model of the MS instrument used to generate the data in the current mzXML instance document.

Category should be of object\_term\_name *msModel*.

#### 4.2.3 msIonisation

Element of type *ontologyEntryType* (see A.1 *ontologyEntryType*).

The ionisation type used by MS instrument used to generate the data in the current mzXML instance document.

Category should be of object\_term\_name msIonisation.

#### 4.2.4 msMassAnalyzer

Element of type *ontologyEntryType* (see A.1 *ontologyEntryType*).

The mass analyzer part of the MS instrument used to generate the data in the current mzXML instance document.

Category should be of object\_term\_name msMassAnalyzer.

#### 4.2.5 msDetector

Element of type *ontologyEntryType* (see A.1 *ontologyEntryType*).

The detector of the MS instrument used to generate the data in the current mzXML instance document.

Category should be of object\_term\_name msDetector.

#### 4.2.6 software

Description of the acquisition software running the MS instrument.

The following attributes are defined for the *software* element:

- *type* (required): in this position the value of this attribute must be equal to "acquisition".
- *name* (required): the name of the acquisition software running the MS instrument.
- *version* (required): the version of the acquisition software running the MS instrument.
- *completionTime* (optional): time (using xs:duration type) of completion of the dataAcquisition operation. This will for most application be the same value stored in the *endTime* attribute of the *msRun* element.

This is how the software element should look like in an mzXML instance document for a dataset generated using Xcalibur.

```
<software    type="acquisition"
             name="Xcalibur"
             version="1.3 alpha 6"/>
```

#### 4.2.7 msResolution

Element of type *ontologyEntryType* (see A.1 *ontologyEntryType*).

The resolution of the MS instrument used to generate the data in the current mzXML instance document.

Category should be of object\_term\_name msResolution.

#### 4.2.8 operator

Information on the person operating the MS instrument used to acquire the data represented in the current mzXML instance document.

The following attributes are defined for the *software* element:

- *first* (required): the first name of the person operating the MS instrument
- *last* (required): the last name of the person operating the MS instrument
- *phone* (optional): the phone number of the person operating the MS instrument

- *email* (optional): the email address of the person operating the MS instrument
- *URI* (optional): a URI associated with the person operating the MS instrument

#### 4.2.9 nameValue

Element of type *namevalueType* (see A.2 *namevalueType*).

This element is used to store any additional property of the instrument not included elsewhere in the *msInstrument* element.

#### 4.3 dataProcessing

This element describes any type of data processing (e.g. conversion from native to mzXML, centroiding, noise reduction, peak finding, etc.) performed during the creation of the current instance document. The description will include name and version of the program used, a description of the input parameters and a comment field which can reference a publication illustrating the processing algorithm.

Every mzXML instance document must have at least one instance of this element describing the software used to convert the native acquisition file into the current mzXML instance document. Any additional software tool used to further process an mzXML instance document should be added in chronological order.

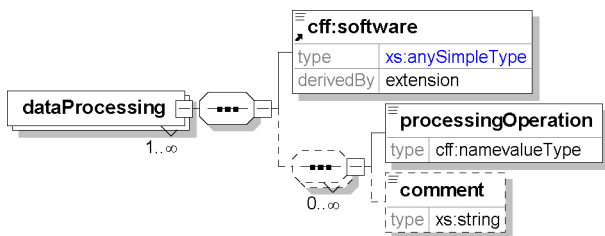


Figure 4: the *dataProcessing* element

The following attributes are defined for the *dataProcessing* element:

- *intensityCutoff* (optional): this is the minimal intensity value for an m/z - intensity pair to be included into the current mzXML instance document.
- *centroided* (optional): if equal to 1 the data in the current mzXML instance document has been centroided.
- *deisotoped* (optional): if equal to 1 the data in the current mzXML instance document has been deisotoped.
- *chargeDeconvoluted* (optional); if equal to 1 the charge states in the current mzXML instance document have been deconvoluted.
- *spotIntegration* (optional): this attribute is specific for LC-MALDI experiments and indicates if peaks eluting over multiple spots have been integrated into a single spot.

Remember that the value of these attributes apply to the whole mzXML instance document. Data processing operations that were performed only on a subset of scans should be stored in the *scan* element.

This is how the *dataProcessing* element could look like for a dataset converted and centroided using Thermo2mzXML.

```

<dataProcessing centroided="1">
  <software type="conversion"
    name="Thermo2mzXML"
    version="1"/>
</dataProcessing>
  
```

### 4.3.1 software

Description of the conversion / processing software used to process the data represented in the current mzXML instance document.

The following attributes are defined for the *software* element:

- *type* (required): in this position the value of this attribute must be either equal to "conversion" or to "processing".
- *name* (required): the name of the software tool used to process the data.
- *version* (required): the version of the software tool used to process the data.
- *completionTime* (optional): time (using xs:duration type) of completion of the data processing operation.

This is how the software element could look like in an mzXML instance document converted using Thermo2mzXML.

```
<software type="conversion"
name="Thermo2mzXML"
version="1"/>
```

### 4.3.2 processingOperation

Element of type *namevalueType* (see A.2 *namevalueType*).

This element is used to store any additional data processing operation not included elsewhere in the *dataProcessing* element.

The element can either be used to store input parameters as well as data processing operations used to create the current mzXML instance document.

## 4.4 separation

Although the mzXML format only represents information generated by mass spectrometers, some applications of mass spectrometry are so tightly coupled to a separation technique (e.g. on-line micro-capillary liquid chromatography mass spectrometry) that we felt compelled to add a separation element into the mzXML Schema. Since in a strict sense this is outside of the scope of the mzXML format, the separation element has not been developed, but is instead only a placeholder (Creating Variable Content Container Elements.

<http://www.xfront.com/BestPracticesHomepage.html>) for connecting an additional XML Schema describing a separation technique to an mzXML Schema. A simple example on how to implement such a Schema for a column separation can be found at "[http://sashimi.sourceforge.net/schema\\_revision/mzXML\\_2.0/mzXML\\_idx\\_separation\\_1.0.xsd](http://sashimi.sourceforge.net/schema_revision/mzXML_2.0/mzXML_idx_separation_1.0.xsd)".

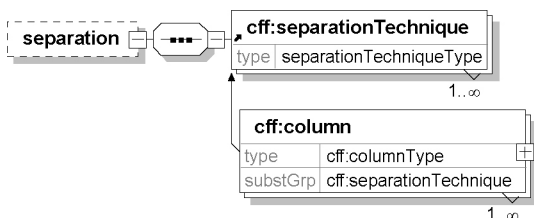


Figure 5: the *separation* element



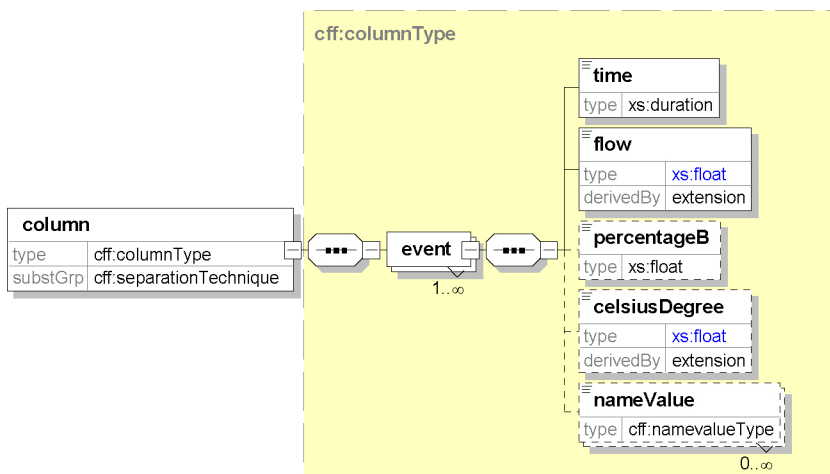


Figure 6: the *column\_separation* Schema

The mzXML Schema and the various separation Schemas can be combined in a single Schema like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://sashimi.sourceforge.net/schema_revision/mzXML_2.0"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://sashimi.sourceforge.net/schema_revision/mzXML_2.0"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:include schemaLocation="http://sashimi.sourceforge.net/schema_revision/mzXML_2.0/mzXML_idx_2.0.xsd"/>
  <xs:include
schemaLocation="http://sashimi.sourceforge.net/schema_revision/mzXML_2.0/separations/column_1.0/column_separation_1.0.xsd"/>
  <xs:annotation>
    <xs:documentation>This Schema shows how the separation element in the mzXML Schema can be used by combining the
mzXML Schema with the column_separation Schema.</xs:documentation>
  </xs:annotation>
</xs:schema>
```

This Schema includes the mzXML Schema and a Schema that describes a chromatographic column, allowing to use the separation element of the mzXML Schema. (Note that the column\_separation Schema can also be used in the absence of the mzXML Schema.)

An instance document that reference the above Schema would have the same exact structure of a standard mzXML instance document, but it can also make use of the separation element in the following way:

```
<separation>
  <column>
    <event>
      <time>PT1S</time>
      <flow unit="ul/min">1</flow>
      <percentageB>5</percentageB>
    </event>
    <event>
      <time>PT60S</time>
      <flow unit="ul/min">1</flow>
      <percentageB>15</percentageB>
    </event>
  </column>
</separation>
```

#### 4.5 spotting

This element stores those characteristics of a MALDI experiment which are constant for each

acquisition such as the matrix composition, the plate type and geometry, and the spotting robot model used, if any.

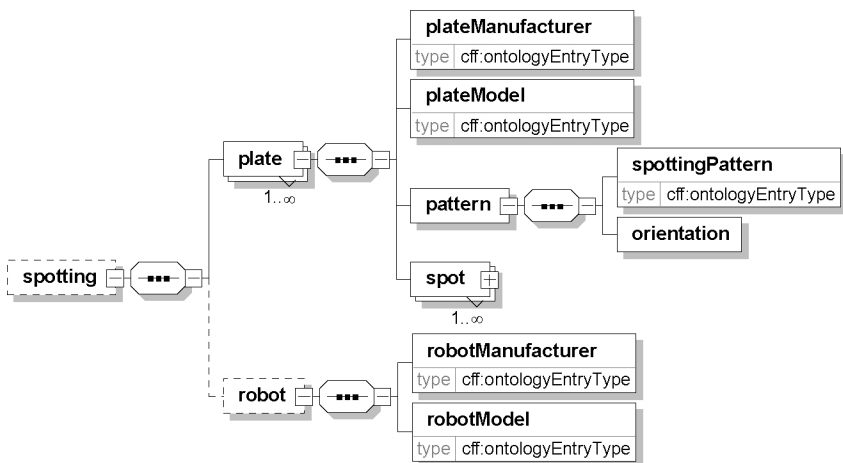


Figure 7: the *spotting* element

#### 4.5.1 plate

This element stores information about MALDI plates used to generate the current mzXML instance document.

The following attributes are defined for the *plate* element:

- *plateID* (required): this is a unique identifier associated with the plate. *plateID* is used as a key referenced by the *maldi* element under *scan* (see 4.6.3 maldi).
- *spotXCount* (required): number of spots on the x axis.
- *spotYCount* (required): number of spots on the y axis.

##### 4.5.1.1 plateManufacturer

Element of type *ontologyEntryType* (see A.1 *ontologyEntryType*).

The manufacturer of the MALDI plate used to generate the data in the current mzXML instance document.

Category should be of object\_term\_name plateManufacturer.

##### 4.5.1.2 plateModel

Element of type *ontologyEntryType* (see A.1 *ontologyEntryType*).

The model of the MALDI plate used to generate the data in the current mzXML instance document.

Category should be of object\_term\_name plateModel.

##### 4.5.1.3 pattern

This element describes the pattern used when spotting the sample on the MALDI plate used to generate the current mzXML document.

###### 4.5.1.3.1 spottingPattern

Element of type *ontologyEntryType* (see A.1 *ontologyEntryType*).

The pattern used to spot the sample on the MALDI plate used to generate the data in the current mzXML instance document.

Category should be of object\_term\_name spottingPattern.

#### 4.5.1.3.2 orientation

Each *spottingPattern* is associated with an orientation defined by the *spotID* of the first two spots to be deposited on the MALDI plate.

The following attributes are defined for the *orientation* element:

- *firstSpotID* (required): this is a unique identifier associated with the spot that was deposited first on the MALDI plate. The *firstSpotID* attribute is keyrefed to the *spotID* attribute of the *spot* element (see 4.5.1.4 spot).
- *secondSpotID* (required): this is a unique identifier associated with the spot that was deposited second on the MALDI plate. The *secondSpotID* attribute is keyrefed to the *spotID* attribute of the *spot* element (see 4.5.1.4 spot).

#### 4.5.1.4 spot

This element store information about each single spot present on the MALDI plate used to generate the current mzXML instance document.

The following attributes are defined for the *spot* element:

- *spotID* (required): this is a unique identifier associated with the spot that was deposited first on the MALDI plate. The *spotID* attribute is a key referenced by the *firstSpotID* and *secondSpotID* attributes of the *orientation* element (see 4.5.1.3.2 orientation) and by the *spotID* attribute of the *maldi* element (see 4.6.3 maldi)
- *spotXPosition* (required): x coordinate of the spot identified by the *spotID* attribute
- *spotYPosition* (required): y coordinate of the spot identified by the *spotID* attribute
- *spotDiameter* (optional): diameter (in mm) of the spot identified by the *spotID* attribute

##### 4.5.1.4.1 maldiMatrix

Element of type *ontologyEntryType* (see A.1 *ontologyEntryType*).

The matrix type used on the spot defined by *spotID*.

Category should be of object\_term\_name maldiMatrix.

#### 4.5.2 robot

Information about the robot used to spot the MALDI plate represented in the current mzXML instance document.

The following attributes are defined for the *robot* element:

- *timePerSpot* (required): this is the time (using xs:duration type) that the robot spent on each individual spot.
- *deadVolume* (optional): the dead (in ul) volume of the spotting robot.

##### 4.5.2.1 robotManufacturer

Element of type *ontologyEntryType* (see A.1 *ontologyEntryType*).

The manufacturer of the robot used to spot the plate used to generate the data stored in the current mzXML instance document. Category should be of object\_term\_name robotManufacturer.

##### 4.5.2.2 robotModel

Element of type *ontologyEntryType* (see A.1 *ontologyEntryType*).

The model of the robot used to spot the plate used to generate the data stored in the current mzXML instance document. Category should be of object\_term\_name robotModel.

#### 4.6 scan

This element has attributes to describe, amongst others, the retention time, the MS level, the polarity of the ion source, the ionization energy and the mode of acquisition (full, Selected Ion Monitoring, Selected Reaction Monitoring, etc.) for the scan being described. The scan element contains a reference of itself. This provides an intuitive way to store scans sharing a common ancestor (e.g. a common survey scan).

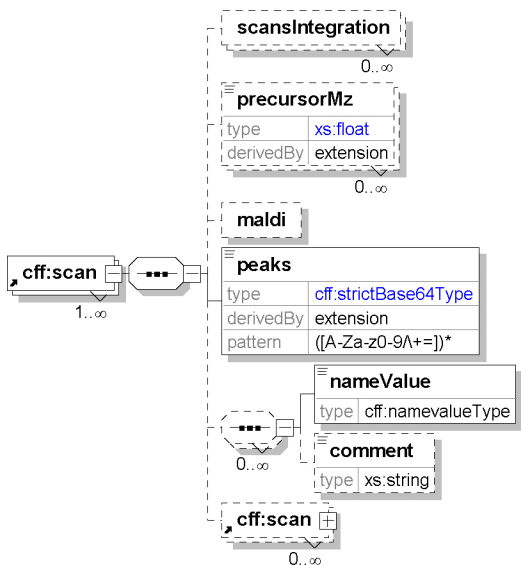


Figure 8: the *scan* element

The following attributes are defined for the *scan* element:

- *num* (required): the scan number for the current *scan* element. The values of the *num* must start from 1 and increase sequentially!
- *msLevel* (required): the MS level for the current *scan* element. 1 represents an MS scan, 2 an MS/MS scan, and so on ...
- *peaksCount* (required): the total number of m/z - intensity pairs represented in the current *scan* element.
- *polarity* (optional): the polarity of the current *scan* element. Allowed values are: "+", "-", and "any".
- *scanType* (optional): the type of the scan for the current *scan* element.
- *centroided* (optional): this can be used to specify if the current scan has been centroided. This attribute should be used when only a subset of the total scans in the current mzXML instance document have been centroided. If the all document has been centroided than the *dataProcessing* element should be used instead (see 4.3 dataProcessing).
- *deisotoped* (optional): this can be used to specify if the current scan has been deisotoped. This attribute should be used when only a subset of the total scans in the current mzXML instance document have been deisotoped. If the all document has been deisotoped than the *dataProcessing* element should be used instead (see 4.3 dataProcessing).
- *chargeDeconvoluted* (optional): this can be used to specify if the chage states in the current scan have been deconvoluted. This attribute should be used when only a subset of the total scans in the current mzXML instance document have been deconvoluted. If the all document has been deconvoluted than the *dataProcessing* element should be used instead (see 4.3 dataProcessing).
- *retentionTime* (optional): the rentention time (using xs:duration type) for the current scan.
- *ionisationEnergy* (optional): the ionisation energy for the current scan.
- *collisionEnergy* (optional): the collision energy used to fragment the precursor ion represented in the current *scan* element.
- *startMz* (optional): the lowest m/z value the MS instrument was set to read.
- *endMz* (optional): the highest m/z value the MS instrument was set to read.

- *lowMz* (optional): the lowest actual m/z value in the current scan.
- *highMz* (optional): the highest actual m/z value in the current scan.
- *basePeakMz* (optional): the m/z value for the base peak in the current scan.
- *basePeakIntensity* (optional): the intensity value for the base peak in the current scan.
- *totalIonCurrent* (optional): the total ion current in the current scan.

This is how a scan element might look like in an mzXML instance document:

```
<scan num="18"
  msLevel="2"
  peaksCount="61"
  polarity="+"
  retentionTime="PT3.636S"
  collisionEnergy="35"
  lowMz="390"
  highMz="2000"
  basePeakMz="1595.11"
  basePeakIntensity="51862"
  totalIonCurrent="345415">
  <precursorMz precursorIntensity="61852">1461.44</precursorMz>
  <peaks precision="32"
    byteOrder="network"
    pairOrder="m/z-int">Q+mbYEV+UABEOUrKRQkwAESBIAhFsAgARIJnIkWTsABeHsr4RR5w
AESGAUBFoVgARiIDukViEABEjVQkRF4AAESPLWhEy+AARI+UcES/oABEINx4RYKAAESk7gZE0KAARKcqNERPAABEq
C0aRY7wAESpjXZEKIARKr5VEVncABEq5K0RSIwAESr6PhFQCAARKww0ESDwABErYyiRWTwAESuuNBEi2AARK9gRE
Z++ABEsZf8RQiWAEsZgApFZWAARLR0tkSL4ABEtMD0Rj/kAES1MkREoWAARLV3sEV3MABEu0MIRk4UAES7WEpE8sA
ARMFT3EOegABEwvaqRnkAAETDTuhFsTgARMO89EOfgABExi6wRQmgAETHY6hHSpYARMecpkXjYABEYOPSRjxMAETJ
S1xE0wAARMI4PES0YABEYfuKReoIAETLO5RGAlAARM1BoEXH+ABEz0NgRgoEAETPWShFM+AARNQMSkY3jABE1Jo
MRYRAAETVjoBFUiAARNeXukXPEABE17RWRVogAETX0OZFgWAARNf2BEUFkABE3LgsRsAWAETeMIZFrWgARN9cyk
TIGABE34tCRN4AAETHzX5GX6AAR0hTDEWRqABE6bdYRIgAAETrUOhFiPAARPVZFkWE+AA=</peaks>
</scan>
```

The one of the children elements of the *scan* element is a *scan* element. This provides a hierarchy for storing MS scans. This hierarchy should be used to mimic the parent to children relationship present in the acquisition method used to control the MS instrument. For example if the MS instrument was set to do one MS survey scan followed by 3 MS/MS scans and one MS/MS/MS scan selected from the second MS/MS scan, the structure of the resulting mzXML instance document would be something like this:

```
<scan num="1" msLevel="1" peaksCount="190">
  <peaks precision="32" byteOrder="network" pairOrder="m/z-int">OMITTED</peaks>
  <scan num="2" msLevel="2" peaksCount="56">
    <peaks precision="32" byteOrder="network" pairOrder="m/z-int">OMITTED</peaks>
  </scan>
  <scan num="3" msLevel="2" peaksCount="34">
    <peaks precision="32" byteOrder="network" pairOrder="m/z-int">OMITTED</peaks>
    <scan num="4" msLevel="2" peaksCount="20">
      <peaks precision="32" byteOrder="network" pairOrder="m/z-int">OMITTED</peaks>
    </scan>
  </scan>
  <scan num="5" msLevel="2" peaksCount="78">
    <peaks precision="32" byteOrder="network" pairOrder="m/z-int">OMITTED</peaks>
  </scan>
</scan>
```

#### 4.6.1 scansIntegration

If the current scan has been created by merging multiple scans, this element stores the details of the integration process. There is one *scansIntegration* element for each scan that was integrated to obtain the current *scan* element. Each instance of the *scansIntegration* element identifies a

parent file and a parent scan number.

The following attributes are defined for the *scansIntegration* element:

- *parentFileID* (required): this is a unique identifier for the file containing the parent scan for the scan being described in the current *scan* element. This value is keyrefed to the *filesha1* attribute of the *parentFile* element (see 4.1 parentFile) at the beginning of the current mzXML instance document.
- *num* (required): the scan number from the *parentFileID* that was used to create the current mzXML instance document.

So if for example scan 1 and 2 from an acquisition file called 10mix2.RAW were integrated to create the current scan, we would first need to include the parent file in the *parentFile* element like this:

```
<parentFile    fileName="file://Regis/data2/search/jimmy2/xml-sample-data/LCO/10mix2.RAW"
               fileType="RAWData"
               fileSha1="be3d0b00e80df8a021294dcea8e4a84e5c289f70"/>
```

the *scansIntegration* element would then look like:

```
<scansIntegration    parentFileID="be3d0b00e80df8a021294dcea8e4a84e5c289f70"
                    num="1"/>
<scansIntegration    parentFileID="be3d0b00e80df8a021294dcea8e4a84e5c289f70"
                    num="2"/>
```

#### 4.6.2 precursorMz

This element stores the m/z, the intensity, the charge state and the wideness of the selection window for the precursor ion fragmented in the current scan. There can be multiple instances of the precursorMz element per scan element, to account for fragmentation spectra possessing more than one precursor ion (e.g. in shotgun sequencing experiments with fragments generated by in-source decay).

The following attributes are defined for the *precursorMz* element:

- *precursorScanNum* (optional): a scan number identifying a scan from which the precursor ion was selected.
- *precursorIntensity* (optional): the intensity of the precursor ion.
- *precursorCharge* (optional): the charge state of the precursor ion.
- *windowWideness* (optional): the wideness of the selection window used to select the precursor ion.

#### 4.6.3 maldi

This element stores those parts of data from a MALDI experiment that can vary between multiple scans acquired on the same spot, such as the laser intensity or the duration of the laser excitation.

The following attributes are defined for the *maldi* element:

- *plateID* (required): this is a unique identifier for the plate where the sample that is represented in the current *scan* element was spotted. The *plateID* attribute is keyrefed to the *plateID* attribute of the *plate* element (see 4.5.1 plate).
- *spotID* (required): this is a unique identifier for the spot where the sample that is represented in the current *scan* element was spotted. The *spotID* attribute is keyrefed to the *spotID* attribute of the *spot* element (see 4.5.1.4 spot).
- *laserShootCount* (optional): number of times the laser was fired to generate the current scan.
- *laserFrequency* (optional): frequency of the laser used to generate the current scan.
- *laserIntensity* (optional): intensity of the laser used to generate the current scan.

#### 4.6.4 peaks

This element contains the m/z intensity pairs as base64 encoded binary data. This element can store raw as well as processed m/z - intensity pairs.

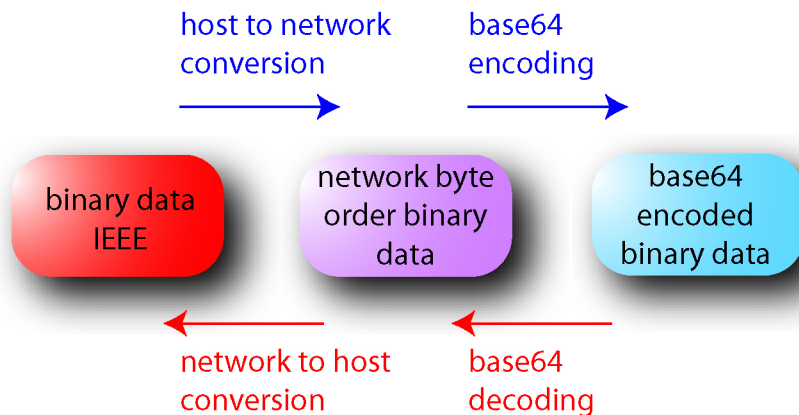


Figure 9: how m/z intensity pairs are stored in the *peaks* element. The steps to store data into a *peaks* element are represented in blue; the steps to extract data from a *peaks* element are represented in red. See also Appendix B.1 peaks encoding and decoding.

The following attributes are defined for the *peaks* element:

- *precision* (required): the precision (in bits) of the binary floating point numbers encoded in the element. This attribute can either have value 32 or 64.
- *byteOrder* (required): the byte order for the binary floating point numbers must be network.
- *pairOrder* (required): the order of the m/z - intensity pairs must be m/z-int.
- *xsi:nil* (optional): if there are no m/z – intensity pairs, but we want to have a *scan* element anyway, the *xsi:nil="1"* attribute can be set. It is important to remember to set this attribute, since an empty *peaks* will not meet the criteria of base64Binary type in the validation step.

The *peaks* element uses the following pattern `([A-Za-z0-9/\+=])*` to restrict the `xs:base64Binary` type. As a consequence the all base64 encoded binary data string must consist of one single line without spaces!

This is how a scan without any peak would look like:

```
<scan num="1" msLevel="1" peaksCount="190">  
  <peaks precision="32" byteOrder="network" pairOrder="m/z-int" xsi:nil="1"/>  
</scan>
```

#### 4.6.5 nameValue

Element of type *namevalueType* (see A.2 *namevalueType*).

This element is used to store any additional property of the scan not included elsewhere in the *scan* element.

#### 4.7 sha1

This element contains a sha1-sum (a digital signature generated by a secure hash algorithm (SHA1 version 1.0. [http://www.w3.org/PICS/DSig/SHA1\\_1\\_0.html](http://www.w3.org/PICS/DSig/SHA1_1_0.html))) that is calculated for the current instance document. This is a unique identifier that will change if a single bit of the file is

modified. It provides a way to determine if the data have been corrupted.

This element is meant to be used only when the

"[http://sashimi.sourceforge.net/schema\\_revision/mzXML\\_2.0/mzXML\\_2.0.xsd](http://sashimi.sourceforge.net/schema_revision/mzXML_2.0/mzXML_2.0.xsd)" Schema is used without the "[http://sashimi.sourceforge.net/schema\\_revision/mzXML\\_2.0/mzXML\\_idx\\_2.0.xsd](http://sashimi.sourceforge.net/schema_revision/mzXML_2.0/mzXML_idx_2.0.xsd)" Schema. When the second Schema is used, the *sha1* element presented in "7. sha1" should be used instead.

The sha1-sum is calculated from the beginning of the document to the end of the opening tag of the *sha1* element (i.e. <sha1>).

## 5. index

This element store the byte offset of a particular element in the current mzXML instance document. This offsets can then be used at parsing time to non-sequentially extract information from the mzXML instance document.

For the moment the only type of index supported by the tools available through the sashimi group indexes the position of all the scan elements.

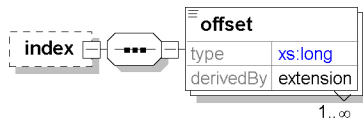


Figure 10: the *index* element

The following attributes are defined for the *scan* element:

- *name* (required): identifies the type of index by specifying the element being pointed. For the moment only "scan" is supported by the tools available from the sashimi group.

This is how a short index would look like:

```
<index name="scan">
  <offset id="1">1190</offset>
  <offset id="2">9866</offset>
  <offset id="3">12518</offset>
  <offset id="4">15810</offset>
</index>
```

### 5.1 offset

This are the actual offsets of the elements identified by the *name* attribute of the *index* element.

The following attributes are defined for the *offset* element:

- *id* (required): a unique identifier for the current instance of the *offset* element. In the case of an index of type "scan" the *id* attribute must correspond to the *num* attribute of the *scan* element.

## 6. indexOffset

This is the offset of the index element.

If the index element is not present *xsi:nil="1"* must be set.

This is how the *indexOffset* element could look like in an mzXML instance document:

```
<indexOffset>14447716</indexOffset>
```

## 7. sha1



This element contains a sha1-sum (a digital signature generated by a secure hash algorithm (SHA1 version 1.0. [http://www.w3.org/PICS/DSig/SHA1\\_1\\_0.html](http://www.w3.org/PICS/DSig/SHA1_1_0.html))) that is calculated for the current instance document. This is a unique identifier that will change if a single bit of the file is modified. It provides a way to determine if the data have been corrupted.  
To prevent using potentially wrong file offsets this element should always be used when the *index* element is used!

The sha1-sum is calculated from the beginning of the document to the end of the opening tag of the *sha1* element (i.e. <sha1>).

This is how the *sha1* element could look like in an mzXML instance document:

```
<sha1>cb8e24162e8a8c10c31e526bbe211e81c15fa8e9</sha1>
```

## 8. file naming convention

mzXML instance documents must end with a .mzXML extension.

## Appendix A mzXML general types

These types are defined in

"[http://sashimi.sourceforge.net/schema\\_revision/mzXML\\_2.0/general\\_types\\_1.0.xsd](http://sashimi.sourceforge.net/schema_revision/mzXML_2.0/general_types_1.0.xsd)". They are used in mzXML\_2.0.xsd and in column\_separation\_1.0.xsd.

### A.1 ontologyEntryType

Elements of this type have two attributes:

- *category* (required): an object\_term\_name present in the mzXML ontology ("<http://sashimi.sourceforge.net/extra/mzXMLOntology.xls>") that has object\_term\_type Class.
- *value* (required): a valid object\_term\_name for the Class defined by the *category* attribute.

### A.2 namevalueType

The idea for this element was borrowed from the MAGE language. It provides an extensible content model to the scan element (Creating Extensible Content Models.

<http://www.xfront.com/BestPracticesHomepage.html>). *nameValue* elements can be used to add entries to the instance document without having to change the Schema. This allows different laboratories to have personalized instance documents, while referring to a centralized common Schema.

Elements of this type have three attributes:

- *name* (optional): a name for the element
- *value* (optional): a value for the element
- *type* (optional): a type for the element

In the mzXML Schema each *namevalueType* element is associated with a *comment* element. The comment element can be used to further describe the content of the *namevalueType* element.

For example, the temperature of the heated capillary of an electrospray instrument could be stored in the following way:

```
<nameValue name="heatedCapillaryCelsiusTemperature"
value="203.4"
type="xs:float"/>
```

## WARNING please read carefully!!

*namevalueType* elements have the potential for creating many dialects of the mzXML Schema

and should not be used as a long time solution for storing data into mzXML instance documents. They are meant as a way to store information that will be relevant only in a limited number of experiments or as a temporary "shelter" to store information that has not yet been included in the official mzXML Schema.

So if you are using a *namevalueType* element to store some information that you think will be of general interest to the community and should be added to the official mzXML Schema, please contact the mzXML MASS committee (AGGIUNGI) and we will evaluate its inclusion into the mzXML Schema.

## Appendix B: code examples

### B.1 peaks encoding and decoding

Encoding the content of the *peaks* element in C:

```
// pData contains the m/z - intensity pairs in host format
// pData_network will contain the m/z - intensity pairs in network format

[ .... snip ....]

for( n = 0 ; n < (2 * (int)pData[0]) ; n++)
{
    (u_int32_t) ((u_int32_t *)pData_network)[n] =
        (u_int32_t) htonl( ((u_int32_t *)pData)[n+1]);
}

// pEncoded will contain the fully encoded m/z - intensity pairs
encoded_length = b64_encode( pEncoded,
                             (unsigned char *) pData_network,
                             (2 * (int)pData[0]) * sizeof(p32_t));
pEncoded[encoded_length] = '\0';

// Everything goes on the same line. No lf or spaces are allowed for this element!
fprintf( pF_xml_scan, "%s</peaks>%c", pEncoded, lf );
fprintf( pF_xml_scan, " </scan>%c", lf );
```

Decoding the content of the *peaks* element in C:

```
// This is a small snippet from the code of RAMP available
// in the cvs repository.

float *readPeaks(FILE * pFI, long lScanIndex)
{
    [ .... snip ....]

    // Base64 decoding the string out of mzXML is in pBeginData the
    // base64 decoded string will be in pDecoded
    b64_decode_mio(pDecoded, pBeginData);

    // And byte order correction pDecoded has network byte order data
    // pPeaks has the byte order of the host machine
    for (n = 0 ; n < (2 * peaksCount) ; n++)
    {
        ((u_int32_t *) pPeaks)[n] = ntohl((u_int32_t) ((u_int32_t *) pDecoded)[n]);
    }
}
```

Decoding the content of the *peaks* element in Perl:

```
// The following code is courtesy of Zack Booth Simpson and John Prince

use MIME::Base64;
```

```

$base64decoded = decode_base64($peaks_as_text);

@hostOrder32 = unpack("N*", $base64decoded);
# unpack the binary data as host ordered 32 bit ints

foreach $i (@hostOrder32) {
    $float = unpack("f", pack("I", $i));
    # The hostOrder32 array contains a list of
    # host ordered 32 bits entities which we want to re-interpret
    # as floats. In Perl this means we have to
    # pack it back as an int and then unpack it as a float
    # This would all have been simpler if only Perl
    # had had a network/host order option on unpack float
    # But we don't so alas we do the ordering operation
    # in the first unpack (N*) and then do the conversion
    # to float in the second
}

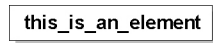
### or to clearly distinguish m/z from intensity:

$done = 0;
while (!$done) {
    $mz = unpack("f", pack("I", shift(@hostOrder32)));
    push(@mzs, $mz);
    $intensity = unpack("f", pack("I", shift(@hostOrder32)));
    push(@intensities, $intensity);
    if (@data <= 0) { $done == 1; }
}

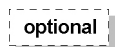
```

## Appendix C: graphic conventions

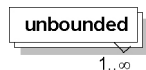
The figures representing parts of the mzXML Schema were generated using XMLSpy. The following is a very brief legend on the interpretation of the various symbols.



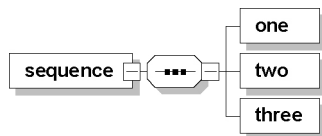
This is how an element looks like



This is how an optional element looks like



This is how an unbounded element looks like



The xs:sequence looks like this