Standard Archive Format for Europe

Primer
## Approval

<table>
<thead>
<tr>
<th>Title</th>
<th>Standard Archive Format for Europe Primer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Issue</td>
<td>Issue 1</td>
</tr>
<tr>
<td>Revision</td>
<td>Revision 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Author</th>
<th>Mathias Moucha</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>23-06-2010</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Approved by</th>
<th>Graeme Mason</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>23-06-2010</td>
</tr>
</tbody>
</table>

## Change Log

<table>
<thead>
<tr>
<th>Reason for Change</th>
<th>Issue/Issue</th>
<th>Revision/Revision</th>
<th>Date/Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creation of the Document</td>
<td>1</td>
<td>0</td>
<td>26-05-2010</td>
</tr>
</tbody>
</table>

## Change Record

Issue: 1 Revision: 1

<table>
<thead>
<tr>
<th>Reason for Change</th>
<th>Page(s)/Page(s)</th>
<th>Paragraph(s)/Paragraph(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minor Updates for ESA acceptance.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Table of Contents

1. PREFACE ............................................................................................................. 1
   1.1 Bibliography ........................................................................................................... 1
   1.2 Acronyms and Abbreviations .................................................................................. 3
   1.3 Glossary .................................................................................................................. 3
   1.4 This SAFE Primer document .................................................................................. 6

2. INTRODUCING SAFE ......................................................................................... 7
   2.1 A Standard Archive Format for Europe .................................................................. 7
   2.2 Background ............................................................................................................ 7
   2.3 A first basic example of SAFE product ................................................................. 8

3. SAFE INHERITS OAIS REFERENCE MODEL ..................................................... 14
   3.1 Introduction .......................................................................................................... 14
   3.2 The CCSDS .......................................................................................................... 14
   3.3 OAIS Main Concepts .............................................................................................. 14
      3.3.1 Different kind of data ....................................................................................... 15
      3.3.2 About Packaging ............................................................................................ 15
      3.3.3 The OAIS Architecture .................................................................................. 16
         3.3.3.1 The Ingestion ........................................................................................... 17
         3.3.3.2 The Archival Storage .............................................................................. 18
         3.3.3.3 The Dissemination ............................................................................... 18
   3.4 Information Package ............................................................................................. 18
      3.4.1 Type of Information Packages ....................................................................... 18
      3.4.2 The Archival Information Package .................................................................. 19
      3.4.3 The Submission and the Dissemination Information Packages ...................... 20

4. SAFE SPECIALIZES XFDU ................................................................................. 21
   4.1 Introduction .......................................................................................................... 21
   4.2 XFDU Package ...................................................................................................... 21
   4.3 The XFDU Package Logical Content ..................................................................... 22
      4.3.1 Data Object & Metadata Sections .................................................................. 23
      4.3.2 The Information Package Map ....................................................................... 23
   4.4 XFDU Manifest Overview .................................................................................... 23
      4.4.1 An informationPackageMap element of informationPackageMapType .......... 24
      4.4.2 A dataObjectSection element of dataObjectSectionType ............................ 25
      4.4.3 A metadataSection element of metadataSectionType .................................. 25

5. SAFE: ADVANCED CONCEPTS ......................................................................... 26
   5.1 A SAFE is an XFDU (restriction and redefinition) ................................................. 26
5.1.1 Value and Occurrence Constraints ................................................................. 26
5.1.2 Rules and Mechanisms .................................................................................. 28
5.2 A Specialization of SAFE is a SAFE ................................................................. 29
5.3 SAFE Information Model .................................................................................. 31
5.4 SAFE and XFDU Types .................................................................................... 32
5.5 SAFE Specializations Types ............................................................................. 34
5.6 SAFE Objects ..................................................................................................... 35
5.6.1 SAFE Data Objects ....................................................................................... 35
5.6.2 SAFE Metadata Objects ............................................................................... 36
5.6.2.1 SAFE Wrapped Metadata Object ............................................................... 36
5.6.2.2 SAFE Referenced Metadata Object ............................................................ 37
5.7 SAFE Control Books ........................................................................................ 39
5.7.1 Core Specifications ........................................................................................ 39
5.7.1.1 Structures .................................................................................................. 39
5.7.1.2 Manifest Type Reference ............................................................................ 39
5.7.1.3 SDF Type Reference .................................................................................. 40
5.7.1.4 Appendixes ................................................................................................ 41
5.7.2 Recommendations for Specialisations ......................................................... 41
5.7.3 Specialisations ............................................................................................... 41
5.7.3.1 Multiple SAFE Specialization Structures ................................................... 41
5.7.3.2 Manifest Type Reference ............................................................................ 42
5.7.3.3 Appendix .................................................................................................... 43
5.8 The SAFE Suite ................................................................................................ 43
5.8.1 Its Structure ................................................................................................... 43
5.8.1.1 Docs ........................................................................................................... 44
5.8.1.2 Etc .............................................................................................................. 44
5.8.1.3 Include ....................................................................................................... 44
5.8.1.4 Lib .............................................................................................................. 44
5.8.1.5 Resources .................................................................................................. 44
5.8.2 Integration of a New Specialisation ............................................................... 46

6 SAFE TOOLS .................................................................................................... 48
6.1 SAFE I/O API ................................................................................................... 48
6.2 SAFE Toolbox .................................................................................................. 49
6.2.1 Presentation ................................................................................................... 49
6.2.2 Manifest, Content Unit and Style Sheet Views .............................................. 50
6.2.2.1 Manifest View ............................................................................................ 51
6.2.2.2 Content Unit View ..................................................................................... 52
6.2.2.3 Style Sheet View ....................................................................................... 54
6.2.3 Binary Visualization and Extraction .............................................................. 55
6.2.4 Image Tool .................................................................................................... 56
6.3 SAFE Publisher ................................................................................................ 57
6.3.1 Overview ....................................................................................................... 57
6.3.2 Docbook Language ........................................................................................ 58
6.3.3 How to Use the Publisher: The Book File ................................................... 58
7

APPENDIX: XML SCHEMA OVERVIEW .............................................................61
7.1 Introduction ...........................................................................................................61
7.2 Basics ......................................................................................................................61
7.2.1 Our First Example: the Platform schema (platform.xsd) ...........................................61
7.2.2 Complex Type Definitions, Element & Attribute Declarations ................................62
7.2.3 Reference to a Global Element ............................................................................63
7.2.4 Occurrence Constraints ......................................................................................63
7.2.5 Content Models ..................................................................................................64
7.2.6 Built-In Datatypes .............................................................................................64
7.3 Advanced Concepts ...............................................................................................65
7.3.1 Target Namespaces .............................................................................................65
7.3.2 Multiple Documents for a Single Schema: the “include” element .........................66
7.3.3 Multiple Documents for a Single Schema: the “import” element ...........................68
7.3.4 Redefining types.................................................................................................69

8

APPENDIX: SDF MARK-UPS ...........................................................................70
8.1 Introduction .............................................................................................................70
8.2 Our First Example: the SDF Description of the “Platform List” .................................70
8.3 SDF Description .....................................................................................................74
8.3.1 SDF Namespace ....................................................................................................74
8.3.2 SDF Block .............................................................................................................74
8.3.3 SDF Length .........................................................................................................74
8.3.4 SDF Occurrence ...................................................................................................75
8.3.5 SDF Encoding .....................................................................................................76
8.3.6 SDF Offset ..........................................................................................................76
8.3.7 SDF Paddings .......................................................................................................77
8.3.8 SDF Array ............................................................................................................77
8.3.9 SDF Signature ......................................................................................................78
8.4 Our Second Example: the SDF Description of the “Measurement” Component ..........79
# Table of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>Example of SAFE ENVISAT AATSR NL Level-0 product – physical view-1</td>
<td>9</td>
</tr>
<tr>
<td>Figure 2</td>
<td>Example of SAFE ENVISAT AATSR NL Level-0 product – physical view-2</td>
<td>9</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Example of SAFE ENVISAT AATSR NL Level-0 product - Manifest File</td>
<td>12</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Example of SAFE ENVISAT AATSR NL Level-0 product – physical and logical views</td>
<td>13</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Obtaining information from Data</td>
<td>15</td>
</tr>
<tr>
<td>Figure 6</td>
<td>OAIS Archive External Data</td>
<td>17</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Information Package Taxonomy</td>
<td>19</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Archival Information Package (AIP)</td>
<td>20</td>
</tr>
<tr>
<td>Figure 9</td>
<td>XML Formatted Data Unit (XFDU) Package</td>
<td>21</td>
</tr>
<tr>
<td>Figure 10</td>
<td>XFDU Package Logical Content</td>
<td>22</td>
</tr>
<tr>
<td>Figure 11</td>
<td>Example of XFDU Manifest</td>
<td>24</td>
</tr>
<tr>
<td>Figure 12</td>
<td>Any SAFE is an XFDU</td>
<td>26</td>
</tr>
<tr>
<td>Figure 13</td>
<td>Content of the XFDU Metadata Section</td>
<td>26</td>
</tr>
<tr>
<td>Figure 14</td>
<td>Content of the XFDU Metadata Section – XML Schema View</td>
<td>27</td>
</tr>
<tr>
<td>Figure 15</td>
<td>Content of the SAFE Metadata Section (1)</td>
<td>27</td>
</tr>
<tr>
<td>Figure 16</td>
<td>Content of the SAFE Metadata Section (1) – XML Schema View</td>
<td>28</td>
</tr>
<tr>
<td>Figure 17</td>
<td>Content of the SAFE Metadata Section (2) – Logical View</td>
<td>28</td>
</tr>
<tr>
<td>Figure 18</td>
<td>Any SAFE Specialization is a SAFE</td>
<td>29</td>
</tr>
<tr>
<td>Figure 19</td>
<td>Example of Metadata Section for a SAFE Specialization – XML Schema View</td>
<td>30</td>
</tr>
<tr>
<td>Figure 20</td>
<td>Example of Package for a SAFE Specialization – Physical View</td>
<td>30</td>
</tr>
<tr>
<td>Figure 21</td>
<td>SAFE Information Model</td>
<td>31</td>
</tr>
<tr>
<td>Figure 22</td>
<td>Example of SAFE startTimeType</td>
<td>32</td>
</tr>
<tr>
<td>Figure 23</td>
<td>Example of SAFE instrumentType</td>
<td>33</td>
</tr>
<tr>
<td>Figure 24</td>
<td>Example of SAFE acquisitionPeriod metadataObject</td>
<td>33</td>
</tr>
<tr>
<td>Figure 25</td>
<td>Example of SAFE ra2GeneralInformation metadataObject</td>
<td>34</td>
</tr>
<tr>
<td>Figure 26</td>
<td>Example of SAFE instrumentType redefined for ASAR Level-0 Specializations</td>
<td>35</td>
</tr>
<tr>
<td>Figure 27</td>
<td>Example of SAFE InstrumentType redefined for ASAR Level-0 Specializations</td>
<td>35</td>
</tr>
<tr>
<td>Figure 28</td>
<td>SAFE Data Object</td>
<td>36</td>
</tr>
<tr>
<td>Figure 29</td>
<td>SAFE Wrapped Metadata Object</td>
<td>37</td>
</tr>
<tr>
<td>Figure 30</td>
<td>SAFE XML Schema Object</td>
<td>38</td>
</tr>
<tr>
<td>Figure 31</td>
<td>SAFE Metadata Object</td>
<td>38</td>
</tr>
<tr>
<td>Figure 32</td>
<td>Example of SAFE Type in the Manifest Type Reference</td>
<td>40</td>
</tr>
<tr>
<td>Figure 33</td>
<td>Example of XFDU Type in the Manifest Type Reference</td>
<td>42</td>
</tr>
<tr>
<td>Figure 34</td>
<td>SAFE Suite Structure</td>
<td>43</td>
</tr>
<tr>
<td>Figure 35</td>
<td>Table of Files/Location Creating a New Specialization (mandatory)</td>
<td>47</td>
</tr>
<tr>
<td>Figure 36</td>
<td>Table of Files/Location Creating a New Specialization (recommended)</td>
<td>47</td>
</tr>
<tr>
<td>Figure 37</td>
<td>SAFE API suite Architecture</td>
<td>49</td>
</tr>
<tr>
<td>Figure 38</td>
<td>fig. 1 - SAFE Toolbox Welcome Panel</td>
<td>50</td>
</tr>
<tr>
<td>Figure 39</td>
<td>SAFE Toolbox Main Window</td>
<td>51</td>
</tr>
<tr>
<td>Figure 40</td>
<td>Example of a SAFE Toolbox Manifest View</td>
<td>52</td>
</tr>
<tr>
<td>Figure 41</td>
<td>Example of a SAFE Toolbox Content Unit View</td>
<td>53</td>
</tr>
<tr>
<td>Figure 42</td>
<td>Symbols used in the Content Unit View</td>
<td>53</td>
</tr>
</tbody>
</table>
Figure 43 - Example of SAFE Toolbox Style Sheet View ..............................................................................55
Figure 44 - Separate View on Component .......................................................................................................56
Figure 45 - SAFE Toolbox Image Menu .........................................................................................................56
Figure 46 - Example of MERIS RR level 1 in SAFE Toolbox Image Tool ....................................................57
Figure 47 - Example of book.xml ....................................................................................................................59
Figure 48 - Example of Part Title ....................................................................................................................59
Figure 49 - Platform XML Schema .................................................................................................................62
Figure 50 - Nested Choice and Sequence Groups ...........................................................................................64
Figure 51 - Platform XML Schema with Target Namespace ...........................................................................66
Figure 52 - Platform XML Schema – Including instrument.xsd Schema .......................................................67
Figure 53 - Instrument XML Schema – Included by platform.xsd Schema ....................................................67
Figure 54 - Acquisition XML Schema ............................................................................................................68
Figure 55 - ENVISAT Platform XML Schema – Redefining platform.xsd ....................................................69
Figure 56 - Example of the Platform List – The Table .....................................................................................70
Figure 57 - Example of the Platform List – The Textfile ..................................................................................71
Figure 58 - Example of the Platform List – The Tree .......................................................................................72
Figure 59 - Example of the Platform List – The SDF Schema ......................................................................73
Figure 60 - Example of SDF Namespace .........................................................................................................74
Figure 61 - Example of SDF Block ..................................................................................................................74
Figure 62 - Example of SDF ASCII Integer (4 bytes long) .............................................................................75
Figure 63 - Example of SDF BINARY Unsigned Byte (2 bits long) ..............................................................75
Figure 64 - Example of SDF Occurrence .........................................................................................................76
Figure 65 - Example of SDF Encoding ............................................................................................................76
Figure 66 - Example of SDF Offset ..................................................................................................................77
Figure 67 - Example of SDF Paddings ..............................................................................................................77
Figure 68 - Example of SDF Array ..................................................................................................................78
Figure 69 - Example of SDF Signature ............................................................................................................79
Figure 70 - Example of the Measurement Component - The Table ..............................................................80
Figure 71 - Example of the Measurement Component - The SDF Schema....................................................83
1 PREFACE

1.1 Bibliography


1.2 Acronyms and Abbreviations

AIP Archival Information Package
ASCII American Standard Code for Information Interchange
CCSDS Consultative Committee for Space Data System
ESA European Space Agency
DIP Dissemination Information Package
OAIS Open Archival Information System
SAFE Standard Archive Format For Europe
SIP Submission Information Package
XFDU XML Formatting Data unit
XML eXtensible Markup Language

1.3 Glossary

Additional abstract XML Schema An additional abstract XML Schema is not the XML Schema that describes a Data Component or a Metadata Component, but is included by the XML Schema that describes a Data Component or a Metadata Component. An additional abstract XML Schema is a
Component within the SAFE Product.

Collection

Refers to Components that are gathered together along with a Manifest.

Component

Refers to a file that can be grouped together to be part of a Collection, or XFDU Package, or SAFE Product.

A Component may be a Data Component, a Metadata Component, an XML Schema Component, etc.

Consumer

The role played by those persons, or client systems, who interact with OAIS services to find preserved information of interest and to access that information in detail.

Content Unit

XML Structure that contains pointers to Data Objects and associated Metadata Objects, and possibly other Content Units.

Data

A reinterpretable representation of Information in a formalized manner suitable for communication, interpretation, or processing.

Examples of data include a sequence of bits, a table of numbers, the characters on a page, the recording of sounds made by a person speaking, or a moon rock specimen.

Data Component

A Component holding data

Data Object

Either a physical or a digital object.

dataObject

A dataObject element of a SAFE Manifest.

A datatype is a 3-tuple, consisting of:

- a set of distinct values, called its Value Space,
- a set of lexical representations, called its Lexical Space,
- a set of Facets that characterize properties of the Value Space, individual values or lexical items.

Fixity Information

The Information which documents the authentication mechanisms and provides authentication keys to ensure that the object has not been altered in an undocumented manner.

Information

Any type of knowledge that can be exchanged. In an exchange, it is represented by data. An example is a string of bits (the data) accompanied by a description of how to interpret a string of bits as numbers representing temperature observations measured in degrees Celsius (the Representation Information).
<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manifest</td>
<td>A document containing Metadata about Components, and the Associations between them. This Information is stored as a Component, using an XML language designed just for this purpose.</td>
</tr>
<tr>
<td>Manifest Type</td>
<td>Any type provided by SAFE or any SAFE Specialization contained in a SAFE Manifest.</td>
</tr>
<tr>
<td>Metadata</td>
<td>Data about other Data.</td>
</tr>
<tr>
<td>Metadata Component</td>
<td>A Component holding data about other data.</td>
</tr>
<tr>
<td>Metadata Object</td>
<td>A SAFE Object that is a metadata for another SAFE Object within the same SAFE Product.</td>
</tr>
<tr>
<td>metadataObject</td>
<td>A metadataObject element of a SAFE Manifest.</td>
</tr>
<tr>
<td>Preservation Description Information</td>
<td>Information which is necessary for adequate preservation of the Content Information and which can be categorized as Provenance, Reference, Fixity, and Context information.</td>
</tr>
<tr>
<td>Producer</td>
<td>The role played by those persons, or client systems, who provide the information to be preserved.</td>
</tr>
<tr>
<td>Representation Information</td>
<td>The information that maps a Data Object into more meaningful concepts. An example is the ASCII definition that describes how a sequence of bits (i.e. a Data Object) is mapped into a symbol.</td>
</tr>
<tr>
<td>Referenced Metadata Object</td>
<td>A Metadata Object stored inside and outside the SAFE Manifest.</td>
</tr>
<tr>
<td>SAFE Manifest</td>
<td>A Manifest conforming the SAFE Core specifications.</td>
</tr>
<tr>
<td>SAFE Object</td>
<td>Either a Data Object or a Metadata Object defined by SAFE.</td>
</tr>
<tr>
<td>SAFE Product</td>
<td>An XFDU Package specialized for Earth Observation data purposes. The term of Product has been selected for historical reason but matches exactly the definition of an XFDU Package.</td>
</tr>
<tr>
<td>SAFE Specialization</td>
<td>A SAFE Specialization is a restriction of the SAFE Core specifications for a more specific type of data. Examples of SAFE Specialization include specializations for ENVISAT or LANDSAT Products, for SPOT Measurements…</td>
</tr>
<tr>
<td>SAFE Type</td>
<td>Type defined in the “<a href="http://www.esa.int/safe/1.3%E2%80%9D">http://www.esa.int/safe/1.3”</a> namespace and part of SAFE.</td>
</tr>
</tbody>
</table>
### Wrapped Metadata

A Metadata Object wrapped inside the SAFE Manifest.

### XFDU Package

A Package that contains an XFDU Manifest and is conformant to the semantics specified in the XFDU Specifications.

### XFDU Type

Type defined in the “urn:ccsds:schema:xfdu:1” namespace and part of SAFE.

### XML Schema Component

A Component holding part or the entire Representation Information of another Component.

### XML Schema Object

A Metadata Object holding part or the entire Representation Information of another SAFE Object.

### 1.4 This SAFE Primer document

This SAFE Primer is a non-normative document intended to provide an easily readable description of the SAFE format, and is oriented towards quickly understanding how to manage a SAFE product.

SAFE Control book 1 Core Specifications (PGSI-GSEG-EOPG-FS-05-0001) and SAFE Control book 2 Recommendation for Specialisations (PGSI-GSEG-EOPG-FS-05-0002) provide the complete normative description of the SAFE format. This primer describes the format through numerous examples which are complemented by extensive references to the normative texts.
2 INTRODUCING SAFE

2.1 A Standard Archive Format for Europe

The SAFE (Standard Archive Format for Europe) has been designed to act as a common format for archiving and conveying data within ESA Earth Observation archiving facilities.

SAFE benefits from the experience gathered while developing standards related to data formats. SAFE intends to resolve the major challenges coming from the packaging and the long-term preservation of Earth Observation data. Special attention has been taken to ensure that SAFE conforms to the ISO 14721:2003 OAIS Reference Model and related standards such as the emerging CCSDS/ISO XFDU packaging format.

Although the primary goal of SAFE is to handle EO data with processing levels close to the usually called "level 0", no limitation exists regarding the packaging of higher level products as well as other technical and scientific information. Actually, experience has demonstrated that packaging and archiving higher processing levels (usually called “level 1” and “level 2”), or auxiliary data in a common format may be effective in many situations. SAFE embodies this concept by offering a single framework for packaging a large variety of information.

2.2 Background

The European Space Agency has been managing the payload data operation of a number of Earth Observation satellites since 1975. The activity includes acquisition, archiving, processing and product distribution of data from ESA and Third Party missions, for which more than 1.5 PetaBytes of data are presently archived.

The ESA EO activity is currently progressing with the operation of various ESA and Third Party satellites: ERS-2, Envisat, Landsat 5/7, NOAA 16/17, SeaWiFS, Spot 2/4, Terra, Aqua, Proba. Future missions are also planned to fly, which will be managed by the ESRIN ESA centre of Frascati, near Rome, Italy: Cryosat, GOCE, SMOS, ADM, GMES, etc., which will substantially increase the size of the long term archive.

The normal mandate for ESA's EO missions are to maintain the archive for at least ten years after the end of the mission. The management of this vast amount of heterogeneous datasets poses problems for their long term preservation, and this is why the EO Ground Segment Department has recognized for many years the need to establish a clear strategy for the management of the long term archives.

It is well known that the data to be preserved for the long term require a special attention, which is reflected in costly operations for their exploitation and maintenance. Among these challenges:
• the datasets have to be regularly converted into new media technology, to prevent the
  problems created by their obsolescence;
• since the long term archive is normally based on datasets archived usually only up to a very
  low processing level, normally called L0, higher level products have to be generated by
  processing systems;
• in addition, in the case of distribution of the data holdings directly in the archived format, it
  is normal practice that they are converted or reformatted into a format more oriented to the
  end-user utilization;
• it is a common requirement to have to extract from the long term archive a portion of the
  single data file (subsetting) to create a “child” product, optionally to be pre-processed
• finally, more and more data are distributed to end-users and exchanged among data holders
  in electronic format over network infrastructure (private Intranets, public Internet, academic
  networks, etc.)

One of the reasons that contribute to the high operations and maintenance costs of the long term
archive is the excessive proliferation of diverse and heterogeneous data formats, caused by mainly
three reasons:

• The lack of an agreed standard in the Earth Observation community, reason for which the
  formats tended to be specific for the sensor(s) each missions carried on board.
• Legacies from old ground segments architectures, which tended not to reuse elements
  previously developed.
• The non-mature status, until recently, of the information technologies and standards used to
  describe and package the data, preventing the creation of a unique format able to satisfy at
  the same time the requirements for the long-term preservation of the data and their handling
  in the processing centers.

Taking all this into account, in early 2004 ESA set up a project called HARM (Historical Archives
Rationalization and Management), which aimed mainly at converting its historical datasets into a
new modern format, based on the latest technologies and standards and able to ensure the long-
term preservation of its holdings. The format developed by the HARM project was named Standard Archive Format for Europe (SAFE).

2.3 A first basic example of SAFE product

Let us start with an example of a SAFE Product deliberately simplified: “EN01_ATS_NL__0P_20041216T233337_20041216T235845_GAE_14601_DB9F.SAFE”.

This product is a directory and contains several files: an XML document named “manifest.safe”, a binary file named “measurement.dat” and another XML file, in fact an XML Schema named “measurement.xsd”.

“measurement.dat” is a heavy file, a series of bits – zeroes and ones. On its own – i.e. taken out of context, it’s merely meaningless data. Before we can interpret this sequence of bits, we must have the “key”, a document with the complete structure and description of the data contained in this binary file. It is what the XML Schema “measurement.xsd” does.
An XML Schema (see Appendix) allows us to describe the contents of any XML document in the form of a tree. An important point about SAFE is that an XML Schema may also describe documents other than XML ones, including ASCII and binary.

Apart from the two files (“measurement.dat” and “measurement.xsd”) – the raw data and its description”, this SAFE Product contains an XML document, the manifest. Note that, even though this product is just an example, a SAFE Product always contains an XML document which is always named “manifest.safe” (plus variable number of files called “components”).

N.B.: a SAFE product contains a manifest named “manifest.safe” (or “MANIFEST.SAFE”). The uppercase notation is to be used only if the supporting file system does not support lower case notation.

The manifest contains general product-related information (e.g. its history, which platform the data come from, the acquisition period, etc.).

The manifest also contains a structured representation of the product. For example, the manifest contains the link between the two files “measurement.dat” and “measurement.xsd” (this link means “the first is described by the second”).

N.B.: the manifest of a SAFE product contains the information which gives the product its meaning. Without the manifest, the product becomes “anonymous”.

Let us open the manifest and examine its contents:

```xml
<xfdu:XFDU xmlns:xfdud="urn:ccsds:schema:xfdud:1"
           xmlns:safe="http://www.esa.int/safe/1.3"
           version="esa/safe/1.3/envisat/aatsr/nl/level-0">
  <!-- INFORMATION PACKAGE MAP SECTION -------------------------------------- -->
  <informationPackageMap>
    <xfdu:contentUnit ID="packageUnit" pdi="processing" dmdID="platform"/>
    <xfdu:contentUnit ID="measurementUnit" repID="measurementSchema"/>
  </informationPackageMap>
  <!-- METADATA OBJECT SECTION --------------------------------------------- -->
  <metadataSection>
    <metadataObject ID="processing" classification="PROVENANCE" category="PDI">
      <metadataWrap vocabularyName="SAFE" mimeType="text/xml">
        <xmlData>
          <safe:processing name="Conversion from ENVISAT to SAFE format"
                           start="2009-10-10T15:28:43.542000Z"/>
          <safe:facility country="France" name="GAEL Consultant"
                         organisation="GAEL Consultant"/>
        </xmlData>
      </metadataWrap>
    </metadataObject>
  </metadataSection>
</xfdu:XFDU>
```
<xmlData>
  <safe:platform>
    <!-- Platform identification -->
    <safe:nssdcIdentifier>2002-009A</safe:nssdcIdentifier>
    <safe:familyName>ENVISAT</safe:familyName>
    <safe:number>1</safe:number>
    <!-- Instrument identification -->
    <safe:instrument>
      <safe:familyName abbreviation="AATSR">Advanced Along Track Scanning Radiometer</safe:familyName>
    </safe:instrument>
    <safe:timeReference>
      <safe:utc>2004-12-16T22:19:00.483654Z</safe:utc>
      <safe:clock>1133104128</safe:clock>
      <safe:clockStep>3906249804</safe:clockStep>
    </safe:timeReference>
  </safe:platform>
</xmlData>
Figure 3 - Example of SAFE ENVISAT AATSR NL Level-0 product - Manifest File

As mentioned above, the manifest is an XML document. This document contains a “root element” called “XFDU” which is declared in the namespace “urn:ccsds:schema:xfdu:1”. This element is a sequence of three sub-elements: “informationPackageMap”, “metadataSection” and “dataObjectSection”.

Let us look first at “dataObjectSection”. The “dataObjectSection” contains a single “dataObject” element whose ID is “measurementData”. Its “fileLocation” sub-element has an attribute “href” whose value is “measurement.dat” (this is the name of the binary file next to the manifest within this product). Via this link, the “dataObject” references the “measurement.dat” component from the manifest.

N.B.: the components can be referenced from the manifest using the “dataObject” elements.

Now let us turn to the “metadataSection”. The “metadataSection” contains three “metadataObject”s whose IDs are “processing”, “platform” and “measurementSchema”. Let us begin with the last one, the “measurementSchema”. Its “metadataReference” sub-element has an attribute “href” whose value is “measurement.xsd” (this is the name of the XML Schema next to the manifest within this product). Via this link, the “metadataObject” references the “measurement.xsd” XML Schema from the manifest.

N.B.: the components can also be referenced from the manifest using the “metadataObject” elements.

In fact, as we will see a little further on, components are always referenced from the manifest via a “dataObject” element, unless the component is an XML Schema. The component is only referenced from the manifest via a “metadataObject” element in this case.

The two other “metadataObject”s, whose IDs are “processing” and “platform”, are very similar. Both contain, wrapped within an “xmlData” sub-element, a sequence of elements declared in the “http://www.esa.int/safe/1.3” namespace. These elements are information about the product history (“processing”) and the platform that acquired the product’s data (“platform”).

repID="measurementSchema">
<getBytes mimeType="application/octetstream">
  <fileLocation locatorType="URL" href="measurement.dat"/>
  <checksum checksumName="MD5">ae02c09bf4a00ab5b813f6bb724ae24a</checksum>
</getBytes>
</dataObject>
</dataObjectSection>
</xfdu:XFDU>
Finally, let us look at the “informationPackageMap”. This section contains information about the package (i.e. the SAFE Product), which is structured and which, rather like a map, gives us a general view of the product.

“informationPackageMap” contains a “contentUnit” sub-element (which is declared in the “urn:ccsds:schema:xfdu:1” namespace); this “contentUnit” also contains a “contentUnit” sub-element. The identifier of the top “contentUnit” is “packageUnit” and for the bottom one “measurementUnit”.

This wrapping means that “the main entity of the SAFE product is the product (or package) itself, and this entity contains a sub-entity which is the Measurement”.

The “informationPackageMap” allows us to have a logical view of the SAFE product:

![Diagram](image)

*Figure 4 - Example of SAFE ENVISAT AATSR NL Level-0 product – physical and logical views*
3 SAFE INHERITS OAIS REFERENCE MODEL

3.1 Introduction

In the first chapter, we have seen that during the development of SAFE, a special attention has been taken to ensure that SAFE conforms to the ISO 14721:2003 OAIS Reference Model.

An “OAIS” (Open Archival Information System) is an archive, consisting of an organization of people and systems that has accepted the responsibility to preserve information and make it available for a Designated Community. It meets a set of such responsibilities as defined in the OAIS Reference Model, and this allows an OAIS to be distinguished from other uses of the term 'archive'. The term 'Open' in OAIS is used to imply that the OAIS Reference Model Recommendation, as well as future related Recommendations and standards, are developed in open forums, and it does not imply that access to the archive is unrestricted. The reference model addresses a full range of archival information preservation functions including ingest, archival storage, data management, access, and dissemination. The OAIS Reference Model is a technical recommendation of the CCSDS.

3.2 The CCSDS

Founded in 1982 by the major space agencies in the world, the Consultative Committee for Space Data Systems (CCSDS) originated as a multi-national forum for the discussion of common space communications issues. It is currently composed of ten member agencies, twenty-two observer agencies, and over 100 industrial associates. The Committee meets periodically to address data systems problems that are common to all participants, and to formulate sound technical solutions to these problems. The goal of the Committee is to enhance governmental and commercial interoperability and cross-support, while also reducing risk, development time and project costs. More than 300 space missions have chosen to fly with CCSDS-developed standards and the number continues to grow.

3.3 OAIS Main Concepts

To fully understand the concepts of the OAIS Reference Model, it is important to define some key words.
3.3.1 DIFFERENT KIND OF DATA

In everyday language, “information” or “data” are synonymous. When we speak about data we think about a sequence of words and/or digits whose precise order confers meaning. OAIS reference Model precisely describes the different types of information.

A **Data** is a reinterpretable representation of information in a formalized manner suitable for communication, interpretation, or processing. Examples of data include a sequence of bits, a table of numbers, the characters on a page, the recording of sounds made by a person speaking, or a moon rock specimen.

An **Information** is any type of knowledge that can be exchanged. In an exchange, it is represented by data. An example is a string of bits (the data) accompanied by a description of how to interpret a string of bits as numbers representing temperature observations measured in degrees Celsius (the representation information).

A **Metadata** is **Data** about other **Data**. An example is the name of the singer (or the band) printed on a CD.

3.3.2 ABOUT PACKAGING

A package is a very important concept for an OAIS. Let us examine its content, which is strictly defined by the OAIS reference Model.

![Figure 5 - Obtaining information from Data](image)

**A Data Object** is either a Physical Object or a Digital Object.

The **Representation Information** is the information that maps a **Data Object** into more meaningful concepts. An example is the ASCII definition that describes how a sequence of bits (i.e., a **Data Object**) is mapped into a symbol.

**An Information Object** is a **Data Object** together with its **Representation Information**.
We can describe these concepts in greater detail:

The **Content Data Object** is the *Data Object*, that together with its associated *Representation Information*, is the original target of preservation. It is the data that has originated the creation of the archive.

The **Content Information** is the set of information that is the original target of preservation. It is an *Information Object* comprised of its *Content Data Object* and its *Representation Information*.

The **Preservation Description Information (PDI)** is the information which is necessary for adequate preservation of the *Content Information* and which can be categorized as Provenance (*Provenance Information*), Reference, Fixity (*Fixity Information*), and Context information.

The **Provenance Information** is the information that documents the history of the *Content Information*. This information tells the origin or source of the *Content Information*, any changes that may have taken place since it was originated, and who has had custody of it since it was originated.

The **Fixity Information** is the information which documents the authentication mechanisms and provides authentication keys to ensure that the *Content Information* object has not been altered in an undocumented manner.

The **Information Package** is the *Content Information* and associated *PDI* which is needed to aid in the preservation of the *Content Information*. The *Information Package* has associated Packaging Information used to delimit and identify the *Content Information* and *PDI*.

### 3.3.3 THE OAIS ARCHITECTURE

Data are archived in three successive and mandatory phases: ingestion, archival storage and dissemination.
3.3.3.1 The Ingestion

Ingestion is the phase during which a producer submits data to an OAIS for archiving.

The persons, or client systems, who provide the information to be preserved are called the **Producer**.

The **Ingest** is the OAIS entity that contains the services and functions that accept Submission Information Packages from Producers, prepares Archival Information Packages for storage, and ensures that Archival Information Packages and their supporting Descriptive Information become established within the OAIS.

A **Submission Information Package (SIP)** is an Information Package that is delivered by the Producer to the OAIS for use in the construction of one or more Archival Information Packages.
3.3.3.2 The Archival Storage

Archival storage is the phase during which an OAIS, having received the data submitted by a producer, stores it in a precise location and keeps a record of this action.

An Archival Information Package (AIP) is an Information Package, consisting of the Content Information and the associated PDI, which is preserved within an OAIS.

The Data Management is the OAIS entity that contains the services and functions for populating, maintaining, and accessing a wide variety of information. Some examples of this information are catalogs and inventories on what may be retrieved from Archival Storage.

3.3.3.3 The Dissemination

Dissemination is the phase during which a Consumer requests an archive to be extracted, in return obtains the corresponding data.

The persons, or client systems, who interact with OAIS services to find preserved information of interest and to access that information in detail are called the Consumer.

The Access entity provides the services and functions that support Consumers in determining the existence, description, location and availability of information stored in the OAIS, and allowing Consumers to request and receive information products.

A Dissemination Information Package (DIP) is the Information Package, derived from one or more Archival Information Packages, received by the Consumer in response to a request to the OAIS.

3.4 Information Package

The conceptual structure for supporting Long Term Preservation of information is the Information Package. An Information Package is a container that contains two types of Information Objects, the Content Information and the PDI.

3.4.1 TYPE OF INFORMATION PACKAGES

There are three types of Information Package:

- The Submission Information Package (SIP);
• The Archival Information Package (AIP);
• The Dissemination Information Package (DIP).

The **SIP** is that package that is sent to an OAIS by a **Producer**. Its form and detailed content is typically negotiated between the **Producer** and the OAIS. Within the OAIS, one or more **SIPs** are transformed into one or more **AIP(s)** for preservation. In response to an Order, the OAIS provides all or a part of an **AIP** to a **Consumer** in the form of a **DIP**.

The exact information contents of the **SIP** and **DIP** and their relationship to the corresponding **AIP** are dependent on the agreements between the archive and its **Producers** and **Consumers**.

### 3.4.2 THE ARCHIVAL INFORMATION PACKAGE

An **AIP** is a specialization of the **Information Package**. The **AIP** is defined to provide a concise way of referring to a set of **Information** that has, in principle, all the qualities needed for permanent, or indefinite, Long Term Preservation of a designated **Information Object**.
Also within the AIP is an Information Object called the PDI. The PDI contains additional information about the Content Information and is needed to make the Content Information meaningful for the indefinite long-term.

The AIP is delimited and identified by the Packaging Information. The Packaging Information may actually be present as a structure on the media that contains the AIP or, it may be virtual in that it is contained in the OAIS Archival Storage function. However, the delimitation and internal identification functions must be well defined in an OAIS.

Each AIP is associated with a structured form of Descriptive Information called the Package Description, which enables the Consumer to locate information of potential interest, analyze that information, and order desired information.

3.4.3 THE SUBMISSION AND THE DISSEMINATION INFORMATION PACKAGES

The SIP is the package that is sent to an OAIS by a producer, while the DIP is the package that is sent to a Consumer by an OAIS.

The form and detailed content of a SIP (or a DIP) are typically negotiated between the Producer (or the Consumer, for a DIP) and the OAIS.
4 SAFE SPECIALIZES XFDU

4.1 Introduction

The main purpose of XFDU Recommendations is to define a CCSDS Recommended Standard for the packaging of data and metadata, including software, into a single package (e.g. file or message) to facilitate information transfer and archiving. XFDU Recommendations are fully compliant with the OAIS Reference Model.

4.2 XFDU Package

A Package Interchange File is a collection of files that have been bundled together into a single container that also contains a Manifest describing the contained files and the relationships among those files.

Figure 9 - XML Formatted Data Unit (XFDU) Package
An **XFDU Package** is a *Package Interchange File* that contains an **XFDU Manifest** and is conformant to the semantics specified in the XFDU Specifications. An **XFDU Package** is a specialization of *Package Interchange File*.

A **Manifest** is a document containing metadata about **Components**, and the relationships among them. This information is stored as a **Component**, using an XML language designed for just this purpose.

An **XFDU Manifest** is a **Manifest** that is conformant to the XML Schema specified in XFDU Specifications.

A **Component** is an object (i.e. file) that can be grouped together to be part of a **Collection**, or Package (a group of **Components** that are gathered together is called a **Collection**).

### 4.3 The XFDU Package Logical Content

![Figure 10 - XFDU Package Logical Content](image-url)
4.3.1 DATA OBJECT & METADATA SECTIONS

The **Data Object Section** is an optional section that contains all Content Data Objects (i.e. Data Objects that are the original target of preservation). Physically, each (Content) Data Object is composed of a `dataObject` element in the XFDU Manifest plus one Component (each `dataObject` element references a single Component).

The **Metadata Section** is an optional section that contains or references all of the static metadata for all items in the XFDU Package. Is considered “metadata” all data which is not the original target of preservation. A Metadata Objet can be physically wrapped by the XFDU Manifest, but it may also be composed of one Component plus one or more elements within the XFDU Manifest.

4.3.2 THE INFORMATION PACKAGE MAP

One or more **Information Package Maps** provide hierarchical views of the content of the XFDU Package, using a series of nested Content Units. An Information Package Map is equivalent to a highest-level Content Unit included in the XFDU Package.

A **Content Unit** is an XML Structure (physically a `contentUnit` element) that contains pointers to Data Objects and associated Metadata Objects; it may also contain other Content Units.

4.4 XFDU Manifest Overview

Here is an example of an XFDU Manifest:

```
<xfdu:XFDU xmlns:xfdu="urn:ccsds:schema:xfdu:1"
    version="esa/safe/1.3/envsat/gomos/nl/level-0">

<!-- INFORMATION PACKAGE MAP SECTION
---------------------------------- -->
<informationPackageMap>
    <xfdu:contentUnit ID="packageUnit" pdi="processing" dmdID="platform">
        <xfdu:contentUnit ID="measurementUnit" repID="measurementSchema">
            <dataObjectPointer dataObjectID="measurementData"/>
        </xfdu:contentUnit>
    </xfdu:contentUnit>
</informationPackageMap>

<!-- METADATA OBJECT SECTION
---------------------------------- -->
<metadataSection>
```
Figure 11 - Example of XFDU Manifest

XFDU is the root element of an XFDU Manifest. This XFDU element contains a sequence of three elements.

4.4.1 AN INFORMATIONPACKAGEMAP ELEMENT OF INFORMATIONPACKAGEMAPTYPE

The informationPackageMap element describes hierarchic structure of the product. Nested contentUnit and dataObjectPointer elements (by their pdiID, dmdID, repID and dataObjectID attributes) reference metadataObject and dataObject elements. Each Content Unit represents one entity part of the XFDU Package. The main Content Unit (the first contentUnit sub-element of the Information Package Map) represents the XFDU Package itself.
4.4.2 A DATAOBJECTSECTION ELEMENT OF DATAOBJECTSECTIONTYPE

The `dataObjectSection` element contains a sequence of any number of `dataObject` elements. A `dataObject` element of `dataObjectType` relates a `Component` (except an XML Schema `Component`) to the XFDU Manifest via its `byteStream/fileLocation/@href` attribute.

4.4.3 A METADATASECTION ELEMENT OF METADATASECTIONTYPE

The `metadataSection` element records all of the metadata for all items in the XFDU Package. Multiple `metadataObjects` of `metadataObjectType` are allowed within the `metadataSection` so that the metadata can be recorded for each separate item within the XFDU Package. A Metadata Object (a `metadataObject` element of `metadataObjectType`) is used to either encapsulate metadata in XML, to point to an XML Schema `Component` within the XFDU Package, or to point to a `dataObject` element in the `dataObjectSection` element.
5 SAFE: ADVANCED CONCEPTS

5.1 A SAFE is an XFDU (restriction and redefinition)

SAFE is based on the XFDU standard. In its essence, SAFE is a profile of XFDU, and it restricts the XFDU specifications for the specific utilization in the EO domain.

During the redefinition (or restriction), some features of XFDU are discarded or constrained according to the specific needs of SAFE.

5.1.1 VALUE AND OCCURRENCE CONSTRAINTS

SAFE may constrain values of particular attributes, occurrences of elements.

For example, XFDU Specification defines that the Metadata Section (metadataSection element) must contain any number of Metadata Objects (metadataObject elements) – even zero:

![Diagram of Metadata Section and Metadata Objects](image)
SAFE Primer
issue 1 revision 1 - 23-06-2010
PGSI-GSEG-EOPG-FS-010-0001
page 27 of 83

Here is its equivalent written in XML Schema:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xfdu="urn:ccsds:schema:xfdu:1"
  targetNamespace="urn:ccsds:schema:xfdu:1"
  elementFormDefault="unqualified"
  attributeFormDefault="unqualified">
  (...)
  <xs:complexType name="metadataSectionType">
    <xs:complexContent>
      <xs:restriction base="xfdu:metadataSectionType">
        (...)
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
  (...)
</xs:schema>
```

**Figure 14 - Content of the XFDU Metadata Section – XML Schema View**

SAFE Specification restricts XFDU Specification and defines that the *Metadata Section* (metadataSection element) must contain a number of *Metadata Objects* (metadataObject elements) greater or equal to two:

![Diagram](image)

**Figure 15 - Content of the SAFE Metadata Section (1)**

Here is its equivalent written in XML Schema:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xfdu="urn:ccsds:schema:xfdu:1"
  targetNamespace="urn:ccsds:schema:xfdu:1"
  elementFormDefault="unqualified"
  attributeFormDefault="unqualified">
  <xs:redefine schemaLocation="xfdu.xsd">
    (...)
  </xs:redefine>
  <xs:complexType name="metadataSectionType">
    <xs:complexContent>
      <xs:restriction base="xfdu:metadataSectionType">
        (...)
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:schema>
```

ESA UNCLASSIFIED
For Internal Use
SAFE Specification may also restrict values of the Metadata Objects (metadataObject elements) – two of the Metadata Objects must be the “processing” and the “platform” Metadata Objects:

```
<x:schema>
  <x:element name="metadataObject"
      type="xfdu:metadataObjectType"
      minOccurs="2" maxOccurs="unbounded"/>
</x:sequence>
</x:restriction>
</x:complexType>
(...)
</x:redefine>
</x:schema>
```

**Figure 16 - Content of the SAFE Metadata Section (1) – XML Schema View**

5.1.2 RULES AND MECHANISMS

SAFE adds rules of consistence, mechanisms of connection between the various components of a SAFE Product (Manifest file, XML schemas, binary files etc.)

Let us take an example. In order to point to an XML Schema from the XFDU Manifest, XFDU Recommendation offers an attribute called “@repID”. This attribute’s type, “xs:IDREFS”, is a sequence of identifiers of the “xs:ID” type. Each one of these must be separated by one or more blanks.

**Figure 17 - Content of the SAFE Metadata Section (2) – Logical View**
Within an XFDU Package, a component can be described by one or more XML Schemas. The order of these schemas within the list is not imposed, because no rule has been set for the case of multiple schemas. However, even if a schema is made-up of several files (see the section 7.3.2), only one of these (the one which includes all the others), in fact describes the contents of this component.

And XFDU Recommendation gives us no means of unambiguous identification.

SAFE Recommendation proposes a simple rule to resolve any possible ambiguity:

⇒ Within the list of XML Schemas, the main schema will always be in first place.

### 5.2 A Specialization of SAFE is a SAFE

We have seen that SAFE is a specialization (a restriction) of XFDU. In the same way, a SAFE Specialization is a specialization (a restriction) of SAFE:

![Diagram](Figure 18 - Any SAFE Specialization is a SAFE)

The redefinition or restriction mechanisms are identical to those seen previously (constraints on values and occurrences, and the use of specific SAFE rules).

Some constraints can also be defined concerning the contents of a SAFE Manifest. For example, the `metadataSection` must contain exactly ten `metadataObject`es.
Some constraints can also be defined concerning the number and, of course, the nature of the Components. For example, the package – or product – must contain exactly five Components.
Just as we can qualify XFDU and SAFE as abstract formats, we can say the role of a SAFE Specialization is to constrain as precisely as possible the characteristics of a specific product-type. The SAFE Specialization has to be the equivalent of a specification for a product-type: all products of a same type have to follow the rules defined by the SAFE Specialization.

5.3 **SAFE Information Model**

![SAFE Information Model Diagram]

A SAFE Product gathers product basic information, such as the processing history, the platform / instrument identification and the acquisition period of the EO data.

The EO data is always completed by Fixity Information and a full Representation Information (these metadata are mandatory, according to the OAIS Reference Model). It may be completed by orbital metadata, an index for access efficiency, a frame set, grid references and quality information, plus any other metadata.
5.4 **SAFE and XFDU Types**

We have seen that as SAFE is a restriction of XFDU, it redefines XFDU Types. For example, we have seen how to constrain the contents of the Metadata Section.

XFDU is a format for storing data. It’s important to understand that XFDU is only a container: nowhere in its specifications does it mention what you put there, i.e. the data itself.

Not only does SAFE restrict the use of the XFDU container for the specific needs of earth observation (by redefining the XFDU Types), it also defines types for the data to be stored within the XFDU container: the SAFE Types.

Here for example is the “safe:startTimeType”, a simple type which corresponds to the oldest data of the product. We can also find in this example the declaration of the “safe:startTime” element of this type:

```xml
<xs:simpleType name="startTimeType">
  <xs:annotation>
    <xs:documentation>
      Time of the oldest data within the current SAFE product.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:dateTime"/>
</xs:simpleType>
```

*Figure 22 - Example of SAFE startTimeType*

SAFE also defines complex types, such as “safe:instrumentType”:

```xml
<xs:complexType name="instrumentType">
  <xs:annotation>
    <xs:documentation>
      Reference to an instrument belonging to an acquiring platform. Conversely to the platform element, there is no absolute standard identifying or referencing the instrument. Therefore, only the usual instrument family name is used to identify the instrument related to the product data. Both instrument sub-elements (i.e. familyName and number) are to be considered as the key to perfectly identify the sensor.
    </xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="familyName" type="safe:instrumentFamilyNameType"/>
    <xs:element name="number" type="safe:instrumentNumberType" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```
SAFE provides a set of typed data, some of which have authorized ranges of values.

SAFE types are the basetypes for the SAFE elements, which will be included in the manifest.

SAFE elements (declared in the “http://www.esa.int/safe/1.3” namespace) must be wrapped within the XFDU elements. Indeed, XFDU provides a mechanism for wrapping data using the “xmlData” element from “xmlDataType”.

Please note that not all SAFE elements (from SAFE types) can be wrapped in any position within the “xmlData” element. Only a precise list of SAFE elements is authorized as xmlData sub-elements:
• processing (typed processingType);
• platform (typed platformType);
• acquisitionPeriod (typed acquisitionPeriodType);
• orbitReference (typed orbitReferenceType);
• frameset (typed framesetType);
• gridReference (typed gridReferenceType);
• qualityInformation (typed qualityInformationType).

This does not mean that these elements are the only elements authorized as xmlData sub-elements. This means that these are the only SAFE elements authorized as xmlData sub-elements.

5.5 SAFE Specializations Types

SAFE Specializations can also provide new typed data, defined just for their products.

Here is an example of a metadataObject wrapping one of these new typed data (an element declared in a namespace other than “http://www.esa.int/safe/1.3” namespace):

```
<metadataObject ID="ra2GeneralInformation"
    classification="DESCRIPTION" category="DMD">
    <metadataWrap textInfo="Acquisition Period" vocabularyName="SAFE" mime="text/xml">
        <xmlData>
            <ra2:level2ProductInformation>
                <ra2:time>
                    <ra2:firstRecord>2004-12-17T02:24:56.231304Z</ra2:firstRecord>
                    <ra2:lastRecord>2004-12-17T04:03:35.971401Z</ra2:lastRecord>
                </ra2:time>
                <ra2:passNumber>0</ra2:passNumber>
                <ra2:rfHardwareConfiguration>A</ra2:rfHardwareConfiguration>
                <ra2:hpaHardwareConfiguration>A</ra2:hpaHardwareConfiguration>
                <ra2:validMeasurements>10000</ra2:validMeasurements>
            </ra2:level2ProductInformation>
        </xmlData>
    </metadataWrap>
</metadataObject>
```

Figure 26 - Example of SAFE ra2GeneralInformation metadataObject

It is also possible to integrate these new elements within the SAFE elements when this is permitted (which is the case in the instrument example below):

```
<xs:complexType name="instrumentType">
    <xs:sequence>
        <xs:element name="familyName"
            type="safe:instrumentFamilyNameType"/>
        <xs:element name="number"
            type="safe:instrumentNumberType"
            minOccurs="0"/>
    </xs:sequence>
</xs:complexType>
```
These new types are not provided by SAFE and are not declared in the http://www.esa.int/safe/1.3 namespace. They can be provided by a SAFE specialization if needed. It is the job of the SAFE specializer to provide the elements/types and to define a namespace for them.

5.6 **SAFE Objects**

A SAFE Product gathers SAFE Objects. There are two kinds of SAFE Objects: SAFE Data Objects and SAFE Metadata Objects.

5.6.1 **SAFE DATA OBJECTS**

A SAFE Data Object is composed of:

- a `dataObject` element of the SAFE Manifest;
- a Data Component.
linked together via the `dataObject/byteStream/fileLocation/@href` attribute.

![Diagram of SAFE Data Object](image)

*Figure 28 - SAFE Data Object*

5.6.2 SAFE METADATA OBJECTS

A SAFE Metadata Object may be a Wrapped Metadata Object or a Referenced Metadata Object.

5.6.2.1 SAFE Wrapped Metadata Object

A SAFE Wrapped Metadata Object is composed of a single `metadataObject` element of the SAFE Manifest.
The `metadataObject` element can wrap within its `xmlData` sub-element any top SAFE elements, such as “safe:platform”, “safe:processing”, “safe:acquisitionPeriod” etc.

### 5.6.2.2 SAFE Referenced Metadata Object

A SAFE Referenced Metadata Object can be:

- an XML Schema Object, composed of:
  - a `metadataObject` element of the SAFE Manifest;
  - an XML Schema Component.

linked together via the `metadataObject/metadataReference/@href` attribute.
Figure 30 - SAFE XML Schema Object

- a Metadata Object, composed of:
  - a `metadataObject` element of the SAFE Manifest;
  - a `dataObject` element of the SAFE Manifest;
  - a Metadata Component.

linked together via the `dataObject/byteStream/fileLocation/@href` attribute.

Figure 31 - SAFE Metadata Object
5.7 **SAFE Control Books**

You will find a lot of books about SAFE apart from this one. We will see in the following sections the two main SAFE volumes, which are normative. The Core Specifications provide SAFE rules, while the Recommendations for Specializations is a best practices guideline for those who want to develop new SAFE specializations. There are plenty of other volumes, each providing one or several SAFE specializations. Apart from the Recommendations for Specializations volume, all have the same content structure.

5.7.1 **CORE SPECIFICATIONS**

The normative SAFE specification is fully contained in the *Standard Archive Format for Europe (SAFE) – Core Specifications – PGSI-GSEG-EOPG-FS-05-0001* (see §Bibliography). The volume is composed of three parts:

- “Structures”;
- “Manifest Type Reference”
- “SDF Type Reference”.

It is completed by a series of appendices.

5.7.1.1 **Structures**

The “Structures” part of the Core Specifications describes the relationship between SAFE and XFDU and provides the information, logical and physical models. It also contains the rules concerning the manifest, components as well as the rules to link them.

5.7.1.2 **Manifest Type Reference**

The “Manifest Type Reference” part of the Core Specifications provides the entire set of SAFE types and the XFDU types redefined for the specific needs of SAFE.

For each type, there are:

- Its name and a short description;
- A structure diagram (for complex types only);
- A table listing its attributes if there are any (for complex types only);
- A table describing its characteristics (for simple types only);
A documentation;
An XML schema fragment.

Figure 32 - Example of SAFE Type in the Manifest Type Reference

5.7.1.3 SDF Type Reference

The “SDF Type Reference” part of the Core Specifications provides the entire set of SDF types.
For each type, just as for SAFE and XFDU types, there are:

- Its name and a short description;
- A structure diagram (for complex types only);
- A table listing its attributes if there are any (for complex types only);
- A table describing its characteristics (for simple types only);
- A documentation;
- An XML schema fragment.

5.7.1.4 Appendixes

XML schemas containing the definitions of the SAFE, XFDU and SDF types are provided in appendix.
A sample manifest and an example of SDF schema is also provided.

5.7.2 RECOMMENDATIONS FOR SPECIALISATIONS

This book provides rules and guidelines for designing and implementing a SAFE Specialization and intends to provide the necessary guidance to assure a consistent and homogeneous development of SAFE Specializations across projects and organizations.

The volume is composed of seven chapters:

- the first chapter provides an overview of a SAFE Specialization;
- chapter 2 provides recommendations for a SAFE Product to be well structured;
- chapter 3 provides recommendations for manipulation of all Manifest Types;
- chapter 4 provides recommendations for the representation Information of Components;
- chapter 5 provides recommendations for identification of a SAFE Product;
- chapter 6 provides recommendations for namespaces used in SAFE;
- the last chapter provides recommendations to define filename of Components, name of different entities gathered by a SAFE Product.

5.7.3 SPECIALISATIONS

5.7.3.1 Multiple SAFE Specialization Structures

For each specialization in the volume, a “Specialization-Name Structures” part (e.g. “ERS AMI SAR Level-0 Structures” or “ENVISAT GOMOS TRA Level-1 Structures”) describes as
accurately as possible the structure and contents of the product and provides the product’s information and physical models. A complete description of the manifest and components contents is also provided in tables.

5.7.3.2 Manifest Type Reference

The “Manifest Type Reference” part provides the entire set of SAFE types and the XFDU types redefined for the specific needs of the SAFE specializations included in the volume. For each type, as in the Standard Archive Format for Europe (SAFE) – Core Specifications, a documentation, a structure diagram etc. are provided.

Figure 33 - Example of XFDU Type in the Manifest Type Reference
5.7.3.3 Appendix

All the XML schemas for the specializations contained in the volume, whether they describe the manifest or the components, as well as examples of manifest, are attached in appendix.

5.8  The SAFE Suite

5.8.1 ITS STRUCTURE

The SAFE-Suite follows the following structure:

![SAFE Suite Structure Diagram]
5.8.1.1 Docs

The “docs” directory contains the SAFE Java & C++ API documentations.

The java “index.html” can be find in the “java” directory:

safe-suite/docs/java/index.html

The c++ “index.html” can be find in the “c++” directory:

safe-suite/docs/c++/index.html

5.8.1.2 Etc

The “etc” directory contains the property files for the SAFE-Suite and for Apache’s Log4j:

safe-suite/etc/log4j.properties
safe-suite/etc/safe.properties

5.8.1.3 Include

The “include” directory contains the SAFE C++ API headers:

safe-suite/include/.*\h

5.8.1.4 Lib

The “lib” directory contains the SAFE C++ API shared objects and the SAFE Java API jars:

safe-suite/lib/java/.*\jar
safe-suite/lib/c++/LINUX/.*\so

5.8.1.5 Resources

The “resources” directory contains a very large number of files from various types.
1. The “docbook” directory contains all files used for the generation of the SAFE Control books (including XML, SVG, PNG files etc.):

   ```
   safe-suite/resources/docbook/*/.*
   ```

   This directory contains all the book.xml files used for the generation of the Control Books by the SAFE Publisher.

2. The “examples” directory contains a complete set of sample manifests:

   ```
   Example of manifest file contained in the “examples” directory:
   • safe-suite-1-3/resources/examples/safe/int/esa/safe/1.3/landsat/tm/level-0/manifest.safe
   ```

   There is one example for each SAFE specialization.

3. The “owl” directory contains ontologies which describe the SAFE Core and the SAFE Specializations.

   ```
   Example of OWL file contained in the “owl” directory:
   • safe-suite-1-3/resources/owl/int/esa/safe/safe-core.owl
   ```

   These ontologies are mainly used by the SAFE Toolbox.

4. The “xql” directory contains all XQL files used by the different modules of SAFE (Transform, Publisher).

   ```
   Example of XQL file contained in the “xql” directory:
   • safe-suite-1-3/resources/xql/int/esa/safe/1.3/envisat/manifest.xql
   ```

   XQL files contain XQuery functions that can be used by the different modules of SAFE for manipulating data. An example is a function that create an ASCII date from an “xs:dateTime”.

5. The “xsd” directory contains all XML schemas relative to the XFDU and SAFE types (relative to the description of the SAFE manifest). This includes types provided by organizations such as Opengis (for GML types referenced within some SAFE types). In addition, the XML schema of SDF types is provided in this “xsd” directory.

   ```
   Examples of XML schemas contained in the “xsd” directory:
   • safe-suite-1-3/resources/xsd/int/esa/safe/1.3/ers/gome/egoc/level-0/xfdu.xsd
   • safe-suite-1-3/resources/xsd/int/esa/safe/1.3/spot/hrv/safe.xsd
   • safe-suite-1-3/resources/xsd/ord/ccds/xfdu/xfdu.xsd
   • safe-suite-1-3/resources/xsd/net/opengis/gml/3.0.0/base/gmlBase.xsd
   • safe-suite-1-3/resources/xsd/fr/gael/drb/impl/sdf/sdf-20020222.xsd
   ```

   The purpose of these XML schemas is to validate any SAFE Manifest.
6. The “xsd-component” directory contains all SDF schemas for the components.

   Examples of SDF schemas contained in the “xsd-component” directory:
   • safe-suite-1-3/resources/xsd-component/int/esa/safe/1.3/landsat/mss/level-0/measurement.xsd
   • safe-suite-1-3/resources/xsd-component/int/esa/safe/1.3/spot/spot-object-types.xsd

These SDF schemas shall be copied within the product when a native product is transformed into a SAFE product.

7. The “xsd-native” directory contains SDF schemas for the native product-types (the product in their original format before transformed into SAFE products).

   Examples of SDF schemas contained in the “xsd-native” directory:
   • safe-suite-1-3/resources/xsd-native/int/esa/envisat/asar/app/level-1/ASA_APP_1P.xsd

Transformers can extract from the native products (via these SDF schemas) any data and put it in the SAFE Manifest or anywhere else in the transformed product.

8. And finally, there is a single file in the “resources” directory, the “product-definition.xml”:

   safe-suite/resources/product-definition.xml

This file is a configuration file used by the different modules of SAFE (Transform, Publisher). It contains information about the product structure, such as the location of the SDF schemas to be copied during the transformation of the native product to the SAFE product, additional metadataObjects to be included within the manifest, etc.).

5.8.2 INTEGRATION OF A NEW SPECIALISATION

Any user wishing to create a new SAFE specialization should first read very carefully the Standard Archive Format for Europe (SAFE) – Recommendations for Specializations – PGSI-GSEG-EOPG-FS-05-0002 (see §Bibliography).

Creating a new SAFE specialization means mainly creating a set of files which describe the specialization.

The table below gives the minimum list of files to be created and their location in the SAFE-Suite.
The *xfdu.xsd* redefining XFDU types for the specific needs of the specialization (plus any number of XML schemas for the description/validation of the manifest file) | resources/xsd

A **complete specification** (i.e. a file document) of the SAFE Specialization, including description of the manifest file and binary description of all components | resources/

---

**Figure 35 - Table of Files/Location Creating a New Specialization (mandatory)**

The following table gives a list of files that is recommended to create and their location in the SAFE-Suite.

<table>
<thead>
<tr>
<th>List of Files</th>
<th>File Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>At least one <strong>SDF schema</strong> for each component of the specialization (plus any number of included SDF schemas). These SDF schemas (which can replace the mandatory binary description of the <strong>complete specification</strong> listed in the previous table) shall be copied within the SAFE Product during the transformation of a native product to a SAFE product</td>
<td>resources/xsd-component</td>
</tr>
<tr>
<td>A <strong>sample manifest</strong> for the specialization</td>
<td>resources/examples</td>
</tr>
<tr>
<td>All files necessary to generate a <strong>Control Book</strong> (which can replace the <strong>complete specification</strong> listed in the previous table)</td>
<td>resources/docbook/</td>
</tr>
</tbody>
</table>

**Figure 36 - Table of Files/Location Creating a New Specialization (recommended)**
6 SAFE TOOLS

6.1 SAFE I/O API

In addition to the SAFE specifications of the Core level and the Specialization level, a set of software APIs dedicated to SAFE product users is available. The main use cases of these software components are:

- Create a SAFE product;
- Open a SAFE product;
- Move a SAFE product;
- Validate a SAFE product;
- Add/Remove/Browse Content Units of the Information Package Map;
- Add/Remove/Retrieve the Metadata Objects;
- Add/Remove/Retrieve the Data Objects;
- Browse/Select/Extract information from either Binary or XML objects;
- Identify/Sort Quality Information to a specific part of the objects.

The top level APIs are provided in Java and C++, the C++ version being a wrapper that requires the Java components. They have been built on top of several other components of interest:

- XFDU Java API: this API provides the general features common to all XFDU packages. This API has been developed jointly with the SAFE Java API. This layer is presently used in the framework of the activity of the CCSDS Information Packaging and Registries (IPR) Working Group in support to the validation of the XFDU recommendation developed by this group. It may also support users that intend to develop applications compliant with all XFDU packages, whether complying SAFE or not.

- XFDU C++ Wrapper: a C++ wrapper on top of the XFDU Java API. Most of the functionality is preserved from the Java API, apart the capability of browsing the binary contents.

- Data Request Broker (DRB): from which SAFE inherits the capability to access data independently from their formats. DRB supports in particular the SDF markup language used for annotating the XML Schema documents acting as representation information of the SAFE product objects.
The following figure represents schematically the architecture of the SAFE API suite:

![SAFE API suite Architecture](image)

**Figure 37 - SAFE API suite Architecture**

### 6.2 SAFE Toolbox

#### 6.2.1 PRESENTATION

The SAFE Toolbox is an application which allows to create, open, explore and edit SAFE products. Different views are available to explore the document through several ways. Other functionalities as Create, Open etc. are provided.

Two modes are proposed: Visualization mode (read-only mode) and Edition mode.

When the application starts, it shows a welcome panel gathering all the possible action:

- “Load Manifest”: it allows to load a safe product by opening a file chooser;
- “Recent Loads”: it lists the recent loaded products;
- “Settings”: allows access to the SAFE Toolbox settings,
- “Help”: the toolbox help (with direct access to the Software User Manual).
This welcome panel allows drag and drop action to load products.

6.2.2 MANIFEST, CONTENT UNIT AND STYLE SHEET VIEWS

Once a product loaded, the SAFE Toolbox main window is separated in two parts:

- Navigation panel, which contains manifest view (1).
- Working panel, which contains working views (Content unit view, style sheet view, editors) in different tabs (2).
The “File” menu allows users to open, close or save Safe products; the “Tools” menu lets users access the application tools such as search; and the “Help” menu allows access to the documentation. If a product is closed, the application returns to the welcome panel.

6.2.2.1 Manifest View

Under this view, it is possible to explore the manifest file node by node: for each node and attribute, available information is displayed: value, attribute overview, namespace.

As we can see on the figure above, the opened file contains the three main sections: informationPackageMap, metadataSection and dataObjectSection. Each section can be expanded or collapsed.

The up-right corner popup menu allows selection of columns that can be hided or displayed in the tree:

- Nodes: the tree representation of nodes;
- Values: values carried out by the nodes;
- Attributes Overview: an overview of attributes (useful to get nodes information without being opened);
- Namespaces: reports the node namespace.
6.2.2.2 Content Unit View

On the right part of the main window, ContentUnit view is opened by default. Using this view, it is possible to explore the opened document through a logical representation of the document.

The figures above show an example of a ContentUnit view.

![Example of a SAFE Toolbox Manifest View](image-url)
Figure 41 - Example of a SAFE Toolbox Content Unit View

Figure 42 - Symbols used in the Content Unit View
The content unit view shows the entire product from the root content unit, to its pointed metadata and data objects. It is able to follow the ID references to resolve the identifier and show the pointed metadata.

In the figure below, the main ContentUnit Section (from InformationPackageMap section) is represented by the yellow rectangle here, named “ENVISAT MERIS FR Level 0”: it corresponds to the opened product. References on metadata are represented by green circles, while his content units are represented by blue rectangles.

All graph elements can be moved, repulsions happened if nodes are too close.

When configured in the toolbox settings manifest view and content unit view are synchronized. This allows changing manifest view selection when selecting a node in the content unit view and vice-versa.

6.2.2.3 Style Sheet View

On the right part of the main window, this view is displayed by clicking on tab “Style Sheet View”. It displays a summary of the product. The SAFE manifest file is always considered as the root context node of the stylesheet. The output of the stylesheet is HTML and SAFE toolbox implements an html viewer able to manage connection with stylesheet processors. The figure above shows an example of a Style Sheet view.
6.2.3 BINARY VISUALIZATION AND EXTRACTION

Binary visualisation is possible by clicking the separate view menu when selecting a data object in the manifest view. This view allows browsing inside the binary components. This view is able to display a very large number of elements without memory overflow with a customised specific limiter.
6.2.4 IMAGE TOOL

When the opened product contains images, a new menu appears in the menu Bar: “Image”. This menu lists all available images in the product: by clicking on one of them, it opens the selected image in a new window.
6.3 **SAFE Publisher**

6.3.1 OVERVIEW

The publisher is an application which generates the SAFE Control Books. The entire set of SAFE specifications listed in the bibliography (from PGSI-GSEG-EOPG-FS-05-0001 to PGSI-GSEG-EOPG-FS-05-0021) was generated by the SAFE publisher.
The principle of the publisher is to convert a docbook file into a document. The output formats supported are PDF and HTML.

6.3.2 DOCBOOK LANGUAGE

DocBook is a semantic markup language for technical documentation.

As a semantic language, DocBook documents do not describe what their contents "look like," but rather the meaning of those contents.

For example, rather than explaining how the abstract for an article might be visually formatted, DocBook simply says that a particular section is an abstract.

It is up to an external processing tool or application to decide where on a page the abstract should go and what it should look like. (And, indeed, to decide whether or not it should be included in the final output at all.)

6.3.3 HOW TO USE THE PUBLISHER: THE BOOK FILE

Each SAFE specification is written as a docbook file named “book.xml”. This file can be placed in the SAFE-Suite in the “resources/docbook” directory (or in another place to be defined by the creator of the specialization):

    safe-suite/resources/docbook/

Example of book.xml files contained in the “docbook” directory:


Here is an example of book.xml:

```xml
<book status="final" id="PGSI-GSEG-EOPG-FS-05-0001"
    xmlns:safe="http://www.esa.int/safe/xsl-utils-200805">
  <bookinfo>
    <title>Standard Archive Format for Europe</title>
    <subtitle>Core Specifications</subtitle>
    <orgname>European Space Agency (ESA)</orgname>
    <authorgroup>
      <author>
        <firstname>Stephane</firstname>
        <surname>Mbaye</surname>
        <affiliation>
          <orgname>GAE Consultant</orgname>
        </affiliation>
      </author>
      <author>
        <firstname>Mathias</firstname>
      </author>
    </authorgroup>
  </bookinfo>
</book>
```
The book.xml is the only file a creator of a new specialization has to write to generate a Control Book. The publisher converts each of the book.xml tags into visually formatted text.

For example, the publisher converts all the part titles:

```
<part>
  <title>Structures</title>
  (...)
</part>
```

**Figure 47 - Example of book.xml**

**Figure 48 - Example of Part Title**
into titles with the following characteristics: centered, Arial font, size 28, italics and bold.

The obvious advantage of this approach is that all the SAFE specification are in the same format. Moreover, in order to write a new volume, an author has only to write a new book.xml, which only contains the contents (the form is taken care of by the publisher).
APPENDIX: XML SCHEMA OVERVIEW

7.1 Introduction

XML-Schema is a language developed by the W3C (World Wide Web Consortium). It is designed to define constraints (structure, content and semantics) and validate XML documents. XML-Schema is based on simple XML syntax.

The purpose of an XML schema is to define a class of XML documents, and so the term “XML instance document” is often used to describe an XML document that conforms to a particular schema.

7.2 Basics

7.2.1 OUR FIRST EXAMPLE: THE PLATFORM SCHEMA (PLATFORM.XSD)

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="platform" type="platformType"/>
  <xs:element name="familyName" type="familyNameType"/>
  <xs:complexType name="platformType">
    <xs:sequence>
      <xs:element name="nssdsIdentifier" type="xs:string"/>
      <xs:element ref="familyName" minOccurs="0"/>
      <xs:element name="instrument" type="instrumentType" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

```xml
<xs:simpleType name="familyNameType">
  <xs:restriction base="xs:string"/>
</xs:simpleType>
```

```xml
<xs:complexType name="instrumentType">
  <xs:sequence>
    <xs:element name="number" type="xs:integer"/>
    <xs:element name="name" type="xs:string"/>
  </xs:sequence>
  <xs:attribute name="mode" type="modeType"/>
</xs:complexType>
```

```xml
<xs:simpleType name="modeType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="active"/>
    <xs:enumeration value="passive"/>
  </xs:restriction>
</xs:simpleType>
```
The Platform XML schema consists of a schema element and a variety of sub-elements, most notably “element”, “complexType” and “simpleType” which determine the appearance of elements and their content in instance documents.

Each of the elements in the schema has a prefix “xs”: which is associated with the XML Schema namespace through the declaration:

```
xmlns:xs="http://www.w3.org/2001/XMLSchema",
```

that appears in the schema node.

The prefix “xs” is used by convention to denote the XML Schema namespace, although any prefix can be used. The same prefix, and hence the same association, also appears on the names of built-in simple types, e.g. “xs:string”. The purpose of the association is to identify the elements and simple types as belonging to the vocabulary of the XML Schema language rather than the vocabulary of the schema author. For the sake of clarity in the text, we just mention the names of elements and simple types (e.g. “simpleType”), and omit the prefix.

### 7.2.2 COMPLEX TYPE DEFINITIONS, ELEMENT & ATTRIBUTE DECLARATIONS

In XML Schema, there is a basic difference between:

- complex types: which allow elements in their content and may carry attributes, and…
- simple types: which cannot have element content and cannot carry attributes.

New complex types are defined using the “complexType” element, new elements are declared using the “element” element, and attributes are declared using the “attribute” element.

For example, instrumentType is defined as a complex type, and within the definition of instrumentType we see two element declarations and one attribute declaration.

The consequence of this definition is that any element appearing in an instance whose type is declared to be instrumentType must consist of two elements and one attribute. These elements must be called number and name as specified by the values of the declarations’ name attributes,
and the elements must appear in the same sequence (order) in which they are declared. The first of these elements will contain an integer, and the second will contain a string. The element whose type is declared to be instrumentType may appear with an attribute called mode which must contain one of the following string: “active” or “passive”.

The instrumentType definition contains only declarations involving the simple types: integer and string. In contrast, the platformType definition contains element declarations involving complex types, e.g. instrumentType, although note that both declarations use the same “type” attribute to identify the type, regardless of whether the type is simple or complex.

Attribute declarations (e.g. mode in instrumentType) must reference simple types because, unlike element declarations, attributes cannot contain other elements or other attributes.

Attributes are always simple typed while elements may be simple or complex typed.

7.2.3 REFERENCE TO A GLOBAL ELEMENT

The element declarations we have described so far have each associated a name with an existing type definition. Sometimes it is preferable to use an existing element rather than declare a new element, for example:

<xs:element ref="familyName" minOccurs="0"/>

This declaration references an existing element, familyName, that was declared elsewhere in the platform.xsd schema. In general, the value of the “ref” attribute must reference a global element, i.e. one that has been declared under “schema” rather than as part of a complex type definition. The consequence of this declaration is that an element called familyName may appear in an instance document, and its content must be consistent with that element's type, in this case,”string”.

7.2.4 OCCURRENCE CONSTRAINTS

The platform/familyName element is optional within PlatformType because the value of the “minOccurs” in its declaration is 0. In general, an element is required to appear when the value of “minOccurs” is 1 or more.

The maximum number of times an element may appear is determined by the value of a “maxOccurs” attribute in its declaration. This value may be a positive integer such as 54, or the term “unbounded” to indicate there is no maximum number of occurrences.

The default value for both the “minOccurs” and the “maxOccurs” attributes is 1.

Attributes may appear once or not at all, but no other number of times, and so the syntax for specifying occurrences of attributes is different than the syntax for elements. In particular,
attributes can be declared with a “use” attribute to indicate whether the attribute is “required”, “optional” or “prohibited”.

7.2.5 CONTENT MODELS

The definitions of complex types in the Platform schema all declare “sequences” of elements that must appear in the instance document. XML Schema also allows declaring “choices” of element.

To illustrate, let’s say that a platform could be identified either by its nssdcIdentifier or by its familyName:

```xml
<xs:complexType name="platformType">
  <xs:sequence>
    <xs:choice>
      <xs:element name="nssdcIdentifier" type="xs:string"/>
      <xs:element ref="familyName" minOccurs="0"/>
    </xs:choice>
    <xs:element name="instrument" type="instrumentType" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
```

**Figure 50 - Nested Choice and Sequence Groups**

The choice group element allows only one of its children to appear in an instance. One child is a nssdcIdentifier, and the second child is a familyName. Hence, in an instance document, the platformType element must contain either a nssdcIdentifier element or a familyName element.

The choice group is followed by the instrument element declaration, and both the choice group and the element declaration are children of a sequence group.

7.2.6 BUILT-IN DATATYPES

The Platform schema declares several elements and attributes that have simple types. Some of these simple types, such as string and integer, are built in to XML Schema, while others are derived from the built-in’s. For example, the mode attribute has a type called modeType that is derived from string.

Both built-in simple types and their derivations can be used in all element and attribute declarations.

Here is an uncompleted list of simple types built in to XML Schema:
• date / time;
• duration;
• boolean;
• string;
• ID / IDREF;
• float;
• integer…

New simple types are defined by deriving them from existing simple types (built-in's and derived). In particular, we can derive a new simple type by restricting an existing simple type, in other words, the legal range of values for the new type are a subset of the existing type's range of values. We use the “simpleType” element to define and name the new simple type. We use the “restriction” element to indicate the existing (base) type, and to identify the "facets" that constrain the range of values.

Main facets are “length”, “pattern” and “enumeration”.

The Platform schema contains an example of a simple type definition. A new simple type called modeType is derived (by restriction) from the simple type string. Furthermore, we constrain the values of modeType using a facet called “enumeration” conjunction with the values “active” and “passive”.

### 7.3 Advanced Concepts

#### 7.3.1 TARGET NAMESPACES

XML Schema allows declaring a target namespace (indicated by the value of the “targetNamespace” attribute). Qualification of local elements and attributes can be globally specified by a pair of attributes, “elementFormDefault” and “attributeFormDefault”, on the schema element (with “qualified” and “unqualified” values).

```xml
<schema xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:platform="http://www.example.com/platform"
    targetNamespace="http://www.example.com/platform"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">
    <element name="platform" type="platform:platformType"/>
    <element name="familyName" type="platform:familyNameType"/>
    <complexType name="platformType">
        <sequence>
            <element name="nssdsIdentifier" type="string"/>
        </sequence>
    </complexType>
</schema>
```
At the beginning of the schema, we declare the elements platform and familyName. They are included in the schema's target namespace. The platform element's type is prefixed, for the same reason that instrumentType is prefixed. In contrast, the familyName element's type, string, is not prefixed.

The schema contains a default namespace declaration, and so unprefixed types such as string and unprefixed elements such as element and complexType are associated with the default namespace “http://www.w3.org/2001/XMLSchema”. In fact, this is the target namespace of XML Schema itself, and so a processor will know to look within the schema of XML Schema – otherwise known as the “schema for schemas” – for the definition of the type string and the declaration of the element called element.

7.3.2 MULTIPLE DOCUMENTS FOR A SINGLE SCHEMA: THE “INCLUDE” ELEMENT

As schemas become larger, it is often desirable to divide their content among several schema documents for purposes such as ease of maintenance, access control, need for factorization and readability.

The various platform and instrument constructions are now contained in two schema files. In order to include one construction inside the other, the first schema contains the “include” element:

```
<element ref="platform:familyName" minOccurs="0"/>
<element name="instrument" type="platform:instrumentType"
    maxOccurs="unbounded”/>
</sequence>
</complexType>

<complexType name="familyNameType">  
  <restriction base="string”/>
</complexType>

</schema>

Figure 51 - Platform XML Schema with Target Namespace
The schema containing the instrument constructs is:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:platform="http://www.example.com/platform"
           targetNamespace="http://www.example.com/platform"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified">
  <xs:include schemaLocation="instrument.xsd"/>
  <xs:element name="platform" type="platform:platformType"/>
  <xs:element name="familyName" type="platform:familyNameType"/>
  <xs:complexType name="platformType">
    <xs:sequence>
      <xs:element name="nssdsIdentifier" type="xs:string"/>
      <xs:element ref="platform:familyName" minOccurs="0"/>
      <xs:element name="instrument" type="platform:instrumentType" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:simpleType name="familyNameType">
    <xs:restriction base="xs:string"/>
  </xs:simpleType>
</xs:schema>
```

**Figure 52 - Platform XML Schema – Including instrument.xsd Schema**

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:platform="http://www.example.com/platform"
           targetNamespace="http://www.example.com/platform"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified">
  <xs:complexType name="instrumentType">
    <xs:sequence>
      <xs:element name="number" type="xs:integer"/>
      <xs:element name="name" type="xs:string"/>
    </xs:sequence>
    <xs:attribute name="mode" type="platform:modeType"/>
  </xs:complexType>
  <xs:simpleType name="modeType">
    <xs:restriction base="xs:string">  
      <xs:enumeration value="active"/>
      <xs:enumeration value="passive"/>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

**Figure 53 - Instrument XML Schema – Included by platform.xsd Schema**
The effect of this include element is to bring in the definitions and declarations contained in instrument.xsd, and make them available as part of the platform schema target namespace.

The one important caveat to using include is that the target namespace of the included components must be **the same as the target namespace** of the including schema.

### 7.3.3 MULTIPLE DOCUMENTS FOR A SINGLE SCHEMA: THE “IMPORT” ELEMENT

The import mechanism that we describe in this section is an important mechanism that enables schema components from **different target namespaces** to be used together, and hence enables the schema validation of instance content defined across multiple namespaces.

Here is a new Acquisition schema that uses the platform element declared within the Platform schema:

```xml
<xsd:schema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" 
    xmlns:platform="http://www.example.com/platform" 
    xmlns:eo="http://www.example.com/eo" 
    targetNamespace="http://www.example.com/oe" 
    elementFormDefault="qualified" 
    attributeFormDefault="unqualified">

    <xsd:import namespace="http://www.example.com/platform" 
    schemaLocation="platform.xsd"/>

    <xsd:element name="acquisition" type="eo:acquisitionType"/>

    <xsd:complexType name="acquisitionType">
    <xsd:sequence>
        <xsd:element name="date" type="xsd:dateTime"/>
        <xsd:element ref="platform:platform"/>
    </xsd:sequence>
    </xsd:complexType>

</xsd:schema>
```

*Figure 54 - Acquisition XML Schema*

In this example, we can reuse the `platform` element declared in the Platform by referencing that element in the `<xs:element ref="platform:platform"/>` declaration.

Concretely, we use the import element to identify `platform`'s target namespace “http://www.example.com/platform”, and we associate the namespace with the prefix “platform” using a standard namespace declaration. The element `platform`, declared in the namespace “http://www.example.com/platform”, may then be referenced as `platform:platform` in any of the report schema's definitions and declarations.
7.3.4 REDEFINING TYPES

In previous sections, we described how to include definitions and declarations obtained from external schema files having the same target namespace. The “include” mechanism enables to use externally created schema components "as-is", that is, without any modification.

The “redefine” mechanism enables to redefine simple and complex types that are obtained from external schema files. Like the “include” mechanism, “redefine” requires the external components to be in the same target namespace as the redefining schema.

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:platform="http://www.example.com/platform"
    targetNamespace="http://www.example.com/platform"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">
    <xs:redefine schemaLocation="platform.xsd">
        <xs:simpleType name="familyNameType">
            <xs:restriction base="platform:familyNameType">
                <xs:enumeration value="ENVISAT"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:redefine>
</xs:schema>
```

**Figure 55 - ENVISAT Platform XML Schema – Redefining platform.xsd**

The “redefine” element acts very much like the “include” element as it includes all the declarations and definitions from the platform.xsd file. The simple type definition “familyNameType” uses the “restriction” syntax to restrict the value of “familyNameType”. However, note that the base type is also “familyNameType”.

Outside of the redefine element, any such attempt to define a simple type with the same name (and in the same namespace) as the base from which it is being derived would cause an error. But in this case, there is no error, and the restricted definition of “familyNameType” becomes the only definition of “familyNameType”.

8 APPENDIX: SDF MARK-UPS

8.1 Introduction

The Structured Data File (SDF) is an implementation of DRB that handles data sources (file, stream, string literals) with a user-defined format. A data file is usually defined as a set of adjacent blocks with various lengths. A Structured Data File is a file containing ASCII or BINARY data which can be described as a hierarchy of data blocks. These blocks are organized in a "tree model" as a file system does with directories, sub-directories and files.

A logical description is first needed to set up the structure of the tree. To do so the XML-Schema language has been chosen.

8.2 Our First Example: the SDF Description of the "Platform List"

Let us take the example of an ASCII file containing a list of platforms, plus for each one: its name, the year it was launched and whether it is still active. Each platform is also given an identifier.

<table>
<thead>
<tr>
<th>ID</th>
<th>Platform Name</th>
<th>Launch Year</th>
<th>In Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ENVISAT</td>
<td>2002</td>
<td>true</td>
</tr>
<tr>
<td>2</td>
<td>ERS-1</td>
<td>1991</td>
<td>false</td>
</tr>
<tr>
<td>3</td>
<td>Landsat-1</td>
<td>1972</td>
<td>false</td>
</tr>
</tbody>
</table>

*Figure 56 - Example of the Platform List – The Table*

Here is the extract of the ASCII file corresponding to our table:

```plaintext
Id='   1'
PlatformName='ENVISAT       '
LaunchYear='2002'
Activity='true '

Id='   2'
PlatformName='ERS-1         '
LaunchYear='1991'
Activity='false'
```
First of all, we notice that the structure of the fields “ID/Platform Name/Launch Year/In Activity” is repeated several times. It’s a possibly unbounded sequence of records.

Secondly, we note that the size of each field is fixed from one record to another. For example, if the value of the “Platform Name” field has fewer digits than expected, blanks are inserted after this value.

Thus, we can describe the structure of these fields and give their size:

1. "ID" Field: 5 bytes;
2. "Platform Name" Field: 20 bytes;
3. "Launch Year" Field: 4 bytes;
4. "In Activity" Field: 5 bytes.

If we look more closely at the text file, we notice that the fields do not follow each other directly. They are separated by labels, i.e. textfields which are fixed for all records:

- "Id=' Label: 4 bytes (constant);
- "ID" Field: 5 bytes (variable);
- "'" + Carriage Return + "PlatformName='" Label: 16 bytes (constant);
- "Platform Name" Field: 20 bytes (variable);
- "'" + Carriage Return + "LaunchYear='" Label: 14 bytes (constant);
- "Launch Year" Field: 4 bytes (variable);
- "'" + Carriage Return + "Activity='" Label: 12 bytes (constant);
- "In Activity" Field: 5 bytes (variable);
- "'" + Carriage Return + Carriage Return Label: 3 bytes (constant).

We can now describe the physical structure of each record to the last byte:

- "Id=' Label: 4 bytes (constant);
- "ID" Field: 5 bytes (variable);
- "'" + Carriage Return + "PlatformName='" Label: 16 bytes (constant);
- "Platform Name" Field: 20 bytes (variable);
- "'" + Carriage Return + "LaunchYear='" Label: 14 bytes (constant);
- "Launch Year" Field: 4 bytes (variable);
- "'" + Carriage Return + "Activity='" Label: 12 bytes (constant);
- "In Activity" Field: 5 bytes (variable);
- "'" + Carriage Return + Carriage Return Label: 3 bytes (constant).

SDF allows us to represent the contents described above as a tree diagram:
and so, as an XML schema:

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:sdf="http://www.gael.fr/2004/12/drb/sdf">
   <xs:element name="platformList">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="platformRecord" type="platformRecordType" minOccurs="0" maxOccurs="unbounded">
               <xs:annotation>
                  <xs:appinfo>
                     <sdf:block>
                        <sdf:occurrence>unbounded</sdf:occurrence>
                     </sdf:block>
                  </xs:appinfo>
               </xs:annotation>
            </xs:element>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```
Figure 59 - Example of the Platform List – The SDF Schema
Generic Schema syntax is described in the previous chapter. Additional SDF tags are described below.

## 8.3 SDF Description

### 8.3.1 SDF NAMESPACE

SDF elements are declared in the namespace:

```
"http://www.gael.fr/2004/12/drb/sdf"
```

![Figure 60 - Example of SDF Namespace](image)

### 8.3.2 SDF BLOCK

All SDF elements must be nested in a block descriptor: a “block” element must be added inside the "appinfo" element. This is the mandatory information for the physical description.

```
<xs:element name="productName" type="xs:string">
  <xs:annotation>
    <xs:appinfo>
      <sdf:block>
        <sdf:encoding>ASCII</sdf:encoding>
        <sdf:length>10</sdf:length>
      </sdf:block>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

![Figure 61 - Example of SDF Block](image)

### 8.3.3 SDF LENGTH

The “length” element is used to set the length of an element.
Length value can be defined in bytes or in bits. The “unit” attribute can be added to set "byte" or "bit" unit. If the "unit" attribute is omitted, the "byte" unit is set by default.

**Figure 62 - Example of SDF ASCII Integer (4 bytes long)**

**Figure 63 - Example of SDF BINARY Unsigned Byte (2 bits long)**

### 8.3.4 SDF OCCURRENCE

The “occurrence” element is used to set the occurrence count of the element in the Structured Data File. The default value is set to 1 if omitted.
8.3.5 SDF ENCODING

The “encoding” element is used to set the encoding method of the element. The possible values are “ASCII” or “BINARY”. If the "encoding" element is omitted the encoding is inherited from the parent. The root has a “BINARY” encoding by default.

```xml
<xs:element name="recordCount" type="xs:integer">
  <xs:annotation>
    <xs:appinfo>
      <sdf:block>
        <sdf:encoding>BINARY</sdf:encoding>
        <sdf:length>4</sdf:length>
      </sdf:block>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

**Figure 65 - Example of SDF Encoding**

8.3.6 SDF OFFSET

The “offset” element is used to set the offset of an element. By default, the offset value is considered as a relative offset from the previous sibling element. This means a shift between the last byte of the last occurrence of the previous element and the first byte of the current one. In most cases, an element is following its previous sibling without any gap therefore no relative offset is needed.

Offset value can be defined in bytes or in bits. The "unit" attribute can be added to set "byte" or "bit" unit. If the "unit" attribute is omitted, the "byte" unit is set by default.

The “origin” attribute is used to set the position (absolute/relative) in the file:

- "previous": the offset is relative to the previous-sibling node;
- "parent": the offset is relative to the parent node;
- "root": the offset is absolute (e.g. from the beginning of the file).

```xml
<xs:element name="latitude" type="xs:double">
  <xs:annotation>
    <xs:appinfo>
      <sdf:block>
        <sdf:origin>previous</sdf:origin>
      </sdf:block>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```
8.3.7 SDF PADDINGS

The “padding” element is used to define a gap (offset) inside an element. Two types are available: the "header" padding and the "footer" padding. They are respectively the offset to skip before and after the element value to be extracted. They can be set together or only one of them.

8.3.8 SDF ARRAY

The “array” element is used when the datatype of the element is a “list”.

The “array” element must have the following children:

- “occurrence”, which define the size of the array.
- “length”, which define the length of each item in the array.
8.3.9 SDF SIGNATURE

The “signature” allows to select an element depending on its content. This node is necessary to handle elements with a variable content (e.g. a choice group). In that case, the SDF engine will select the first substitution with a valid signature.

In the following example:

- if the value (in the instance) of the “auxiliaryData” is 0, then the first element “dayModeMeasurementData” is selected;
- if the value of the “auxiliaryData” is 1, then the second element “nightModeMeasurementData” is selected;
- if the value of the “auxiliaryData” is neither 0 nor 1, then no element is selected.
8.4 **Our Second Example: the SDF Description of the “Measurement” Component**

Our second example is a binary Component named “measurement.dat”. This measurement contains a sequence of records. Each record contains several metadata fields, followed by an Information Source Packet (an ISP). An ISP is made of a Packet Header and a Packet Data Field. The complete description of the measurement follows:

<table>
<thead>
<tr>
<th>Element</th>
<th>Type</th>
<th>Length</th>
<th>Occurrence</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Measurement record:</strong></td>
<td><strong>Complex Type</strong></td>
<td></td>
<td>unbounded</td>
</tr>
<tr>
<td>ISP sensing time</td>
<td>dateTime</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>ground station reference time</td>
<td>dateTime</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>ISP length</td>
<td>unsigned short</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>CRC error count</td>
<td>unsigned short</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>RS error count</td>
<td>unsigned short</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td><strong>Information source packet (ISP):</strong></td>
<td><strong>Complex Type</strong></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Packet Header</td>
<td><strong>Complex Type</strong></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>packet version number</td>
<td>unsigned int</td>
<td>6 bits</td>
<td>1</td>
</tr>
<tr>
<td>packet type</td>
<td>unsigned byte</td>
<td>1 bit</td>
<td>1</td>
</tr>
<tr>
<td>data field header flag</td>
<td>unsigned byte</td>
<td>1 bit</td>
<td>1</td>
</tr>
</tbody>
</table>

**Figure 69 - Example of SDF Signature**
Packet data field | Complex Type | 1
---|---|---
data header | array of unsigned bytes (1 byte each) | 120 | 1
source data | array of unsigned bytes (1 byte each) | 5000 | 1

**Figure 70 - Example of the Measurement Component - The Table**

Here is the SDF description of the Measurement Component:

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:sdf="http://www.gael.fr/2004/12/drb/sdf">
  <xs:element name="measurement">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="measurementRecord" type="measurementRecordType" minOccurs="0" maxOccurs="unbounded">
          <xs:annotation>
            <xs:appinfo>
              <sdf:block>
                <sdf:occurrence>unbounded</sdf:occurrence>
              </sdf:block>
            </xs:appinfo>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="measurementRecordType">
    <xs:sequence>
      <xs:element name="ispSensingTime" type="dateTimeBinary12"/>
      <xs:element name="groundStationReferenceTime" type="dateTimeBinary12"/>
      <xs:element name="ispLength" type="unsignedShortBinary2"/>
      <xs:element name="crcErrorCount" type="unsignedShortBinary2"/>
      <xs:element name="rsErrorCount" type="unsignedShortBinary2"/>
      <xs:element name="informationSourcePacket" type="informationSourcePacketType"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="informationSourcePacketType">
    ...
  </xs:complexType>
</xs:schema>
```
<xs:sequence>
  <xs:element name="packetHeader" type="packetHeaderType"/>
  <xs:element name="packetDataField" type="packetDataFieldType"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="packetHeaderType">
  <xs:sequence>
    <xs:element name="packetVersionNumber" type="xs:unsignedByte">
      <xs:annotation>
        <xs:appinfo>
          <sdf:block>
            <sdf:length unit="bit">6</sdf:length>
          </sdf:block>
        </xs:appinfo>
      </xs:annotation>
    </xs:element>
    <xs:element name="packetType" type="xs:unsignedByte">
      <xs:annotation>
        <xs:appinfo>
          <sdf:block>
            <sdf:length unit="bit">1</sdf:length>
          </sdf:block>
        </xs:appinfo>
      </xs:annotation>
    </xs:element>
    <xs:element name="dataFieldHeaderFlag" type="xs:unsignedByte">
      <xs:annotation>
        <xs:appinfo>
          <sdf:block>
            <sdf:length unit="bit">1</sdf:length>
          </sdf:block>
        </xs:appinfo>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="packetDataFieldType">
  <xs:sequence>
    <xs:element name="dataHeader">
      <xs:annotation>
        <xs:documentation xml:lang="en">
          Data Header: An array of 120 unsigned bytes.
        </xs:documentation>
      </xs:appinfo>
      <sdf:block>
        <sdf:array>
Source Data : An array of 5000 unsigned bytes.

Date time (12 bytes).

Unsigned short (2 bytes).
<xs:simpleType name="unsignedByteBinary1">
  <xs:annotation>
    <xs:documentation>
      Unsigned Byte (1 byte).
    </xs:documentation>
    <xs:appinfo>
      <sdf:block>
        <sdf:encoding>BINARY</sdf:encoding>
        <sdf:length>1</sdf:length>
      </sdf:block>
    </xs:appinfo>
  </xs:annotation>
  <xs:restriction base="xs:unsignedByte"/>
</xs:simpleType>

Figure 71 - Example of the Measurement Component - The SDF Schema