



SDMX self-learning package No. 6

Student book

XML Based Technologies Used in SDMX

| | |
|------------------------|--|
| Produced by | Eurostat, Directorate B: Statistical Methodologies and Tools Unit B-5: Statistical Information Technologies |
| Last update of content | May 2010 |
| Version | 1.0 |

TABLE OF CONTENTS

| | | |
|----------|---|-----------|
| 1 | SCOPE OF THE STUDENT BOOK..... | 1 |
| 2 | XML, SCHEMAS AND NAMESPACES..... | 2 |
| 2.1 | OVERVIEW OF XML | 2 |
| 2.2 | XML NAMESPACE | 4 |
| 2.2.1 | <i>Basic concepts</i> | 4 |
| 2.2.2 | <i>Declaring namespaces</i> | 4 |
| 2.2.3 | <i>Applying namespaces</i> | 5 |
| 2.3 | XML SCHEMA (XSD) | 6 |
| 2.3.1 | <i>Basic concepts</i> | 7 |
| 2.3.2 | <i>Heretical schemas</i> | 9 |
| 2.3.3 | <i>Declaring schemas</i> | 9 |
| 2.4 | XML IN SDMX..... | 10 |
| 2.4.1 | <i>Structures</i> | 11 |
| 2.4.2 | <i>Data and Reference Metadata</i> | 11 |
| 2.5 | CREATING A SCHEMA FROM AN SDMX-ML STRUCTURE FILE | 12 |
| 2.6 | BASIC COMPONENTS OF SDMX FILES | 15 |
| 3 | XSL TRANSFORMATION (XSLT) | 17 |
| 3.1 | INTRODUCTION | 17 |
| 3.2 | STEP BY STEP: TRANSFORMATION BY EXAMPLE | 18 |
| 3.2.1 | <i>XML Copy Editor</i> | 18 |
| 3.2.2 | <i>From an SDMX-ML structure file to an HTML presentation</i> | 18 |
| 3.2.3 | <i>From an SDMX-ML structure file to an XML schema</i> | 20 |
| 3.2.4 | <i>From an SDMX-ML data file to an HTML presentation</i> | 24 |
| 4 | WEB SERVICES | 27 |
| 4.1 | INTRODUCTION | 27 |
| 4.2 | WSDL: WEB SERVICES DESCRIPTION LANGUAGE | 28 |
| 4.3 | SDMX RECOMMENDATIONS FOR WEB SERVICES USING SOAP TECHNOLOGY | 29 |
| 4.4 | SOAP MESSAGES FOR SDMX QUERY | 30 |
| 4.4.1 | <i>Envelope, Header and Body</i> | 31 |
| 4.4.2 | <i>Fault</i> | 32 |
| 5 | ANNEX | 34 |
| 5.1 | URL, URI AND URN..... | 34 |
| 6 | GLOSSARY..... | 35 |

1 **Scope of the student book**

This is the sixth of a set of eight Student Books (see Table 1). Together, they provide the information that is required to master SDMX, putting a particular emphasis on the data model.

This student book focuses on XML technology and the advantages of using it in SDMX. A brief introduction to XML is provided, followed by the presentation of the different ways in which XML is used at Eurostat. Detailed step-by-step exercises are provided, which explain the different specific aspects and applications of SDMX.

Finally, as Web Services are one of the main environments promoting SDMX, this publication presents how XML is used together with Web Services to provide a complete solution in data and metadata sharing methods.

| Ref. | Title |
|-------------|---|
| [01] | Introduction to SDMX |
| [02] | The SDMX Information Model |
| [03] | SDMX-ML Messages |
| [04] | Data Structure Definition |
| [05] | Metadata Structure Definitions |
| [06] | XML based technologies used in SDMX |
| [07] | SDMX architecture using the pull method for data sharing – Part 1 |
| [08] | SDMX architecture using the pull method for data sharing – Part 2 |

Table 1 – Students books on SDMX

Prerequisites

Reading the previous Student Books is strongly recommended.

2 XML, Schemas and Namespaces

2.1 Overview of XML

Extensible Mark-up Language (XML) is a mark-up language published by the W3C in 1998 that is used to describe the content and structure of data in a document. Like all W3C standards, it is an open standard, free for use. Unlike many other data formats, it is expressed in text, so that it can be used on any computer platform equally well.

One way to understand XML is to compare it with HTML. HTML - the Hypertext Markup Language - allows Web browsers to display documents with rich formatting, so that humans can see Web content effectively. XML is not designed for use directly by people, but is instead intended to deliver documents to computer applications over the Internet. As does HTML, XML uses tags to describe content. However, rather than focusing on content presentation, the tags used in XML describe the meaning and hierarchical structure of data. This functionality allows for the sophisticated data types that are required for efficient data interchange between different programs and systems. Further, as XML enables the separation of content and presentation, the content, or data, is portable across heterogeneous systems. Because it provides the facility of defining new tags, XML is also extensible.

XML documents take a string of characters - text and numbers - and organize it so that computers can process it in a far more structured way. XML documents can be treated as hierarchical 'trees', which allows them to be processed more efficiently than a simple string of characters could be. At the same time, for the purposes of transmission over the Internet, the XML documents still function as a simple string of characters, well-suited to exchange between applications. It is this aspect of XML - the creation of a 'structured' type of document - which makes it so powerful for computer applications.

XML syntax uses matching start and end tags to mark up information.

```
<Contact> and </Contact>;
```

Information delimited by tags is called an element.

```
<Contact>
```

Every XML document has a single root element, which is the top-level element that contains all the other elements. Elements that are contained by other elements are often referred to as sub-elements. An element can optionally have attributes. These are structured as name-value pairs which are part of the element and which are used to further define it.

```
<Receiver id="Eurostat">;
```

XML documents begin with a processing instruction: `<?XML ...?>`. This is the XML mark-up declaration [[W3C Section 2.9](#)]. While it is not mandatory, its presence does explicitly identify the document as an XML document and indicates the version of XML to which it was authored.

```
<?XML version="1.0"?>
```

Empty elements have a modified syntax. While most elements in a document are wrappers around some content, empty elements are simply markers of an occurrence. In the modified syntax, the trailing slash indicates to a program processing the XML document that the element is empty and that no matching end-tag should be sought.

```
<Telephone/>
```

An alternate syntax has been introduced for empty elements, which allows the end-tag to be present *if* it immediately follows the start-tag.

```
<Telephone></Telephone>
```

XML documents consist of mark-up and content. Six kinds of mark-up can be applied in an XML document: elements, entity references¹, comments, processing instructions², marked sections³ and document type declarations⁴.

Element: `<Contact>`

Entity References: `<`; meaning `<`

Comments: `<!--The header of the Structure -->`

Processing Instructions: `<?STATParser MessageBox("There is no data")?>`

Marked Sections: `<![CDATA[This <term>value</term> will not be parsed.]]>`

Document Type Declaration: `<!DOCTYPE NAME SYSTEM "file"[]>`

The following is an example of a simple XML document:

```
<?XML version="1.0"?>
<Receiver id="Eurostat">
  <Name xml:lang="en">Eurostat</Name>
  <Contact>
    <Name xml:lang="en">Francesco Rizzo</Name>
    <Department xml:lang="en">Statistics Division</Department>
    <Telephone/>
  </Contact>
</Receiver>
```

There is one element called ‘<Receiver>’ with one attribute ‘id’.

This element is composed but elements ‘<Name>’ and ‘<Contact>’. The value of the element ‘<Name>’ is Eurostat.

The element ‘<Contact>’ is composed by the elements ‘<Name>’ (which value is ‘Francesco Rizzo’), the element ‘<Department>’ (value ‘Statistics Division’), and ‘<Telephone>’ (with no value).

¹ Entity references provide ways to include information in XML documents by reference rather than by typing characters into the document directly. These entity references are created to avoid problems with parsing the XML document.

² Processing Instructions are information for the application. PI's allow documents to contain instructions for applications. They are not really of interest to the XML parser. Instead, the instructions are passed on to the application that uses the parser. The purpose of processing instructions is to represent special instructions for the application.

³ The portion of an XML document requiring special treatment.

⁴ A Document Type Declaration is information for the parser upon which the validity of XML documents is checked.

2.2 XML namespace

XML namespaces (xmlns) are used to create uniquely named *elements* and *attributes*, in order to avoid element name conflicts inside the XML file. This ensures reusability (reuse of elements and attributes), modularity (use of elements and attributes from other standard results) and extensibility (the incorporation of elements and attributes from other vocabularies).

In XML, namespaces are defined by a [W3C](#) recommendation called Namespaces⁵.

2.2.1 Basic concepts

An XML namespace is a sequence of characters that is identified by a URI⁶ reference. For example:

'http://www.SDMX.org/resources/SDMXXML/schemas/v2_0/registry', which is the namespace of the SDMX Registry, or:

http://www.SDMX.org/resources/SDMXXML/schemas/v2_0/structure, which is the namespace of the SDMX Structure. These are used in XML documents as element types and/or attribute names. The URI used does not necessarily reference a real file or URL.

Namespaces are identical if and only if the strings are identical, that is, if they consist of the same character sequence.

The URI in an XML document is associated to a prefix. That prefix is used with each element, to indicate which namespace the element belongs to.

2.2.2 Declaring namespaces

A namespace is declared by using a family of reserved XML *attributes* inside the root element of the XML document. Such *attributes*' names must either be **xmlns** or contain **xmlns:** as a prefix. In the same way as other XML *attributes*, these attributes may either be provided explicitly or by default. An *attribute's* value is a URI reference, the string identifying the namespace. The namespace name has the characteristic of being unique.

Example using 'xmlns' as attribute name:

Xmlns attribute name:

```
xmlns
```

Attribute value (URI):

```
http://www.SDMX.org/resources/SDMXXML/schemas/v2_0/message
```

Namespace declaration:

```
xmlns="http://www.SDMX.org/resources/SDMXXML/schemas/v2_0/message"
```

Example using "xmlns:" as prefix to another attribute name:

Attribute name using xmlns prefix:

⁵ See Namespaces in XML 1.0 (Second Edition): <http://www.w3.org/TR/2006/REC-xml-names-20060816/>

⁶ An URI (Uniform Resource Identifier) is the addressing technology for identifying resources on the Internet or private intranet. See: Annex 5.1 "URL, URI and URN".

```
xs
```

Attribute value (URI):

```
http://www.w3.org/2001/XMLSchema
```

Namespace declaration:

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

The namespace declaration applies to the element within which it is specified, and to all elements contained by that element, unless it is overridden by another namespace declaration having the same attribute name.

2.2.3 Applying namespaces

Applying namespaces to elements

The following is an example of a DSD where the elements match the prefix ‘structure’ declared (DSD for Demography Domain):

```
<Structure xmlns="http://www.SDMX.org/resources/SDMXML/schemas/v2_0/message"
xmlns:common="http://www.SDMX.org/resources/SDMXML/schemas/v2_0/common"
xmlns:structure="http://www.SDMX.org/resources/SDMXML/schemas/v2_0/structure"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.SDMX.org/resources/SDMXML/schemas/v2_0/message
http://katharma.agilis-sa.gr/SDMXML/SDMXMessage.xsd">
  <Header>
    <ID>DEMOGRAPHY_RAPID_V21</ID>
    <Test>>false</Test>
    <Name xml:lang="en">Demography Rapid Questionnaire DSD</Name>
    <Prepared>2008-05-14T12:00:00+02:00</Prepared>
    <Sender id="ESTAT">
      <Name xml:lang="en">Statistical Office of the European Communities</Name>
    </Sender>
  </Header>
  <CodeLists>
    <structure:CodeList id="CL_DEMO" agencyID="ESTAT" version="1.0" isFinal="true">
      <structure:Name xml:lang="en">Demography code list</structure:Name>
      <structure:Code value="PJANT">
        <structure:Description xml:lang="en">Population on 1 January of Year
X</structure:Description>
      </structure:Code>
      <structure:Code value="LBIRTHST">
        <structure:Description xml:lang="en">Births in Year
X</structure:Description>
      </structure:Code>
    </structure:CodeList>
  </CodeLists>
</Structure>
```



```

<structure:Code value="DEATHST">
  <structure:Description xml:lang="en">Deaths in Year
  X</structure:Description>
</structure:Code>

```

.... continue....

Applying namespaces to attributes

The following is an example in which elements and attributes match the prefix ‘sender’ and ‘info’:

```

<?xml version="1.0"?>
<sender:person xmlns:sender="http://person.com/sender" xmlns:info="http://www.sdmx.org/info" >
  <info:name sender:givenName="Francesco" sender:familyName="Rizzo" />
  <info:address sender:city="Luxembourg" />
</sender:person>

```

2.3 XML Schema (XSD)

One feature of XML is that it allows for the description of data structures so that they can be effectively validated. This aspect of XML is extremely useful when working with statistical data, because data validation is critical for applications which process statistics.

The way this works is important: a type of information is analyzed, so that its internal structure can be described. Questions such as ‘How is the data organized?’ or ‘What are the different types of data (numbers, text, codes, etc.)?’ need to be answered. Once the structure of the information is understood, an XML document can be defined for that information termed an ‘XML Document Type’. The document type can be described using an XML Schema, so that any application which receives an XML document of that type understands the rules about how it should be organized, and what type of data it should contain.

An XML schema defines the permitted building blocks of an XML document, itself written in XML.

SDMX makes use of the schema definition language known as XML Schema (XSD), which was published as a [W3C](#) recommendation in May 2001.

The characteristics of an XSD Schema are:

- XSD Schema is an XML document;
- XSD Schema supports Inheritance (one schema can inherit from another schema);

The following code states that the complex content inside the element `<xsd:complexContent>` stands to be inherit with the base name `FlowComplexType`.

```

<xs:complexType name="DataflowsType">
  <xsd:complexContent>
    <xsd:extension base="xsd:FlowComplexType">
      <xs:annotation>
      <xs:documentation>DataflowsType contains one or more
data flows.
      </xs:documentation>
      </xs:annotation>
      <xs:sequence>
        <xs:element name="Dataflow"

```

```

                                type="DataflowType"
maxOccurs="unbounded"/>
                                </xs:sequence>
                                </xsd:extension>
                                </xsd:complexContent>
</xs:complexType>

```

To call this last complexContent, the following example must be created:

```
<xsd:extension base="xsd:FlowComplexType ">
```

- XSD schema provides the ability to define new data types from a data type that is already defined inside the XSD Schema;
- XSD schema provides the ability of specifying data types for both elements and attributes.

2.3.1 Basic concepts

The root element of an XSD Schema (already defined as an XML document) is <schema>. The XSD namespace declaration is provided inside the <schema> element. The namespace declaration defines the XML file as being an XSD Schema.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

Element is the most important part of the schema. The kind of information that must be parsed from the XML document is specified at this point.

The following example states that the *element* ‘Obs’ can occur an unlimited number of times (maxOccurs) but must appear at least once (minOccurs):

```
<xsd:element name="Obs"
  minOccurs="1"
  maxOccurs="unbounded">
```

An *element* is limited by its type. For example, a simple-type element can contain text only, whereas an element of complex type can contain child elements and attributes. The diagram below provides an overview of the possible types of XML schema elements. Starting with the basic type *element*, two types can be defined: *simple* and *complex*. Each has different further possible characteristics as can be seen from the following diagram.

One Simple Type can be ‘atomic’ or ‘non-atomic’ if it is ‘user-derived’. And in the case of being ‘Built-in’, the possible values are ‘Primitive’ or ‘Derived’

In the case of a Complex Type, it can be ‘Empty’, ‘Simple Content’ or ‘Complex Content’. Only in the case of ‘Complex Content’ it can be a ‘Sequence’, ‘All’ or ‘Choice’.

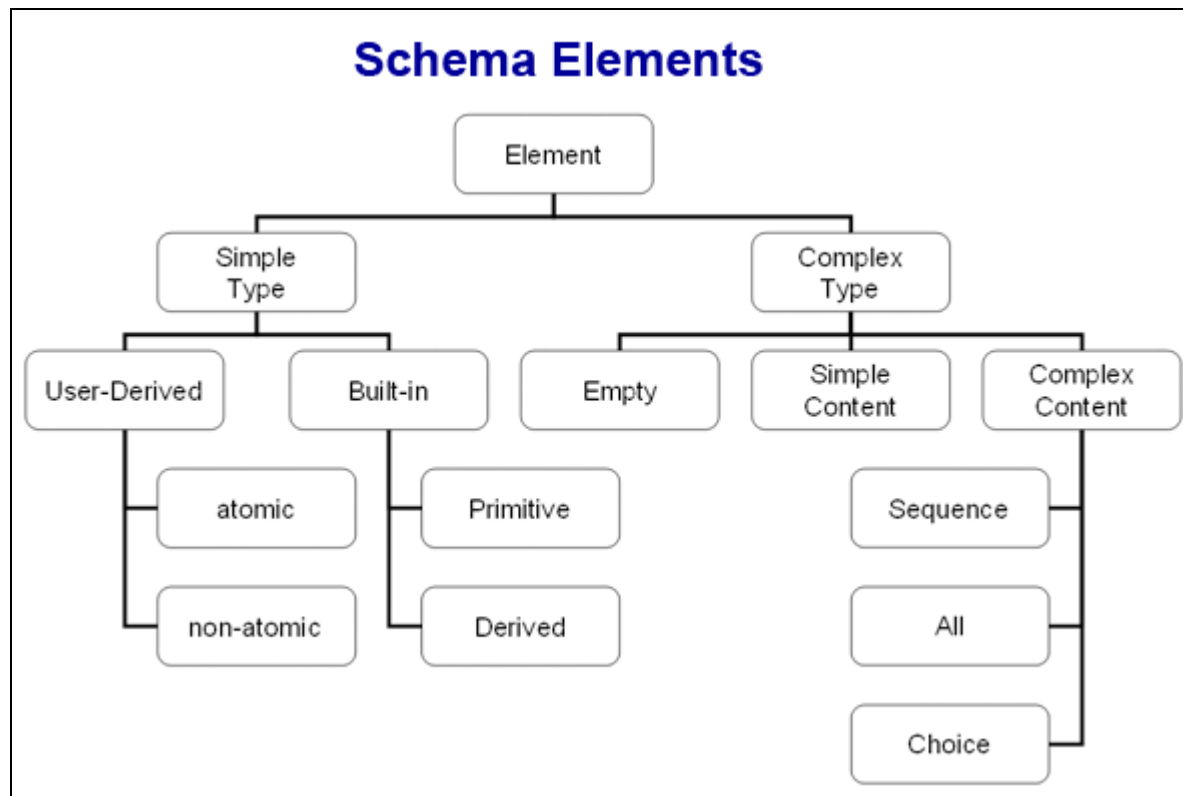


Figure 2 – Schema Elements

The Recommended SDMX schemas are available on the web site www.sdmx.org. This contains descriptions of each schema according to the SDMX File it is used to validate.

For example, inside a schema created to validate an SDMX-ML data file there are several specifications. One of them includes the element ‘POSTAL_CODEType’:

```

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

In this case, the specification of type, ‘POSTAL_CODEType’, is included with one restriction. This means that, if any element of this type is declared, the value of the data file must be string.

Another example can be given, which uses the type TimePeriodType. This type's description is given inside the element <xs:documentation>. Element ‘xs:union’ consists of a sum of other elements which must already be described in the schema (xs:dateTime xs:date xs:gYearMonth xs:gYear PeriodType):

```

<xs:simpleType name="TimePeriodType">
  <xs:annotation>
    <xs:documentation>TIME_PERIOD is not completely expressable in XML Schema's date type: instead we use the union of dateTime, date, gYearMonth, and gYear. The default name for the concept is TIME_PERIOD. Bi-annual, tri-annual, quarterly, and weekly periods have special formats (see PeriodType, above), but other periods would be described in terms of their beginning date or time (eg. a period of a decade is identified with a four-digit year corresponding to the decades' first year).</xs:documentation>
  </xs:annotation>
  <xs:union memberTypes="xs:dateTime xs:date xs:gYearMonth xs:gYear PeriodType"/>
</xs:simpleType>
  
```

In a further example, type is PeriodType. The type's description is found inside the element ‘xs:annotation’ . The one restriction is that of being an ‘xs:string’ type. Inside the element

‘xs:pattern’, one finds different patterns which may be used to display the value using this type:

```
<xs:simpleType name="PeriodType">
  <xs:annotation>
    <xs:documentation>PeriodType provides a list of tokens for specifying common periods: Quarterly: Q1, Q2, Q3, Q4; Weekly: W1 - W52; Triannual: T1, T2, T3; Biannual: B1, B2. These values appear after a four-digit year indicator, followed by a dash (ie. 2005-Q1).</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:pattern
      value="(\\d\\d\\d\\d\\-Q1\\d\\d\\d\\d\\-Q2\\d\\d\\d\\d\\-Q3\\d\\d\\d\\d\\-Q4\\d\\d\\d\\d\\-T1\\d\\d\\d\\d\\-T2\\d\\d\\d\\d\\-T3\\d\\d\\d\\d\\-B1\\d\\d\\d\\d\\-B2\\d\\d\\d\\d\\-W1\\d\\d\\d\\d\\-W2\\d\\d\\d\\d\\-W3\\d\\d\\d\\d\\-W4\\d\\d\\d\\d\\-W5\\d\\d\\d\\d\\-W6\\d\\d\\d\\d\\-W7\\d\\d\\d\\d\\-W8\\d\\d\\d\\d\\-W9\\d\\d\\d\\d\\-W10\\d\\d\\d\\d\\-W11\\d\\d\\d\\d\\-W12\\d\\d\\d\\d\\-W13\\d\\d\\d\\d\\-W14\\d\\d\\d\\d\\-W15\\d\\d\\d\\d\\-W16\\d\\d\\d\\d\\-W17\\d\\d\\d\\d\\-W18\\d\\d\\d\\d\\-W19\\d\\d\\d\\d\\-W20\\d\\d\\d\\d\\-W21\\d\\d\\d\\d\\-W22\\d\\d\\d\\d\\-W23\\d\\d\\d\\d\\-W24\\d\\d\\d\\d\\-W25\\d\\d\\d\\d\\-W26\\d\\d\\d\\d\\-W27\\d\\d\\d\\d\\-W28\\d\\d\\d\\d\\-W29\\d\\d\\d\\d\\-W30\\d\\d\\d\\d\\-W31\\d\\d\\d\\d\\-W32\\d\\d\\d\\d\\-W33\\d\\d\\d\\d\\-W34\\d\\d\\d\\d\\-W35\\d\\d\\d\\d\\-W36\\d\\d\\d\\d\\-W37\\d\\d\\d\\d\\-W38\\d\\d\\d\\d\\-W39\\d\\d\\d\\d\\-W40\\d\\d\\d\\d\\-W41\\d\\d\\d\\d\\-W42\\d\\d\\d\\d\\-W43\\d\\d\\d\\d\\-W44\\d\\d\\d\\d\\-W45\\d\\d\\d\\d\\-W46\\d\\d\\d\\d\\-W47\\d\\d\\d\\d\\-W48\\d\\d\\d\\d\\-W49\\d\\d\\d\\d\\-W50\\d\\d\\d\\d\\-W51\\d\\d\\d\\d\\-W52)"
    />
  </xs:restriction>
</xs:simpleType>
```

2.3.2 Heretical schemas

This feature is useful when a schema has a functionality that another schema wants to use.

To re-use another schema, an including sentence in the original schema must point to the secondary schema. A schema can include more than one schema. To do this, as many includes are needed as secondary schemas are used. The sentence should approach:

```
<xsd:include schemaLocation="structure.xsd"/>
```

Also an import sentence can be used, declaring the namespace of the schema called:

```
<xs:import namespace="http://www.SDMX.org/resources/SDMXXML/schemas/v2_0/common"
schemaLocation="SDMXCommon.xsd"/>
```

2.3.3 Declaring schemas

The association may be achieved via mark-up⁷ within the XML document itself, or via some external means such as programming code (Java, .Net...).

In order to validate an XML file against a schema, the ‘xsi:schemaLocation’ attribute is used inside the root element of the XML file. This attribute consists, in nodes, of two components: namespace and schema.

The following is an example of associating an XML document to three schemas via mark-up by using the attribute ‘xsi:schemaLocation’:

```
<CompactData xmlns="http://www.SDMX.org/resources/SDMXXML/schemas/v2_0/message"
xmlns:compact="http://www.SDMX.org/resources/SDMXXML/schemas/v2_0/compact"
xmlns:estat_sts="urn:sdmx:org.sdmx.infomodel.keyfamily.KeyFamily=ESTAT:STS:compact"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.SDMX.org/resources/SDMXXML/schemas/v2_0/message SDMXMessage.xsd
urn:sdmx:org.sdmx.infomodel.keyfamily.KeyFamily=ESTAT:STS:compact ESTAT_STS_Compact.xsd
http://www.SDMX.org/resources/SDMXXML/schemas/v2_0/compact SDMXCompactData.xsd">
```

⁷ See Chapter 2.5: “Basic Components of SDMX Files” for a complete example.

The following are examples of validating the XML file with programming code by using Java and .NET:

Java:

```
import java.io.IOException;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.Document;
import org.w3c.dom.Node;
import org.xml.sax.SAXException;
...
public XmlTester(String xmlFile) throws ParserConfigurationException, SAXException, IOException {
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    System.out.println("DocumentBuilderFactory: "+ factory.getClass().getName());

    factory.setNamespaceAware(true);
    factory.setValidating(true);
    factory.setAttribute("http://java.sun.com/xml/jaxp/properties/schemaLanguage",
"http://www.w3.org/2001/XMLSchema");

    // Specify our own schema - this overrides the schemaLocation in the xml file
    factory.setAttribute("http://java.sun.com/xml/jaxp/properties/schemaSource", "file:./test.xsd");

    DocumentBuilder builder = factory.newDocumentBuilder();
    builder.setErrorHandler( new SimpleErrorHandler() );

    Document document = builder.parse(xmlFile);
    Node rootNode = document.getFirstChild();
    System.out.println("Root node: "+ rootNode.getNodeName() );
}
```

.Net:

```
public void processSDMX(string sdmxQueryMEessage){
    SimpleXMLValidation(sdmxQueryMessage,
        System.Web.Hosting.HostingEnvironment.MapPath("~/SDMX-ML
        Schemas(SDMXMessage.xsd"))
}
```

2.4 XML in SDMX

SDMX is similar to many other open technology standards because it describes a number of types of information as XML documents. Statistical data is highly structured, and so XML is a natural tool for describing it.

The major benefit of having statistical data and metadata described and formatted as XML documents is that this information becomes easy for computer applications to process. There are many different technologies which use XML - the most important of these are the 'Web

Services' technologies, which assume that information passed back and forth between services is encoded in XML documents. All of the major tools used by software developers provide support for XML: Java, .NET, and virtually every other programming language provide excellent support for working with XML, and the major database platforms also support it.

Using XML allows for mainstream technology tools to work with the statistical data and metadata, making it easier and cheaper for statistical organizations to develop and maintain their applications.

The details of the specific benefits of XML in relation to SDMX are detailed in the following sections.

2.4.1 Structures

At the core of SDMX are the structural metadata artifacts described in the Information Model. These artifacts form the basis for all statistical data and reference metadata, and describe how these are structured. XML is used to represent these objects.

One feature of XML that lends itself well to the representation of structural metadata is the hierarchical nature of XML documents. Elements in XML are nested inside other elements, creating a tree like structure. Similar nesting is used in the Information Model. An example is an Item Scheme, which may contain child Items. These child Items only exist within the context of the Item Scheme. The hierarchical nature of XML allows this to be accurately represented, where the elements that describe Items are contained within the element that describes the Item Scheme.

Another feature of XML that is very useful in terms of describing the structural metadata in the SDMX Information Model is the manner in which the XML constructs are defined. XML has object-oriented features that allow the relationships in the Information Model to be represented in the XML structures. Although such details are not obvious to the end users, they become very powerful when using XML tools to process the structural metadata.

In summary, XML allows the structural metadata artifacts in the SDMX information model to be described in way the represents the intentions of the Information Model. The XML representation of structural metadata is predictable and simple to process. Further, because there are so many tools available that understand XML, the core functions of SDMX that are based on the structural metadata are much simpler to implement.

2.4.2 Data and Reference Metadata

Data in SDMX is typically oriented as time series. In the Information Model, a collection of Observation (each associated with a time value) or aggregated in a key, which identifies the phenomena being measured. The structure of the data (namely the key descriptors and the attributes associated with the data) is defined by a Key Family.

XML allows data to be structured both in this hierarchical group and according the Key Family which describes the format of the data. As a result of this structuring, processing data becomes simpler. XML also allows the data to be validated according to its underlying Key Family. This validation helps to provide some reliability to the data.

Similar to data, reference metadata is structured against a Metadata Structure Definition. Within a Metadata Structure Definition, Concepts can be organized into hierarchical Report Structures. XML allows this hierarchy to be expressed. Within the hierarchy of a report, the usage of a particular Concept can have a specific range of usages, where sometimes it is

optional and other times it is required. XML allows these requirements to be described in the underlying structure of the reference metadata XML.

2.5 Creating a schema from an SDMX-ML structure file

All SDMX-ML document types share common building rules at the message level ('SDMXMessage.xsd') as well as a set of common low-level components ('SDMXCommon.xsd'). Due to this, header information and basic structure will always be the same for all SDMX-ML documents, as all of them are validated against these two schemas. A list of schemas is also used in SDMX Standard, depending on the message format:

- Schema for describing all types of structural metadata – for data sets (Data Structure Definitions), for metadata sets (Metadata Structure Definitions), for related groups of metadata and data structures, and for all types of structural objects involved in registry-based exchanges ('SDMXStructure.xsd').
- Generic data schema for data-sharing exchange in Generic Format ('SDMXGenericData.xsd').
- Generic query schema for invoking web services ('SDMXQuery.xsd').
- Data Structure Definition-specific schema Compact format ('SDMXCompactData.xsd').
- Data Structure Definition-specific schema for Utility format ('SDMXUtilityData.xsd').
- Data Structure Definition-specific schema for cross-sectional format ('SDMXCrossSectionalData.xsd').
- Generic schema for registry interfaces ('SDMXRegistry.xsd').
- Generic schema for reference metadata sets ('SDMXRefMetadata.xsd').
- Metadata-structure-definition-specific schema for metadata sets.

In addition to these different schemas, standard mappings (XSLT Files) and corresponding transformation tools have been developed to create Data Structure Definition schemas (used to validate SDMX Data Files), as well as to transform XML data files from an XML data description format to another format (eg. from Generic to Compact, from Generic to Cross-Sectional, etc...).

Some SDMX-ML schemas are the same for all Data Structure Definitions and Metadata Structure Definitions. These include:

- SDMXMessage.xsd, for the generic description of the basic message structure common to all SDMX-ML messages;
- SDMXStructure.xsd, for the description of Data Structure Definitions, Metadata Structure Definitions, dataflows, metadataflows, codelists, concepts, structure sets, processes, hierarchical codelists, and reporting taxonomies;
- SDMXGenericData.xsd, for the description of data across Data Structure Definitions for generic processing;

- SDMXQuery.xsd, for the marking-up of queries against SDMX-conformant databases and web services;
- SDMXCommon.xsd, describing the common constructs used in other schemas;
- SDMXGenericMetadata.xsd, for the generic reporting of reference metadata;
- SDMXRegistry.xsd, for all interactions with the SDMX Registry Services.

Of these, only the SDMXStructure message and the SDMXGenericData message are required for general data exchange. For the generic exchange of reference metadata, only the SDMXStructure message and the SDMXGenericMetadata message are required. The documentation for each of these schemas is provided in the document ‘SDMX-ML: SCHEMA AND DOCUMENTATION (VERSION 2.0).’

Schemas are specific to Data Structure Definitions and Metadata Structure Definitions. Therefore, no single schema applies to all SDMX Messages. In consequence, although a single schema cannot be published, standard mappings (XSLT Files) are provided. The schemas can be gauged from an examination of SDMX Structure messages that describe the Data Structure Definition/Metadata Structure Definitions on which they are based. Free tools are planned, which aim to enable the creation of structure-specific schemas based on the Data Structure Definitions and Metadata Structure Definitions⁸.

These schemas are all as similar as possible. They vary according to where key values and attributes may be specified within the common structure. A difference that is less obvious is seen in the Utility and Metadata Report schemas, which are designed to carry as much structural metadata as possible in order to allow ‘typical’ XML tools⁹ (such as schema-guided editors and parsers) to benefit from the availability of this data. Such tools generally cannot consult the key family or metadata structure definition for structural metadata.

In the following example, one sees part of a Utility Schema generated from Demography Domain DSD. These are the rules generated for the ‘Obs’ element and ‘ObsType’ complexType:

```
...
<xs:element name="Obs" substitutionGroup="utility:Obs" type="ObsType"/>
  <xs:complexType name="ObsType">
    <xs:complexContent>
      <xs:extension base="utility:ObsType">
        <xs:sequence>
          <xs:element name="TIME" type="common:TimePeriodType"/>
          <xs:element name="OBS_VALUE" type="xs:double"/>
          <xs:element name="Annotations" type="common:AnnotationsType" minOccurs="0"/>
        </xs:sequence>
        <xs:attribute name="OBS_STATUS" type="CL_OBS_STATUS" use="optional"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

```

⁸ See Chapter 3: “XSL Transformations (XSLT)” for a complete example using XSLT Files process.

⁹ See Chapter 3: “XSL Transformations (XSLT)” for a complete example using XSLT Files process.


```

</xs:complexContent>
</xs:complexType>
...

```

Note that for all Data Structure Definition-specific and Metadata Structure Definition-specific schemas, namespaces must be constructed according to the rules for registry URN¹⁰ identifiers, as described in Section 5.2 of the SDMX Registry Interfaces specification, with the addition of a single field at the end of the URN:

- For Utility schemas: ‘xmlns:utility’ - the following is an example of schema ‘SDMXUtilityData.xsd’ using this identifier (only the beginning of the file):

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- Copyright SDMX 2004 - www.sdmx.org -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
xmlns="http://www.SDMX.org/resources/SDMXML/schemas/v2_0/utility"
xmlns:common="http://www.SDMX.org/resources/SDMXML/schemas/v2_0/common"
xmlns:utility="http://www.SDMX.org/resources/SDMXML/schemas/v2_0/utility"
targetNamespace="http://www.SDMX.org/resources/SDMXML/schemas/v2_0/utility">
...

```

- For Compact schemas: ‘xmlns:compact’ - the following is an example of schema ‘SDMXCompactData.xsd’ using this identifier (only the beginning of the file):

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- Copyright SDMX 2005 - www.sdmx.org -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
xmlns="http://www.SDMX.org/resources/SDMXML/schemas/v2_0/compact"
xmlns:compact="http://www.SDMX.org/resources/SDMXML/schemas/v2_0/compact"
xmlns:common="http://www.SDMX.org/resources/SDMXML/schemas/v2_0/common"
targetNamespace="http://www.SDMX.org/resources/SDMXML/schemas/v2_0/compact">

```

- For Cross-Sectional schemas: ‘xmlns:cross’ - the following is an example of schema ‘SDMXCrossSectionalData.xsd’ using this identifier (only the beginning of the file):

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- Copyright SDMX 2005 - www.sdmx.org -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
xmlns="http://www.SDMX.org/resources/SDMXML/schemas/v2_0/cross"
xmlns:cross="http://www.SDMX.org/resources/SDMXML/schemas/v2_0/cross"
xmlns:common="http://www.SDMX.org/resources/SDMXML/schemas/v2_0/common"
targetNamespace="http://www.SDMX.org/resources/SDMXML/schemas/v2_0/cross">

```

¹⁰ An URN (Uniform Resource Name) is an Internet resource with a name that, unlike a URL, has persistent significance - that is, the owner of the URN can expect that someone else (or a program) will always be able to find the resource. See Annex 5.1: "URL, URI and URN".

- For Metadata Report schemas: ‘:metadatareport’ - the following is the header of ‘MedatadaReport.xsd’ using this identifier (only the header):

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Copyright SDMX 2005 - www.sdmx.org -->
<xs:schema targetNamespace="http://www.SDMX.org/resources/SDMXML/schemas/v2_0/metadatareport"
xmlns="http://www.SDMX.org/resources/SDMXML/schemas/v2_0/metadatareport"
xmlns:common="http://www.SDMX.org/resources/SDMXML/schemas/v2_0/common"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
```

2.6 Basic components of SDMX Files

The following explains the meaning of each of the basic components of the SDMX Files.

An example of a Compact File, inside the root ‘CompactData’ element is:

```
<CompactData xmlns="http://www.SDMX.org/resources/SDMXML/schemas/v2_0/message"
xmlns:compact="http://www.SDMX.org/resources/SDMXML/schemas/v2_0/compact"
xmlns:estat_sts="urn:sdmx:org.sdmx.infomodel.keyfamily.KeyFamily=ESTAT:STS:compact"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.SDMX.org/resources/SDMXML/schemas/v2_0/message SDMXMessage.xsd
urn:sdmx:org.sdmx.infomodel.keyfamily.KeyFamily=ESTAT:STS:compact ESTAT_STS_Compact.xsd
http://www.SDMX.org/resources/SDMXML/schemas/v2_0/compact SDMXCompactData.xsd">
```

Elements xmlns (namespace)

Four different namespaces are declared, which will be used inside the file. The basic one, **xmlns**; the one corresponding to the data file's domain and DSD, **xmlns:estat_sts**; the one relating to the compact format, **xmlns:compact**; and the one relating to the schema instance, **xmlns:xsi**.

Elements Schema Locations

The following sentence is used to declare the schemas that are used to validate the example Compact File:

```
xsi:schemaLocation="http://www.SDMX.org/resources/SDMXML/schemas/v2_0/message SDMXMessage.xsd
urn:sdmx:org.sdmx.infomodel.keyfamily.KeyFamily=ESTAT:STS:compact ESTAT_STS_Compact.xsd
http://www.SDMX.org/resources/SDMXML/schemas/v2_0/compact SDMXCompactData.xsd"
```

It consists of nodes of two elements. The first one corresponds to the namespace, and the second to the schema location, with complete or relative path. There can be as many nodes as schemas one may wish to validate against. In this example there are three nodes:

First node:

“http://www.SDMX.org/resources/SDMXML/schemas/v2_0/message”
as namespace and the Schema Location ‘SDMXMessage.xsd’.

Second node: In this case the name space is a URN, that is a unique namespace declaration for this file; for this reason it begins with ‘urn’. “urn:sdmx:org.sdmx.infomodel.keyfamily.KeyFamily=ESTAT:STS:compact” and ‘ESTAT_STS_Compact.xsd’ as the Compact Schema used to validate the data file, created from the DSD. The schema ‘ESTAT_STS_Compact.xsd’ is the schema against which all the Compact Data Files need to be validated in order to know whether the data file's structure and content is correct. The schema can be generated via SDMX Registry, or by using Transformation Files, as explained in Chapter 3.

Third node: With the declaration of the namespace for compact format, “`http://www.SDMX.org/resources/SDMXXML/schemas/v2_0/compact`” and the relative path to the schema ‘`SDMXCompactData.xsd`’.

One Schema File Example:

This schema is a Compact Schema created from a DSD. This schema is used to validate any SDMX Data File in Compact format of the same domain that the DSD from which this schema was created. Below are the main elements of the root ‘`xs:schema`’ element:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:common="http://www.SDMX.org/resources/SDMXXML/schemas/v2_0/common"
xmlns:compact="http://www.SDMX.org/resources/SDMXXML/schemas/v2_0/compact"
xmlns="urn:estat:sdmx.infomodel.keyfamily.KeyFamily=ESTAT:MILK_TABLEA_M:2.1:compact"
xmlns:ns1="http://www.SDMX.org/resources/SDMXXML/schemas/v2_0/message"
targetNamespace="urn:estat:sdmx.infomodel.keyfamily.KeyFamily=ESTAT:MILK_TABLEA_M:2.1:compact"
elementFormDefault="qualified" attributeFormDefault="unqualified">

<xs:import namespace="http://www.SDMX.org/resources/SDMXXML/schemas/v2_0/compact"
schemaLocation="SDMXCompactData.xsd"/>

<xs:import namespace="http://www.SDMX.org/resources/SDMXXML/schemas/v2_0/common"
schemaLocation="SDMXCommon.xsd"/>

<xs:import namespace="http://www.SDMX.org/resources/SDMXXML/schemas/v2_0/message"
schemaLocation="SDMXMessage.xsd"/>
```

Elements xmlns (namespace)

Five namespaces are specified in this case. The namespace for the schemas, **xmlns:xs**; the common namespace, **xmlns:common**, which is relative to any SDMX schema; the namespace **xmlns:compact**, which is a compact schema; the basic namespace, ‘**xmlns**’; and, finally, the namespace called ‘**xmlns:ns1**’, which is the one that is used in the data file when the location of the schema is declared against which it is going to be validated.

Elements Schema Locations

As an SDMX Schema is also an XML file, in order to validate it, different import sentences are declared after the root element of the XML file, in order to obtain the reference to all the necessary schemas.

In this case:

```
<xs:import namespace="http://www.SDMX.org/resources/SDMXXML/schemas/v2_0/compact"
schemaLocation="SDMXCompactData.xsd"/>

<xs:import namespace="http://www.SDMX.org/resources/SDMXXML/schemas/v2_0/common"
schemaLocation="SDMXCommon.xsd"/>

<xs:import namespace="http://www.SDMX.org/resources/SDMXXML/schemas/v2_0/message"
schemaLocation="SDMXMessage.xsd"/>
```

There are three schemas in this XML file. Therefore, the SDMX schema used to validate SDMX Data Files is also validated against: ‘`SDMXCompactData.xsd`’, ‘`SDMXCommon.xsd`’ and ‘`SDMXMessage.xsd`’.

Other attributes and elements

Other specifications are included in this schema. The ‘`targetNamespace`’ is the namespace of the Data File. It is required at the time of validation. This means that the ‘`targetNamespace`’ must be the same as the one declared for the data file. In this example, the data file should include the following declaration according to the ‘`targetNamespace`’:

```
xmlns:mil="urn:estat:sdmx.infomodel.keyfamily.KeyFamily=ESTAT:MILK_TABLEA_M:2.1:compact"
```

Finally, the attributes ‘elementFormDefault’¹¹ and ‘attributeFormDefault’¹² should always have the same values as those seen here. These two values are used to indicate whether or not locally declared elements and attributes must be unqualified.

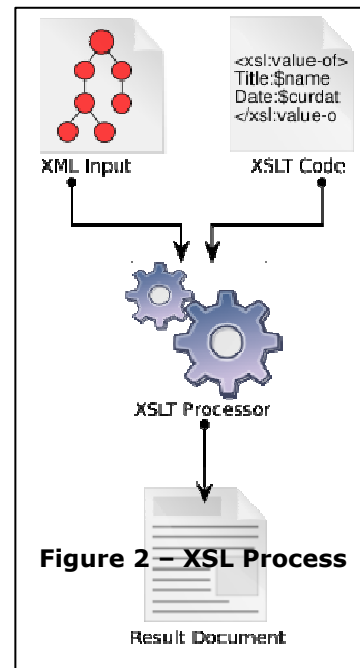
3 XSL Transformation (XSLT)

3.1 Introduction

XSLT is a XML-based language used for the transformation of XML documents into other XML or ‘human-readable’ documents (eg. HTML documents) describing the semantics of formatting information for different media. XSLT became a [W3C Recommendation](#) in November 1999.

The basic idea is that the original document is not changed. Rather, based on the content of the original document, a new one is created by using an XSLT document. As a language, XSLT is influenced by functional languages¹³, and by text-based pattern matching languages¹⁴.

The following is an example of the extraction of an XSLT file. It contains a conditional sentence, which depends on the element ‘codelist’.



```
<xsl:if test="not(@codelist)">
  <xsl:value-of select="@conceptRef"/>
  <xsl:text>Type</xsl:text>
</xsl:if>
```

The main element is the conditional element `<xsl:if>`¹⁵. This means that if the expression to be evaluated (test) does not have an attribute ‘codelist’ (@16), the output value will be

¹¹ Qualification of local elements and attributes can be globally specified by a pair of attributes, `elementFormDefault` and `attributeFormDefault`, on the schema element, or can be specified separately for each local declaration using the `form` attribute. All such attributes' values may each be set to unqualified or qualified, to indicate whether or not locally declared elements and attributes must be unqualified.

¹² See footnote 11.

¹³ Functional languages have a mathematical style. A functional program is constructed by applying functions to arguments.

¹⁴ Pattern matching is the act of checking for the presence of the constituents of a given pattern.

¹⁵ The `<xsl:if>` element is used to put a conditional test against the content of the XML file. The value of the required test attribute contains the expression which is to be evaluated.

¹⁶ @ in XSLT is used to refer to the attributes of the expression being processed.

extracted from the attribute ‘conceptRef’ (xsl:value-of select17), and with the literal text ‘Type’ (xsl:text18). (i.e. if the value of the attribute ‘conceptRef’ is ‘example’, the output is ‘exampleType’).

3.2 Step by Step: Transformation by Example

3.2.1 XML Copy Editor

Tool: XML Copy Editor 1.2.0.4

URL: <http://xml-copy-editor.sourceforge.net/>

The XML Copy Editor application provides an improve solution for the management of XML files within the W3D Standard. XML Copy Editor is free software released under [GNU General Public License](#). It can be used for three main different purposes:

- Visualization tool: XML structures can be visualized in hierarchical view.
- Validation tool: Validates an XML element against a provided Schema.
- Transformation tool: Transformation of an XML element using XSLT files.

The following examples make use of these main three characteristics of the application.

3.2.2 From an SDMX-ML structure file to an HTML presentation

Step 1: Open Structure XML File

Go to ‘File -> Open’ and select the Structure XML we are going to work with: (ESTAT+EUROSTAT_STS+1.0.xml)

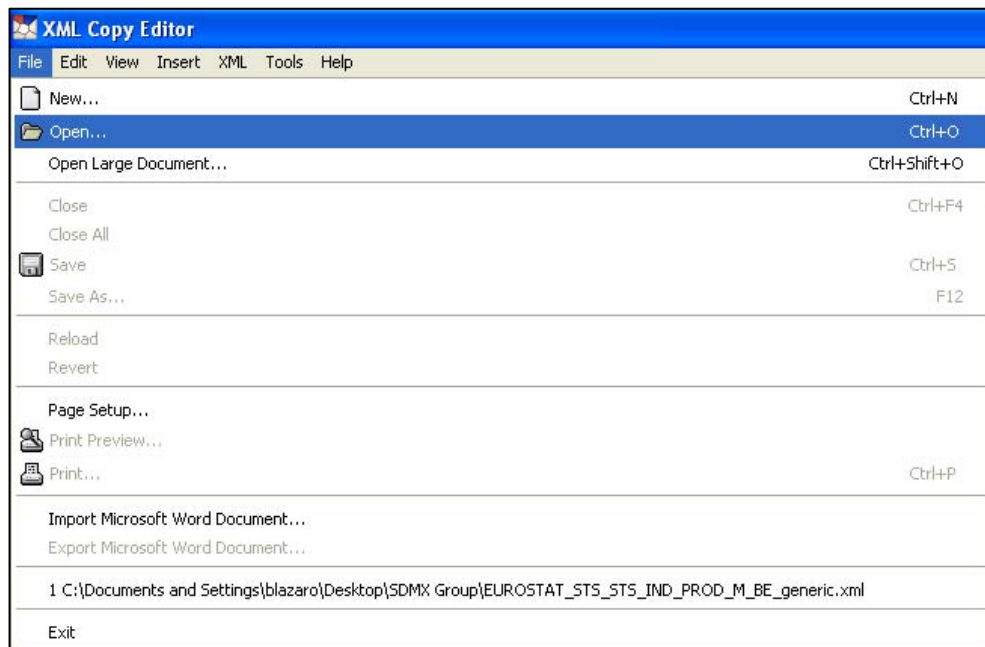


Figure 3 – Step 1. From an SDMX-ML structure file to an

¹⁷ The <xsl:value-of> element extracts the value of a selected node.

¹⁸ The <xsl:text> element is used to write literal text to the output.

HTML presentation

Step 2: Go to 'XML -> XSL Transform...' and select the XSL file we are going to work with: (StructureToHtml.xsl)

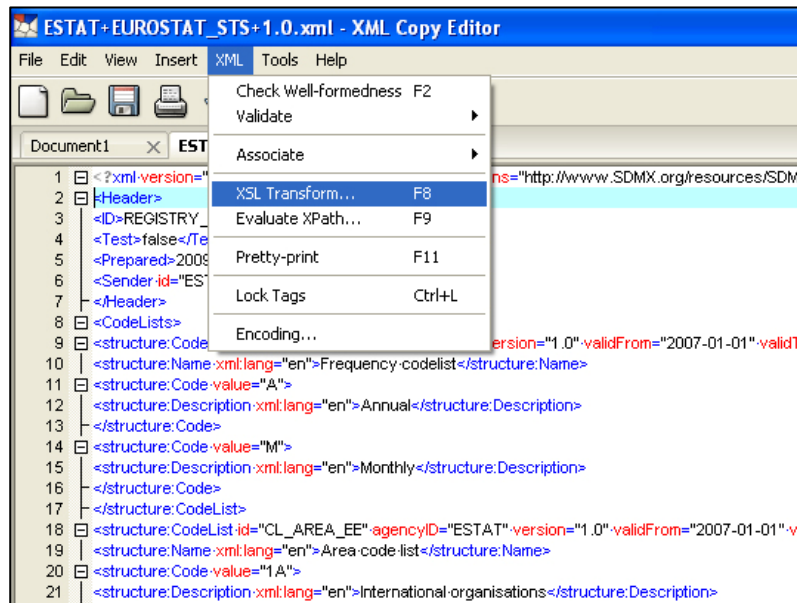


Figure 4 – Step 2. From an SDMX-ML structure file to an HTML presentation

A new file is generated by the XML Copy Editor tool with HTML code. A new HTML page has been created.

Step 3: We save this new Document as an HTML page. (Structure_In_HTML.html)

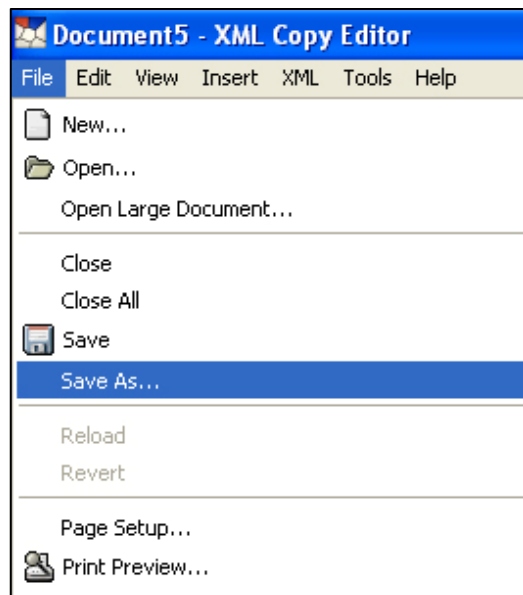


Figure 5 – Step 3. From an SDMX-ML structure file to an HTML presentation

Step 4: One can open this new HTML file in any HTML browser and see the results.

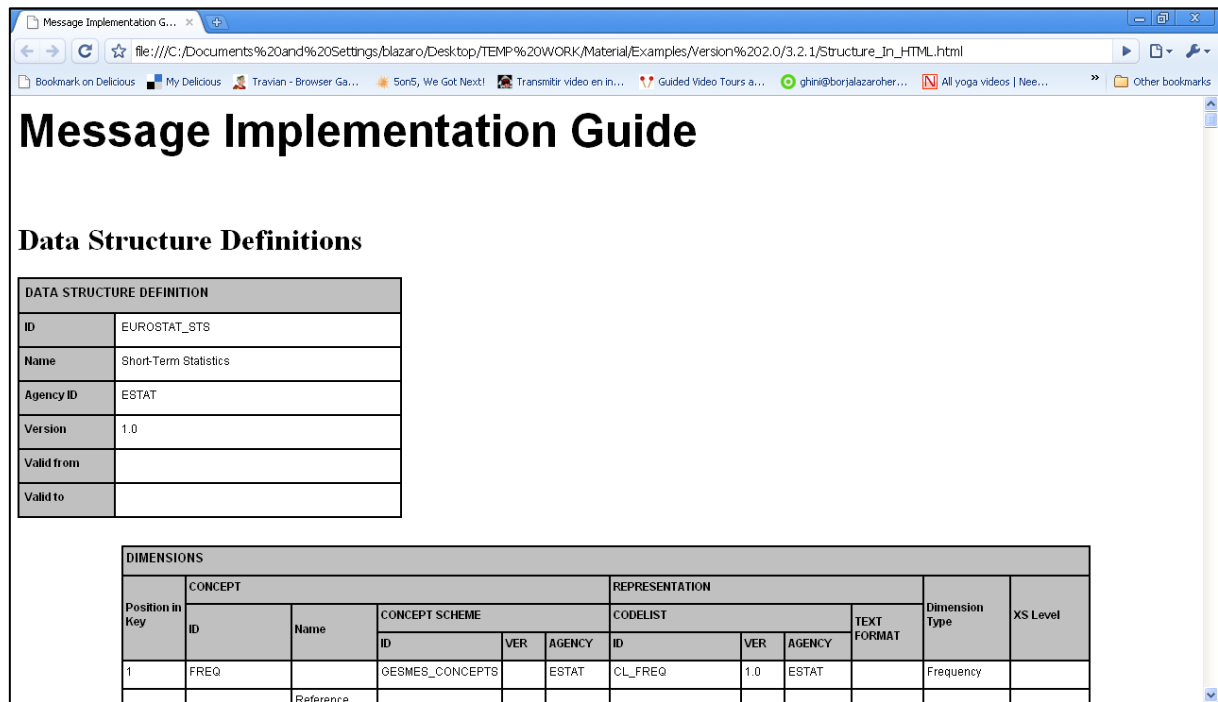


Figure 6 – Step 4. From an SDMX-ML structure file to an HTML presentation

3.2.3 From an SDMX-ML structure file to an XML schema

Tool: XML Copy Editor 1.2.0.4

URL: <http://xml-copy-editor.sourceforge.net/>

Step 1: Open Structure XML File

Go to 'File -> Open' and select the Structure XML we are going to work with: (ESTAT+EUROSTAT_STS+1.0.xml)

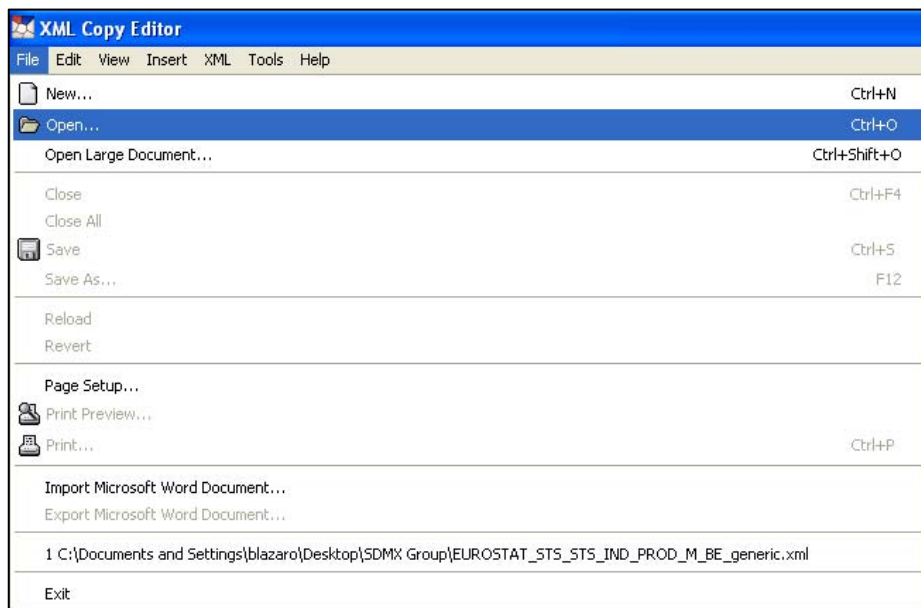


Figure 7 – Step 1. From an SDMX-ML structure file to an XML Schema

Step 2: Go to ‘XML -> XSL Transform...’ and select the XSL file we are going to work with: (StructureToCompact.xsl)

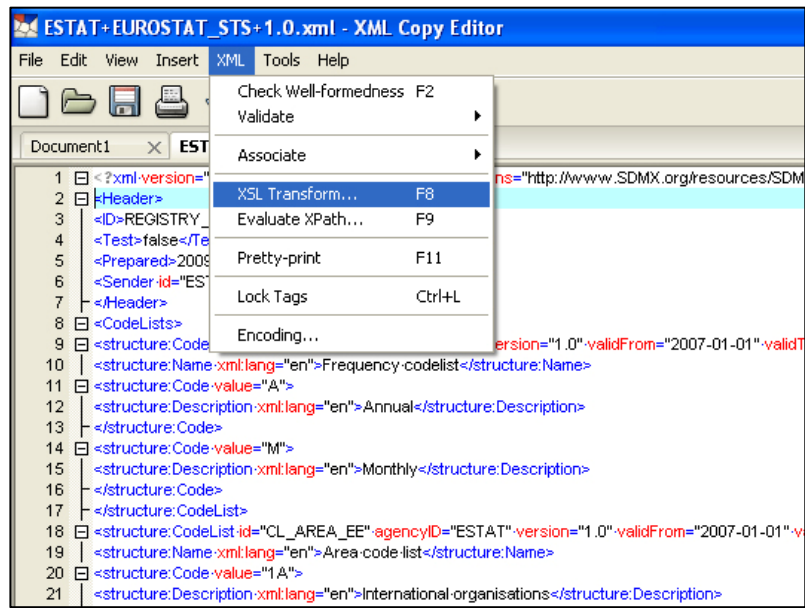


Figure 8 – Step 2. From an SDMX-ML structure file to an XML Schema

Step 3: A new file is generated by the XML Copy Editor tool with XML code. A new XML File has been created.

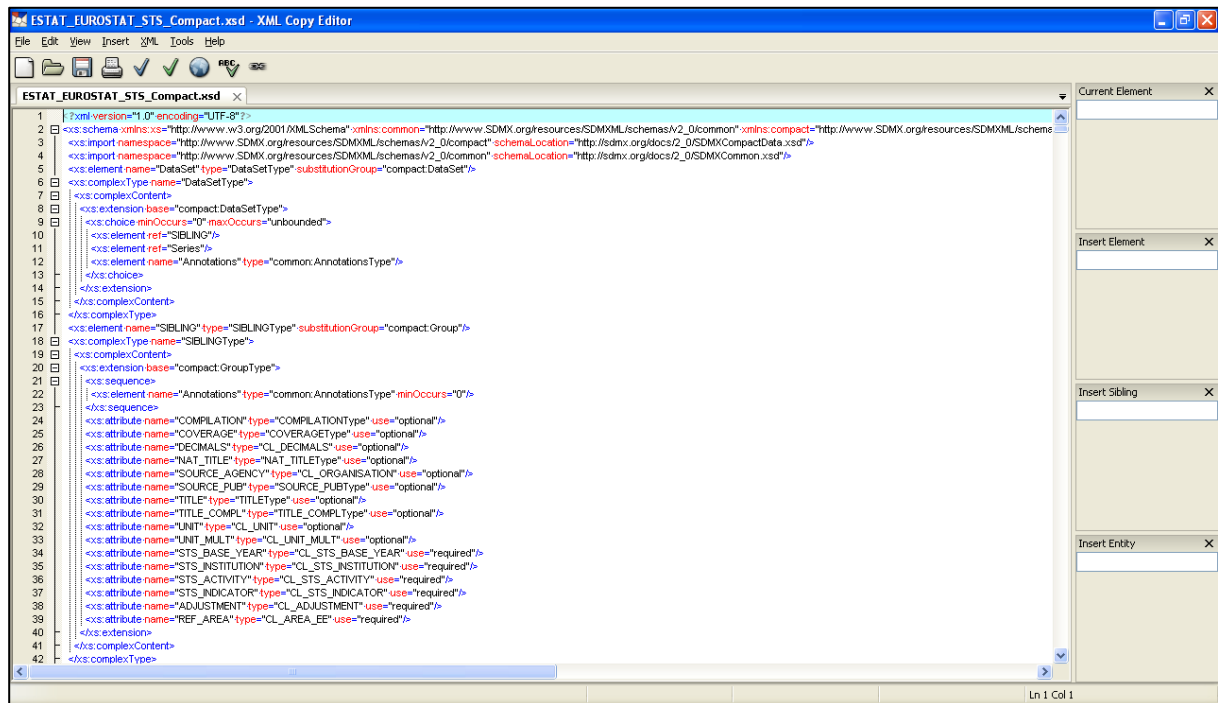


Figure 9 – Step 3. From an SDMX-ML structure file to an XML Schema

We save this new document as an XSD File (Schema).
(ESTAT_EUROSTAT_STS_Compact.xsd)

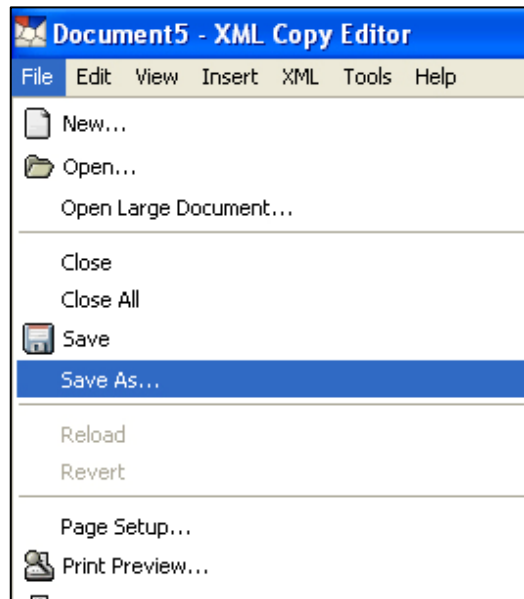


Figure 10 – Step 3. From an SDMX-ML structure file to an XML Schema

Step 4: One can use this Schema to validate XML Data Files.

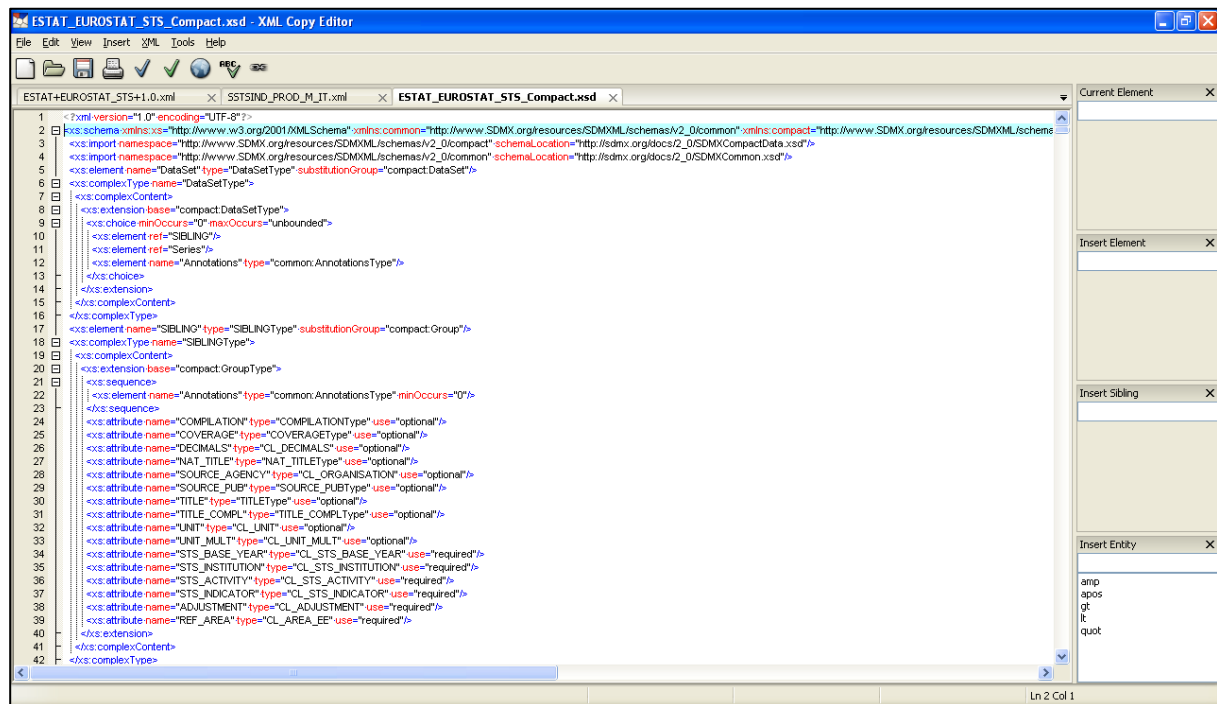


Figure 11 – Step 4. From an SDMX-ML structure file to an XML Schema

Open the XML Data File.

Go to 'File -> Open' and select the XML Data File we are going to work with:

(SSTSIND_PROD_M_IT.xml)

As one can see, some namespaces inside the XML Data File need to be taken into consideration:

```
xmlns:estat_sts="urn:estat:sdmx.infomodel.keyfamily.KeyFamily=ESTAT:EUROSTAT_STSTS:1.0:compact"
```

and

```
xsi:schemaLocation="http://www.SDMX.org/resources/SDMXXML/schemas/v2_0/message
xsd/SDMXMessage.xsd
urn:estat:sdmx.infomodel.keyfamily.KeyFamily=ESTAT:EUROSTAT_STSTS:1.0:compact
schema_to_validate.xsd"
```

Xmlns:estat_sts is the namespace which needs to be declared in the previously created schema (ESTAT_EUROSTAT_STSTS_Compact.xsd), in order to specify, in terms of validation, the namespace used in the data file.

In order to do this, in the schema 'ESTAT_EUROSTAT_STSTS_Compact.xsd', one needs to add the following namespace as an attribute inside the element 'xs:schema':

```
xmlns="urn:estat:sdmx.infomodel.keyfamily.KeyFamily=ESTAT:EUROSTAT_STSTS:1.0:compact"
```

In the XML data file, according to the 'xsi:schemaLocation' attribute, there are two schemas against which the XML data file is validated: 'SDMXMessage.xsd' and 'schema_to_validate.xsd'. SDMXMessage is an SDMX Schema which we are calling locally (because we have copied www.sdmx.org schemas locally in 'xsd' folder), and 'schema_to_validate.xsd', which should be the schema we already created beforehand. This means that we need to change 'schema_to_validate.xsd' to 'ESTAT_EUROSTAT_STSTS_Compact.xsd', the one we created, in order to tell the parser that we wish to validate the XML data file against our schema. When one checks the namespace that is being used in the XML data file for the 'schema_to_validate.xsd' file (already changed to 'SSTSIND_PROD_M_IT.xml'), one sees:

```
"urn:estat:sdmx.infomodel.keyfamily.KeyFamily=ESTAT:EUROSTAT_STSTS:1.0:compact".
```

This namespace also needs to be in the 'ESTAT_EUROSTAT_STSTS_Compact.xsd' schema, which is declared as targetNamespace inside the element 'xs:schema', in order to maintain the references, in the following way:

```
targetNamespace="urn:estat:sdmx.infomodel.keyfamily.KeyFamily=ESTAT:EUROSTAT_STSTS:1.0:compact"
```

This means, that the 'xs:schema' element of 'ESTAT_EUROSTAT_STSTS_Compact.xsd' should be:

```
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:common="http://www.SDMX.org/resources/SDMXXML/schemas/v2_0/common"
xmlns:compact="http://www.SDMX.org/resources/SDMXXML/schemas/v2_0/compact"
targetNamespace="urn:estat:sdmx.infomodel.keyfamily.KeyFamily=ESTAT:EUROSTAT_STSTS:1.0:compact"
```

```
xmlns="urn:estat:sdmx.infomodel.keyfamily.KeyFamily=ESTAT:EUROSTAT_STSTS:1.0:compact" elementFormDefault="qualified" attributeFormDefault="unqualified">
```

and the 'CompactData' element of 'SSTSIND_PROD_M_IT.xml' should be:

```
<CompactData xmlns="http://www.SDMX.org/resources/SDMXML/schemas/v2_0/message"
xmlns:common="http://www.SDMX.org/resources/SDMXML/schemas/v2_0/common"
xmlns:compact="http://www.SDMX.org/resources/SDMXML/schemas/v2_0/compact"
xmlns:cross="http://www.SDMX.org/resources/SDMXML/schemas/v2_0/cross"

xmlns:generic="http://www.SDMX.org/resources/SDMXML/schemas/v2_0/generic"

xmlns:query="http://www.SDMX.org/resources/SDMXML/schemas/v2_0/query"

xmlns:structure="http://www.SDMX.org/resources/SDMXML/schemas/v2_0/structure"

xmlns:utility="http://www.SDMX.org/resources/SDMXML/schemas/v2_0/utility"

xmlns:estat_sts="urn:estat:sdmx.infomodel.keyfamily.KeyFamily=ESTAT:EUROSTAT_STS
:1.0:compact"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.SDMX.org/resources/SDMXML/schemas/v2_0/message
xsd/SDMXMessage.xsd
urn:estat:sdmx.infomodel.keyfamily.KeyFamily=ESTAT:EUROSTAT_STS:1.0:compact
ESTAT_EUROSTAT_STS_Compact.xsd">
```

One can now validate the Data File.

The message should be: 'SSTSIND_PROD_M_IT.xml is valid'.

3.2.4 From an SDMX-ML data file to an HTML presentation

Tool: XML Copy Editor 1.2.0.4

URL: <http://xml-copy-editor.sourceforge.net/>

Step 1: Open the Compact Data XML File

Go to 'File -> Open' and select the Compact Data XML we are going to work with:

(SSTSIND_PROD_M_IT.xml)

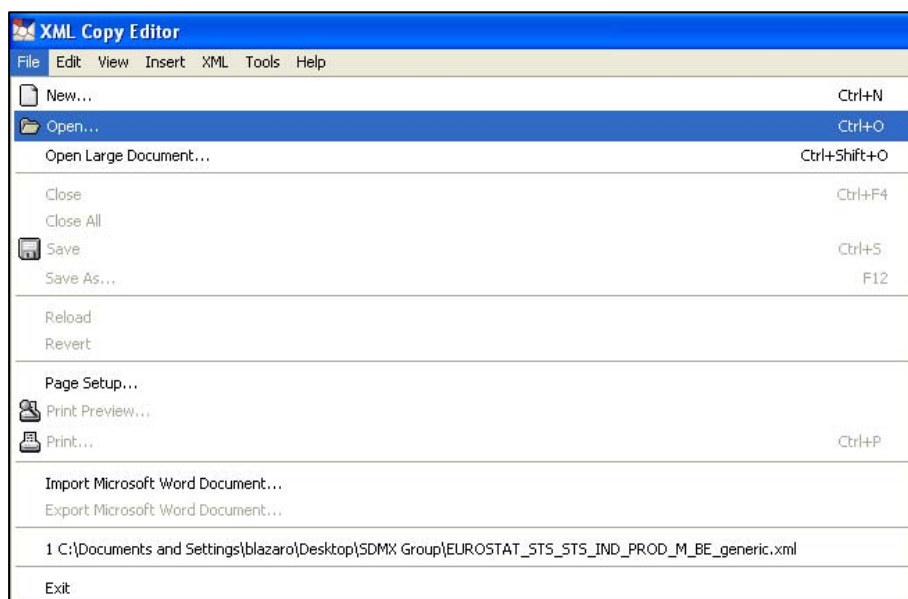


Figure 12 – Step 1. From an SDMX-ML data file to an HTML presentation

Step 2: One needs to change two values of the file ‘CompactData_to_HTML.xml’ before using it in the transformation process. The transformation file we are going to use should now be ready.

We add the values of the data file and the schema inside the ‘CompactData_to_HTML.xml’ file. INPUT_FILE is the data file that is to be processed and LOOKUP_FILE the schema we require, in this case ‘ESTAT_EUROSTAT_STS_Compact.xsd’.

The document should begin with the two following lines with ‘NO DATA’ elements:

```
<xsl:param name="INPUT_FILE">NO DATA</xsl:param>
<xsl:param name="LOOKUP_FILE">NO DATA</xsl:param>
```

One needs to add the following data and save the file:

```
<xsl:param name="INPUT_FILE">SSTSIND_PROD_M_IT.xml</xsl:param>
<xsl:param name="LOOKUP_FILE">ESTAT_EUROSTAT_STS_Compact.xsd</xsl:param>
```

It is important to store these two files (data file and schema) in the same folder as the ‘CompactData_to_HTML.xml’ file, as relative paths are being used to locate the documents. One otherwise needs to specify the absolute path to where the documents are stored.

```
<xsl:param name="INPUT_FILE">c:/files/SSTSIND_PROD_M_IT.xml</xsl:param>
<xsl:param
name="LOOKUP_FILE">c:/files/ESTAT_EUROSTAT_STS_Compact.xsd</xsl:param>
```

Step 3: Our data file has been loaded in XML Editor. We select the action ‘transform’ for this file, selecting the ‘xsl’ file (CompactData_to_HTML.xml) to perform this action.

Go to ‘XML -> XSL Transform...’ and select the XSL file we are going to work with: (CompactData_to_HTML.xml)

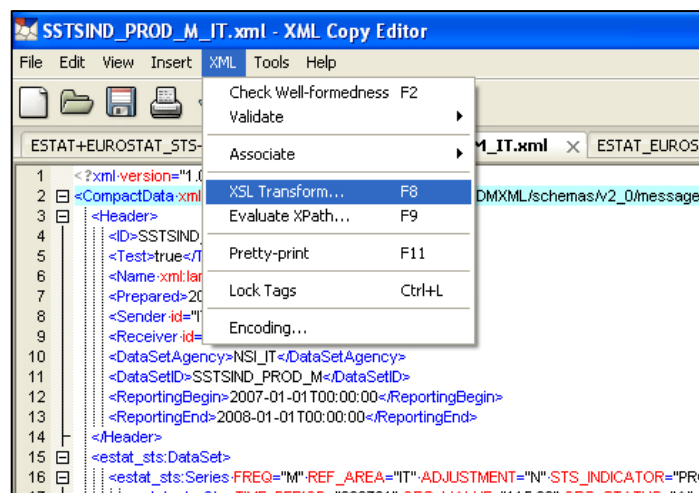


Figure 13 – Step 2. From an SDMX-ML data file to an HTML presentation

Step 3: A new file is generated by the XML Copy Editor tool with HTML code. A new HTML page has been created.

This new Document is saved as an HTML page.

(CompactData_In_HTML.html)

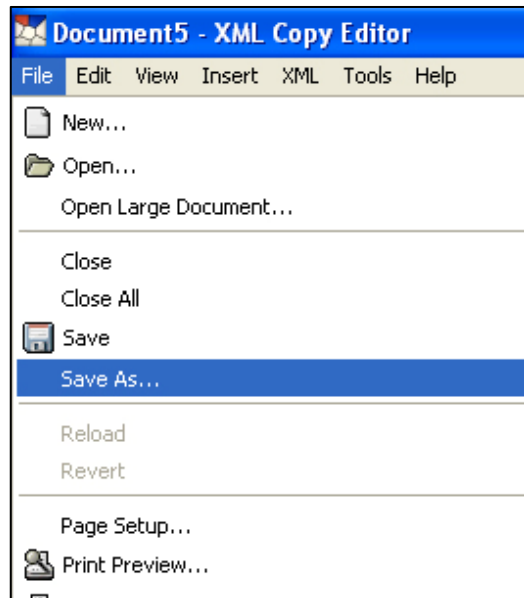


Figure 13 – Step3. From an SDMX-ML data file to an HTML presentation

Step 4: One can open the new HTML file in any HTML browser and see the results.

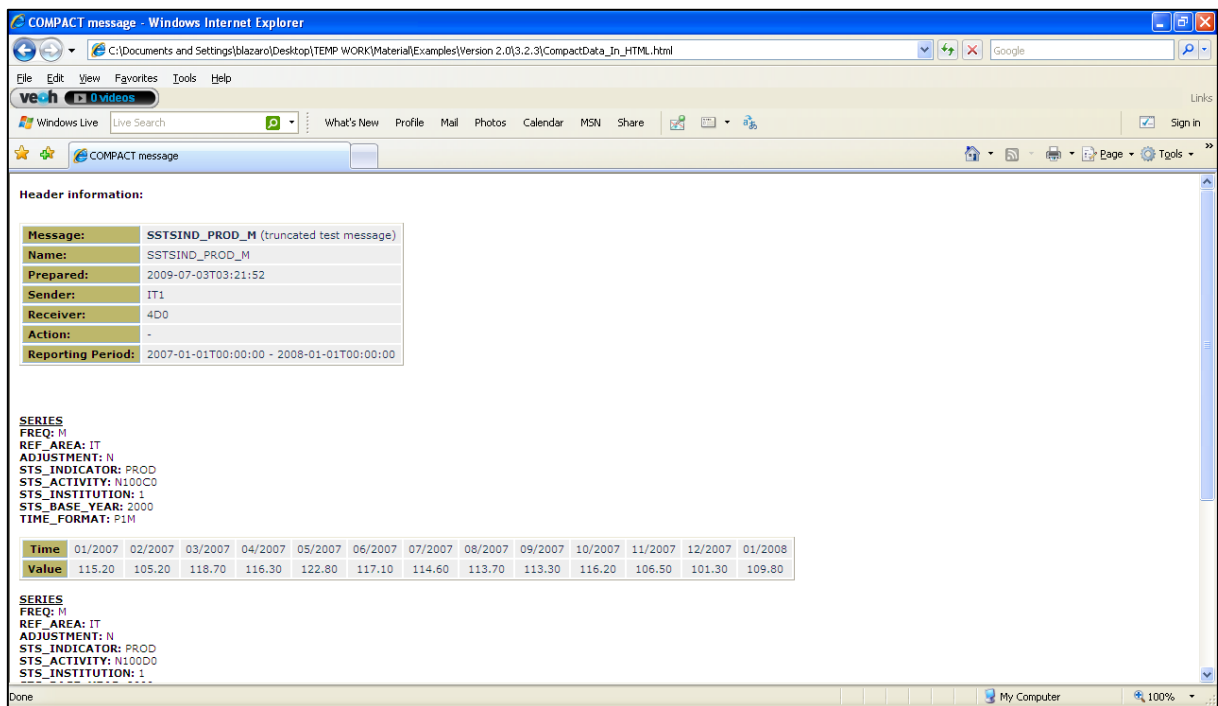


Figure 14 – Step 4. From an SDMX-ML data file to an HTML presentation

4 Web Services

4.1 Introduction

Web services represent the coming generation of Internet technologies. They enable computer applications to exchange data directly over the Internet, essentially allowing modular or distributed computing in a more flexible fashion than ever before.

In order to enable web services to function, however, many standards are required:

for requesting and supplying data;

for expressing the enveloping data which is used to package exchanged data;

for describing web services to one another, to allow for easy integration into applications that use other web services as data resources.

SDMX, with its focus on the exchange of data by using Internet technologies, will provide some of these standards in what concerns statistical data and metadata. Many web-service standards also already exist, and there is no need to re-invent them for specific use within the statistical community:

SOAP¹⁹ and the Web Services Description Language (WSDL) can be used by SDMX to complement the data and metadata exchange formats they standardise. Despite the promise of SOAP and WSDL, it has been found that various implementations by vendors were not, in fact, interoperable. It was for this reason that the Web Services - Interoperability (WS-I) initiative was started. This consists of a group of vendors who have all implemented the same web-service standards in the same way, and have verified this fact by carrying out interoperability tests. They publish profiles describing how to use interoperable web-service standards. The work of WS-I is considered to be appropriate to use with SDMX in order to meet the needs of the statistical community.

Representational State Transfer or REST basically means that each unique URL is a representation of some object. The HTTP methods such as GET and POST are the verbs that the developer can use to describe the necessary create, read, update, and delete actions to be performed. Some may see an analogy to operations in SQL, which also relies on a few common verbs. However, the REST style and HTTP protocol are mutually exclusive, and REST does not require HTTP.

Rest web services are not explained further in this Student Book. The recommendations for the use of this technology will be described in version 2.1 of the SDMX Standard Used Guidelines. The information provided in this Student Book is the one that stands for the version 2.0 of the SDMX Standard.

¹⁹ SOAP, originally defined as Simple Object Access Protocol, is a protocol specification for exchanging structured information in the implementation of Web Services in computer networks.

4.2 WSDL: Web Services Description Language

As it is important to be able to describe communications in a structured way, WSDL addresses this need by defining an XML grammar describing network services as collections of communication endpoints²⁰, which are capable of exchanging messages.

A WSDL document defines services as collections of network endpoints, or ports. In WSDL, the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings²¹. This allows the reuse of abstract definitions: messages, which are abstract descriptions of the data being exchanged, and port types which are abstract collections of operations. The concrete protocol and data format specifications for a particular port type constitute a reusable binding. A port is defined by associating a network address with a reusable binding, and a collection of ports define a service.

It is important to observe that WSDL does not introduce a new type of definition language. WSDL recognises the need for rich type systems describing message formats, and supports the XML Schemas specification (XSD) as its canonical type system. However, as it is unreasonable to expect a single type system grammar to be used to describe all message formats present and future, WSDL allows the use of other types of definition languages via extensibility.

The following is a complete WSDL File (CensusHub WSDL):

```
<?xml version="1.0" encoding="UTF-8"?>
<wSDL:definitions
  xmlns:census="http://localhost:8080/axis/services/CensusHubWebService"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:wSDL="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="CensusHub"
  targetNamespace="http://localhost:8080/axis/services/CensusHubWebService">
  <!-- Types -->
  <wSDL:types>
    <xsd:schema
      targetNamespace="http://localhost:8080/axis/services/CensusHubWebService">
targetNamespace="http://localhost:8080/axis/services/CensusHubWebService">
      <xsd:element name="QueryMessage" type="xsd:anyType" />
      <xsd:element name="CrossSectionalData" type="xsd:anyType" />
    </xsd:schema>
  </wSDL:types>

  <!-- Messages -->
  <wSDL:message name="GetCrossSectionalDataRequest">
    <wSDL:part element="census:QueryMessage" name="CrossSectionalDataRequest" />
  </wSDL:message>
  <wSDL:message name="GetCrossSectionalDataResponse">
    <wSDL:part element="census:CrossSectionalData" name="CrossSectionalDataResponse" />
  </wSDL:message>
</wSDL:definitions>
```

²⁰ An endpoint is the entry point to a service, a process, or a queue or topic destination.

²¹ Binding is the creation of a simple reference to something that is larger and more complicated and used frequently. The simple reference can be used instead of having to repeat the larger entity.

```

</wsdl:message>

<!-- Port -->
<wsdl:portType name="CensusHub">
  <wsdl:operation name="GetCrossSectionalData">
    <wsdl:input message="census:GetCrossSectionalDataRequest" />
    <wsdl:output message="census:GetCrossSectionalDataResponse" />
  </wsdl:operation>
</wsdl:portType>

<!-- Binding -->
<wsdl:binding name="CensusHubSOAP" type="census:CensusHub">
  <soap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http" />
  <wsdl:operation name="GetCrossSectionalData">
    <soap:operation
      soapAction="http://localhost:8080/axis/services/CensusHubWebService
        /GetCrossSectionalData" />
    <wsdl:input>
      <soap:body parts="CrossSectionalDataRequest" use="literal" />
    </wsdl:input>
    <wsdl:output>
      <soap:body parts="CrossSectionalDataResponse" use="literal" />
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>

<!-- Service -->
<wsdl:service name="CensusHub">
  <wsdl:port binding="census:CensusHubSOAP" name="CensusHubSOAP">
    <soap:address
      location="http://localhost:8080/axis/services/CensusHubWebService" />
  </wsdl:port>
</wsdl:service>
</wsdl:definitions>

```

In addition, WSDL defines a common binding mechanism. This is used to attach a specific protocol or data format or structure to an abstract message, operation, or endpoint. It allows the reuse of abstract definitions.

4.3 SDMX recommendations for web services using SOAP technology

All SDMX web services should be described using WSDL instances, in order to specify the aspects of the multiple-message exchange they support.

The global element for each XML data and metadata format within SDMX should be specified as the content of the replies to each exchange. The function names for each identified pattern are specified below, along with the type of SDMX ML payload.

The SDMX web service is composed of a set of data exchanges. Thus, the SDMX web service implements a ‘multiple-message exchange pattern’ (WSDL terminology). These exchanges are enumerated below, along with an indication of whether the SDMX web service is required to support them:

- Obtain Key Family
- Obtain Codelists
- Obtain Concepts
- Obtain Metadata Structure Definition
- Obtain Generic Data
- Obtain Compact Data
- Obtain Utility Data
- Obtain Cross-Sectional Data
- Obtain Reference Metadata
- Obtain Metadata Report
- Obtain Hierarchical Codelist
- Obtain Structure Set
- Obtain Reporting Taxonomy
- Obtain Process

In order to ensure interoperability between SDMX web services, compliance with sections of the WS-I Profile 1.1 is recommended for all SDMX web services. The documentation can be found at:

<http://www.ws-i.org/Profiles/BasicProfile-1.1-2004-08-24.html>.

The recommended sections are those concerning the use of SOAP and WSDL. UDDI, while useful for advertising the existence of SDMX web services, is not necessarily central to SDMX interoperability.

Due to the fact that some queries may produce very large numbers of data points or large amounts of reference metadata as a response, it is recommended that an SDMX web service support the use of the ‘DefaultLimit’ field in the SDMXQuery message. If a response is larger than the suggested default limit in the query, then the response should be truncated. A truncated response is a partial response, but it must still be a valid SDMX-ML document. The fact that it is truncated should be indicated by the ‘Truncated’ field in the Header element of the response message.

Note that the default limit is to be interpreted as an order-of-magnitude suggestion and not as a literal limit – it is not always easy to predict exactly what the effects of a truncation will be when the web service still must produce a valid SDMX-ML instance.

It is the responsibility of the querying service to adjust the query and to re-send it, to produce a non-truncated response, if that is what is needed.

4.4 SOAP messages for SDMX Query

A SOAP message is an XML document containing:

- an Envelope element to identify the XML document as a SOAP message,
- a Header element for the header information,
- a Body element containing call and response information,
- a Fault element to manage errors and status information.

4.4.1 Envelope, Header and Body

The SOAP Envelope tag ‘<SOAP-ENV:Envelope ... >’ in the request message specifies that the SOAP message's encoding style follows the schema defined at <http://schemas.xmlsoap.org/soap/encoding/>. The Envelope can contain an optional header and a mandatory body.

SOAP Headers are optional. They are used to transmit authentication or session management data. Authentication and session management are out of the scope of the SOAP protocol but this was designed to allow flexibility in SOAP.

The SOAP Body tag ‘<SOAP-ENV:Body>’ encapsulates a single method tag porting the name of the method itself.

The following is an example of a SOAP message:

```
<SOAP-ENV:Envelope>
  <SOAP-ENV:Body>
    <GetCrossSectionalData>
      <Query> <- MS .Net implementation only
        <QueryMessage>
          ...
        </QueryMessage>
      </Query> <- MS .Net implementation only
    </GetCrossSectionalData>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The root element is the Envelope, which in this case does not contain the optional Header. But it does contain the mandatory Body. The Body contains the method call ‘GetCrossSectionalData’.

The operation only has one parameter, the SDMX QueryMessage element. The content of the QueryMessage element corresponds to the SDMX specification.

<Query> parameter can be added, which includes the QueryMessage. This is configurable per web service, depending on the technology used (.NET). The SDMX QueryMessage contains a Header and the Query.

```
<QueryMessage>
  <Header>
    -- Header Information --
  </Header>
  <Query>
    -- Query Information --
  </Query>
</QueryMessage>
```

The header section specifies the hypercube selected (in this case CENSUSHUB_Q_XS1) and related data.

```
<Header>
  <ID>CENSUSHUB_Q_XS1</ID>
  <Test>true</Test>
  <Name xml:lang="en">CENSUSHUB_Q_XS1</Name>
  <Prepared>2008-07-14T10:37:09</Prepared>
  <Sender id="4D0"/>
  <Receiver id="IT1"/>
</Header>
```

4.4.2 Fault

The SOAP Fault element is used to transmit error messages.

A Fault element must appear as the child element of a Body element. A Fault element can only appear once in a SOAP message.

The SOAP Fault element is made up of the following elements:

- Faultcode <faultcode>: to identify the fault.
- Faultstring <faultstring>: to describe the fault.
- Faultfactor <faultfactor>: description of whom or what caused the fault.
- Detail <detail>: specific error information related to the ‘Body’ element.

The following example depicts full SOAP envelopes including SOAP faults:

```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <soap:Fault>
      <faultcode>soap:Client</faultcode>
      <faultstring>not compulsory message</faultstring>
      <faultfactor>GetCompactData</faultfactor>
      <detail>
        <Error xmlns="http://sodi.istat.it/sdmxWS">
          <ErrorNumber>2000</ErrorNumber>
          <ErrorMessage>
            Error due to a non correct client message
          </ErrorMessage>
          <ErrorSource>GetCompactData</ErrorSource>
        </Error>
      </detail>
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

```
</Error>  
</detail>  
</soap:Fault>  
</soap:Body>  
</soap:Envelope>
```

5 Annex

5.1 *URL, URI and URN*

URI (Uniform Resource Identifier)

A URI indicates the name and the address of the resource on the web. The URI is subdivided into URL and URN.

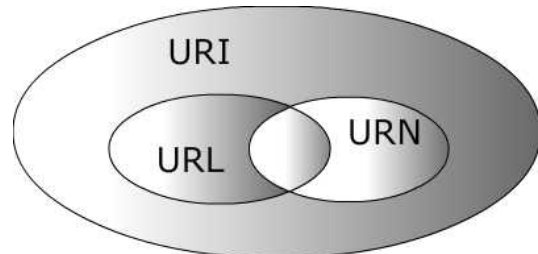


Figure 15 – URL, URI and URN

URL (Uniform Resource Locator)

A URL is the address of a resource on the web. A URL defines how the resource can be obtained and from where it can be obtained.

URN (Uniform Resource Name)

A URN is the name of a resource on the web. A URN does not imply the availability of the identified resource and never informs about how the resource can be obtained.

URI = URL + URN

If the string is 'This is SDMX', it is a URN, because it describes something's name but not its location.

eg.: ISBN: 1-930110-59-6

If the string is 'SDMX is managed in Luxembourg', it is a URL, as it describes a location.

eg.: <http://www.sdmx.org/files/report.html>

And if the string is 'This is SDMX and it is managed in Luxembourg', then it is a proper URI.

eg.: [files/report.html](#)

6 Glossary

Table 3 presents the list of concepts and acronyms with their definition.

| Concept | Definition |
|------------------|---|
| <i>DSD</i> | Data Structure Definition |
| <i>EDIFACT</i> | Electronic Data Interchange for Administration, Commerce and Transport |
| <i>ESMS</i> | Euro SDMX Metadata Structure |
| <i>GESMES/TS</i> | GESMES Time Series data exchange message |
| <i>HTML</i> | Hypertext Markup Language |
| <i>ISO</i> | International Organisation for Standardisation |
| <i>IT</i> | Information Technology |
| <i>MSD</i> | Metadata Structure Definition |
| <i>REST</i> | Representational State Transfer |
| <i>RSS</i> | Really Simple Syndication (<i>also used Rich Site Summary</i>) - family of Web feed formats to publish frequently updated information |
| <i>SDMX</i> | Statistical Data and Metadata eXchange. |
| <i>SDMX-EDI</i> | SDMX Electronic Data Interchange - EDIFACT format for exchange of SDMX-structured data and metadata |
| <i>SDMX-IM</i> | SDMX Information Model |
| <i>SDMX-ML</i> | SDMX Markup Language - XML format for the exchange of SDMX-structured data and metadata |
| <i>SOAP</i> | Simple Object Access Protocol |
| <i>SQL</i> | Sequence Query Language |
| <i>UDDI</i> | Universal Description, Discovery and Integration |
| <i>UML</i> | Unified Modelling Language |
| <i>URI</i> | Uniform Resource Identifier |
| <i>URL</i> | Uniform Resource Locator |
| <i>URN</i> | Uniform Resource Name |
| <i>W3C</i> | World Wide Web Consortium |
| <i>WSDL</i> | Web Services Description Language |
| <i>XML</i> | EXtensible Markup Language |
| <i>XSD</i> | XML Schema |
| <i>XSL</i> | Extensible Stylesheet Language |
| <i>XSLT</i> | XSL Transformations |

Table 3 – Glossary