Head First SDMX-ML

Using SDMX-ML to publish the euro foreign exchange reference rates on the ECB website

European Central Bank

2006-11-28

Abstract

The aim of this tutorial is to introduce you to SDMX-ML, using a "real world" task, the publication of the euro foreign exchange reference rates in SDMX-ML.

Table of Contents

Objectives and prerequisites

The aim of this tutorial is to introduce you to SDMX-ML, using a "real world" task, the publication of the euro foreign exchange reference rates [http://www.ecb.int/stats/exchange/eurofxref/html/index.en.html] in SDMX-ML.

The tutorial will look at the technology from both sides of the fence: the side of the data provider (how can we use SDMX-ML to publish statistical data on our website?) and the side of the data consumer (what kind of useful things can we do with an SDMX-ML data file?). Before doing this however, we will present, in a nutshell, the SDMX information model and some of the SDMX-ML formats.

To make the most out of this tutorial, basic knowledge of XML and XML-related technologies (such as XML Schemas, XSLT and SAX) is expected. Some of the tasks described in the tutorial will also require the use an XML validating parser (for instance Apache Xerces [http://xerces.apache.org/] or xmllint from libxml2 [http://xmlsoft.org/]) and an XSLT processor (such as Apache Xalan [http://xalan.apache.org/], Saxon [http://saxon.sourceforge.net/] or xsltproc from libxml2).

Most of the exercises can be run using the supplied build file, for those who use Ant [http://ant.apache.org/] as their build system.

A brief overview of SDMX

The Statistical Data and Metadata Exchange initiative [http://www.sdmx.org/] is sponsored by 7 in-

stitutions (BIS, ECB, Eurostat, IMF, OECD, UN, World Bank) to foster standards for the exchange of statistical information. The version 1 of the standard is an ISO standard (ISO/Technicial Specification 17369:2005). It offers an information model to represent statistical data and metadata, as well as several formats to represent the model (SDMX-EDI and several SDMX-ML formats). It also proposes a standard way of implementing web services, including the use of registries.

The SDMX information model in a nutshell

The list below describes the minimal knowledge needed about the SDMX information model [1] so that we can start developing an application based on the SDMX standard:

1. **Descriptor concepts:** In order to make sense of some statistical data, we need to know the concepts associated to it (for example, the figure 1.2953 alone is pretty meaningless, but if we know that this is an exchange rate for the US dollar against the euro on the 23 November 2006, it starts to make more sense).

2. **Packaging structure:** Statistical data can be grouped together. The following levels are defined: the *observation level* (the measurement of some phenomenon), the *series level* (the measurement over time of some phenomenon, usually following a regular interval), the *group level* (group of series. A well-known example is the sibling group which contains a set of series which are identical except that they are measured with different frequencies) and the *data set level* (made up of several groups, for instance to cover a specific statistical domain). The descriptor concepts mentioned in point 1 can be attached at various levels in this hierarchy.

3. **Dimensions and attributes:** There are two types of descriptor concepts: the ones which both identify and describe the data are called *dimensions*, and those which are purely descriptive are called *attributes*.

4. **Keys:** Dimensions are grouped into *keys*, which allow the identification of a particular set of data (for example, a series). The key values are attached at the series level, and are given in a fixed sequence. By convention, frequency is the first descriptor concept, and the other concepts are assigned an order for that particular data set. Partial keys can be attached to groups.

5. **Code lists:** Each possible value for a dimension is defined in a *code list*. Each value on that list is given a language-independent abbreviation (a *code*) and a language-specific description. Attributes are sometimes represented with codes, but sometimes represented by free-text values. This is fine as the purpose of an attribute is solely to describe and not to identify the data.

6. **Data Structure Definitions:** A *Data Structure Definition* (key family) specifies a set of *concepts* which describe and identify a set of data. It tells which concepts are *dimensions* (identification and description), and which are *attributes* (just description), and it gives the *attachment level* for each of these concepts, based on the packaging structure (*Data Set*, *Group*, *Series*, *Observation*) as well as their status (mandatory versus conditional). It also specifies which *code lists* provide possible values for the dimensions, as well as the possible values for the attributes, either as code lists or free text fields.

The various SDMX-ML formats

SDMX-ML supports various use cases and therefore defines several XML formats [2]. For the scope of this tutorial, the two following formats will be used:

1. **The Structure Definition format** . This format will be used to define the structure (concepts, code lists, dimensions, attributes, etc) of the key families.

[1] For a detailed description of the SDMX information model, see section 02 of the SDMX Standards Version 2.0 Complete Package [http://www.sdmx.org/standards/standards_package_2_0.aspx]. A very useful introduction to the basic concepts is available at the end of this document.
[2] For a detailed description of SDMX-ML, see section 03 of the SDMX Standards Version 2.0 Complete Package

2. **The Compact format**. This format will be used to define the data file. It is not a generic format (it is specific to a Data Structure Definition), but it is designed to support validation and is much more compact so as to support the exchange of large datasets.

Now that we know the basics, we can start developing our application.

Data provider: Publishing the euro foreign exchange reference rates in SDMX-ML

We want to publish the euro foreign exchange reference rates data on our website. The first step is to analyze the kind of data we are dealing with, and then to create the Structure Definition file to represent this data. We will then generate a schema out of the Structure Definition file, which we will use to validate the data file. Finally, we will create the XML data file to be published on the website.

The Data Structure Definition

For the purpose of this exercise, the Data Structure Definition defined in the table below will be used [3].

Table 1. The Data Structure Definition: dimensions, measures and attributes

Dimensions			
Type	Concept	Representation	Description
Dimension 1 (role is frequency)	FREQ	CL_FREQ	Interval of time between observations (daily in this case).
Dimension 2	CURRENCY	CL_CURRENCY	The currency whose value is being measured against the base currency (for instance, US dollar).
Dimension 3	CURRENCY_DENOM	CL_CURRENCY	The base currency (the euro in this case).
Dimension 4	EXR_TYPE	CL_EXR_TYPE	The exchange rate type (spot in this case).
Dimension 5	EXR_SUFFIX	CL_EXR_SUFFIX	Exchange rate series variation (Average or standardised measure for a given frequency in this case).
Dimension (role is time)	TIME_PERIOD	Time Point Set	The date at which an observation was made.

[3] It is similar to the official ECB_EXR1 key family but has been slightly simplified for the scope of this exercise.

			It is not part of the series key but is attached to the observation level.
Measure			
OBS_VALUE (The measured value)			
Attributes			
Concept	Assignment level	Representation	Description
OBS_STATUS (mandatory)	Observation	CL_OBS_STATUS	The observation status (normal, estimated, forecast, etc). Normal in this case.
OBS_CONF (optional)	Observation	CL_OBS_CONF	The observation confidentiality. All published data are free, but it's good practice to mention it anyway.
TIME_FORMAT (mandatory)	Series	Time Duration Set	ISO 8601 [http://en.wikipedia.org/wiki/ISO_8601#Duration] compliant way to describe duration (in this case, P1D)
COLLECTION (mandatory)	Series	CL_COLLECTION	When the information was collected (end of period etc). In this case, A (Average of observations through period).
UNIT (mandatory)	Group	CL_UNIT	The unit used (for example, RUB for Russian rouble).
UNIT_MULT (mandatory)	Group	CL_UNIT_MULT	Whether the data is in millions, billions, etc. In this case, the value is "0" (unit).
DECIMALS (mandatory)	Group	CL_DECIMALS	The number of decimals.
TITLE_COMPL (mandatory)	Group	Up to 1050 characters	A human-readable title describing a certain group of data (e.g.: ECB reference exchange rate, Australian dollar/Euro, 2:15 pm (C.E.T.)).

Creation of the SDMX-ML Structure Definition file

Now that we have a good overview of the structure of the data we want to make available on the website, we can formally define this structure using SDMX-ML. The Structure Definition format is the one to use for this kind of tasks, as it contains the description of structural metadata such as key families, concepts, and code lists.

The header

Not taking into account the initial XML declaration, the XML file (`ecb_exr1_structure.xml`) starts with the `Structure` element and the standard SDMX `Header`.

```
<?xml version="1.0" encoding="UTF-8"?>
<Structure
        xmlns="http://www.SDMX.org/resources/SDMXML/schemas/v2_0/message"
        xmlns:message=
                "http://www.SDMX.org/resources/SDMXML/schemas/v2_0/message"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="
                http://www.SDMX.org/resources/SDMXML/schemas/v2_0/message
                SDMXMessage.xsd
                http://www.SDMX.org/resources/SDMXML/schemas/v2_0/structure
                SDMXStructure.xsd">
<Header>
        <ID>IREF000506</ID>
        <Test>false</Test>
        <Name>ECB structural definitions</Name>
        <Prepared>2006-10-25T14:26:00</Prepared>
        <Sender id="4F0"/>
</Header>
```

The root element is defined (`Structure`), with the namespaces and schema definitions attached as attributes (`SDMXMessage.xsd` and `SDMXStructure.xsd`). The SDMX Message namespace is used by all other SDMX-ML namespace modules. It contains the common message constructs, including the common `Header` information (Message `ID`, `Name`, `Prepared` date, etc). The `id` attribute of the `Sender` element specifies the sender of the data, which is the European Central Bank in this case (code `4F0` within this context of data sender, taken from the code list CL_ORGANISATION).

Apart from the `Header` element, the XML file also contains the 3 following main elements, all belonging to the SDMX Structure namespace: `CodeLists`, `Concepts` and `KeyFamilies`.

The concepts

The `Concepts` element contains a list of concepts used to identify and describe the data. All the concepts used in the Data Structure Definition are included in this list.

Each `Concept` element contains 2 attributes: the ID of the agency responsible for the concept ("ECB") and the concept ID (for example "UNIT_MULT"). Both are identifiers, and, as such, are language independent. The `Name` element contains a language-dependent description of the concept (as specified in the `xml:lang` attribute).

```
<Concept agencyID="ECB" id="COLLECTION">
```

```
        <Name xml:lang="en">Collection indicator</Name>
</Concept>
```

The code lists

The `CodeLists` element contains a list of `CodeList` elements. Each `CodeList` element contains 2 attributes: the ID of the Agency responsible for the code list ("ECB") and the code list ID (for example "CL_EXR_SUFFIX"). The `Name` element contains a description of the code list in a specific language. Each code list also contains a list of codes, with an attribute for the code value and a language-dependent description of the code. The code lists define the possible values taken by the dimensions and by the coded attributes.

```
<CodeList agencyID="ECB" id="CL_EXR_SUFFIX">
        <Name xml:lang="en">Exch. rate series variation code list</Name>
        <Code value="A">
                <Description xml:lang="en">Average or standardised
                measure for given frequency</Description>
        </Code>
        <Code value="E">
                <Description xml:lang="en">End-of-period</Description>
        </Code>
</CodeList>
```

The key families

Now that we have our list of concepts and code lists, we can start defining the structure of our Data Structure Definition.

The `KeyFamily` element contains the ID of the Agency responsible for the Data Structure Definition definition ("ECB"), the `id` for the Data Structure Definition ("ECB_EXR1"), a `uri` (we use the Data Structure Definition namespace for this purpose) and the name of the Data Structure Definition in a specific language.

```
<KeyFamily agencyID="ECB" id="ECB_EXR1"
        uri="http://www.ecb.int/vocabulary/stats/exr/1">
        <Name xml:lang="en">Exchange Rates</Name>
<Components>
```

Then the components of the Data Structure Definition are defined, starting with the dimensions.

```
<Dimension conceptRef="FREQ" codelist="CL_FREQ" isFrequencyDimension="true"/>
<Dimension conceptRef="CURRENCY" codelist="CL_CURRENCY"/>
<Dimension conceptRef="CURRENCY_DENOM" codelist="CL_CURRENCY"/>
<Dimension conceptRef="EXR_TYPE" codelist="CL_EXR_TYPE"/>
<Dimension conceptRef="EXR_SUFFIX" codelist="CL_EXR_SUFFIX"/>
<TimeDimension conceptRef="TIME_PERIOD"/>
```

Each `Dimension` element contains references to a descriptor concept and the code list from which the dimension value has to be taken. For example, the dimension which represents the concept of frequency takes its values from the CL_FREQ code list and, as such, can only take one of the fol-

lowing values: A (Annual), B (Business), D (Daily), E (Event), H (Half-Yearly), M (Monthly), Q (Quarterly) and W (Weekly). The `isFrequencyDimension` is attached to the dimension which represents the frequency (`FREQ` in this case); there can be only one such dimension per Data Structure Definition.

The `TimeDimension` is a special dimension that must be included in any Data Structure Definition which will be used for time-series formats (such as the GenericData, CompactData and UtilityData).

The order of declaration of the dimensions is important as it describes the order in which the dimensions will appear in the keys (except for the time dimension, which is not part of the key).

The `Group` element declares any useful groupings of data, such as sibling groups.

```
<Group id="Group">
        <DimensionRef>CURRENCY</DimensionRef>
        <DimensionRef>CURRENCY_DENOM</DimensionRef>
        <DimensionRef>EXR_TYPE</DimensionRef>
        <DimensionRef>EXR_SUFFIX</DimensionRef>
</Group>
```

Then, we indicate which attribute will contain the measured value. Conventionally, it is associated with the `OBS_VALUE` concept.

```
<PrimaryMeasure conceptRef="OBS_VALUE"/>
```

Finally, we list the attributes. An `Attribute` element will contain information such as the concept used for the attribute, the attachment level (i.e. "Observation", "Group", "Series", "DataSet") and whether it is mandatory or not (i.e. "Mandatory" versus "Conditional"). Coded attributes will indicate from which code list the values should be taken, while, for uncoded attributes, a specific format may by specified using the `TextFormat` element. For attributes attached to the group level, we specify the id of the group to which the attributes are attached with an `AttachmentGroup` element. The concept of time format is identified with the `isTimeFormat` attribute with a value of `true` and is typically a mandatory series level attribute whose value is taken from ISO8601.

```
<Attribute conceptRef="TIME_FORMAT" attachmentLevel="Series"
        assignmentStatus="Mandatory" isTimeFormat="true">
        <TextFormat textType="String" maxLength="3"/>
</Attribute>
<Attribute conceptRef="OBS_STATUS" attachmentLevel="Observation"
        codelist="CL_OBS_STATUS" assignmentStatus="Mandatory"/>
<Attribute conceptRef="DECIMALS" attachmentLevel="Group"
        codelist="CL_DECIMALS" assignmentStatus="Mandatory">
        <AttachmentGroup>Group</AttachmentGroup>
</Attribute>
```

As a last step, we use an XML validating parser to validate our Structure Definition file and make sure that it is compliant with the SDMX-ML standard [4]. You can use the task `validateStructure` supplied in the Ant build file to perform this step.

```
bash$ ant validateStructure
Buildfile: build.xml

validateStructure:
[xmlvalidate] 1 file(s) have been successfully validated.

BUILD SUCCESSFUL
Total time: 2 seconds
```

Creation of the schema file

Now that we have a valid Structure Definition file, we can generate an XML schema file for our Data Structure Definition. The SDMX initiative offers some free tools to developers [5] and one of these tools is an XSL file (`StructureToCompact.xsl`) that creates an XML schema out of a Structure Definition file for a selected Data Structure Definition [6].

To create our schema, we need to:

1. Use an XSL parser using our Structure Definition file (`ecb_exr1_structure.xml`) as XML input file, the XSL file (`StructureToCompact.xsl`), and the desired filename for our XML schema file (for example: `ecb_exr1_compact.xsd`) as parameters.

2. Open the file in an editor and add the default namespace in the `xs:schema` element (`xmlns="http://www.ecb.int/vocabulary/stats/exr/1"`) [7].

You can use the task `generateCompactSchema` supplied in the Ant build file to perform this step.

```
bash$ ant generateCompactSchema
Buildfile: build.xml

generateCompactSchema:
     [xslt] Processing ecb_exr1_structure.xml to ecb_exr1_compact.xsd
     [xslt] Loading stylesheet StructureToCompact.xsl

BUILD SUCCESSFUL
Total time: 2 seconds
```

We now have an XML Schema file (`ecb_exr1_compact.xsd`), which we will use to validate the XML data file before publishing it on our website.

Creation of the SDMX-ML data file for the time-series view

It's now time to create our XML data file (`ecb_exr1_compact.xml`). This is normally done by ~~extracting the data~~ out of a database and creating the XML data file.

[4] SDMX schemas are available on the SDMX website and can also be downloaded at the following location: http://www.sdmx.org/data/2_0/SDMX_2_0_SECTION_03B_SDMX-ML_Schemas_and_Samples.zip [http://www.sdmx.org/data/2_0/SDMX_2_0_SECTION_03B_SDMX-ML_Schemas_and_Samples.zip] . For the purpose of this exercise, the schema files have been downloaded and placed in the same directory as the files written for this tutorial.

[5] The tools can be downloaded (after registration) from: http://www.metadatatechnology.com/software/index.cgi

[6] Currently, the tools work with version 1 of the standard only. We have adapted the XSL code, so as to work with version 2 of the standards as well. Changes between the original version and the ECB version are commented.

[7] Due to some technical limitations (you cannot set the attribute xmlns in XSLT 1.0), we need to perform this step after the XML schema file has been generated. For this tutorial, this step is performed manually.

The header

When we open the file, we should recognize some similarities with the Structure Definition file, as all SDMX messages share some common constructs (i.e.: the elements from the SDMXMessage namespace, such as the `Header`).

```
<CompactData
        xmlns="http://www.SDMX.org/resources/SDMXML/schemas/v2_0/message"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation=
        "http://www.SDMX.org/resources/SDMXML/schemas/v2_0/message
        SDMXMessage.xsd">
<Header>
        <ID>EXR-HIST_2006-11-29</ID>
        <Test>false</Test>
        <Name xml:lang="en">Euro foreign exchange reference rates</Name>
        <Prepared>2006-11-23T08:26:29</Prepared>
        <Sender id="4F0">
                <Name xml:lang="en">European Central Bank</Name>
                <Contact>
                        <Department xml:lang="en">DG Statistics</Department>
                        <URI>mailto:statistics@ecb.int</URI>
                </Contact>
        </Sender>
</Header>
```

One difference is that the `Header` element contains a bit more information about contact information.

The DataSet element

The message continues with the higher possible level of grouping [8], the `DataSet`. A namespace, an XML schema and the dataset ID are added to the element.

```
<DataSet
        xmlns="http://www.ecb.int/vocabulary/stats/exr/1"
        xsi:schemaLocation="http://www.ecb.int/vocabulary/stats/exr/1
        ecb_exr1_compact.xsd"
        datasetID="ECB_EXR1">
```

The Group element

Then comes the next level of grouping, the `Group` element. As it is a sibling group, it contains all dimensions (CURRENCY, CURRENCY_DENOM, EXR_TYPE and EXR_SUFFIX), except the frequency (FREQ), which is wild carded, and the date/time information (TIME_PERIOD), which is attached to the observation level. Apart from the dimensions, it also includes the attributes which are attached to the group level (DECIMALS, UNIT, UNIT_MULT and TITLE_COMPL).

```
<Group CURRENCY="AUD" CURRENCY_DENOM="EUR" EXR_TYPE="SP00"
        EXR_SUFFIX="A" DECIMALS="4" UNIT="AUD" UNIT_MULT="0"
        TITLE_COMPL="ECB reference exchange rate, Australian dollar/Euro"/>
```

[8] Multiple datasets may be merged in the same data file using an SDMX MessageGroup message.

The Series element

Going further down into the package grouping, we reach the `Series` level, which contains the same dimensions as the group plus the frequency. It also includes the attributes which are attached to the series level (TIME_FORMAT and COLLECTION).

```
<Series FREQ="D" CURRENCY="AUD" CURRENCY_DENOM="EUR"
        EXR_TYPE="SP00" EXR_SUFFIX="A" TIME_FORMAT="P1D" COLLECTION="A">
```

The Observation element

The `Series` element also contains the list of observations, and we have now reached the lower possible level in the package grouping. The observations contain the time dimension (TIME_PERIOD), the measured value (OBS_VALUE) and the attributes attached to the observations level (OBS_STATUS and OBS_CONF).

```
<Obs TIME_PERIOD="1999-01-04" OBS_VALUE="1.9100" OBS_STATUS="A" OBS_CONF="F" />
```

These last three elements (`Group`, `Series` and `Obs`) will be repeated for all group of data to be published.

Now that we have our data file ready, we should validate it using an XML validating parser, before publishing it on our website. You can use the task `validateData` supplied in the Ant build file to perform this step.

```
bash$ ant validateData
Buildfile: build.xml

validateData:
[xmlvalidate] 1 file(s) have been successfully validated.

BUILD SUCCESSFUL
Total time: 6 seconds
```

Once this has been done, the job of the data publisher is over and we have reached our goal of publishing an SDMX-ML data file on our website. We can now use it to retrieve and display exchange rates data.

Data consumer: retrieving and displaying exchange rates

Several XML technologies are available for processing an XML data file [9].

[9] Familiarity with XML technologies is expected, so only a brief overview of these technologies is offered. There are of course other technologies that one could use to process an XML data file but we will limit ourselves to XSLT, SAX and the DOM, for the purpose of this tutorial, as these are probably the most frequently used ones, that are not language or platform dependent.

1. **The XSL Transformation language** [10]. XSLT can be used to transform the XML data file (or parts of it) into other formats, such as (X)HTML, CSV, PDF, WML, etc. For instance, we could use this technology to create an (X)HTML table displaying the rates for all currencies at a specific period.

2. **The Document Object model (DOM)** [11]. DOM is an object-oriented model of an XML document, which represents it as a tree structure. You can use the DOM to read from and write to an XML data file. However, DOM stores the entire document tree in memory, which means that it is resource intensive, especially for large XML data files. If all we need is a sequential read or a one-time selective read, DOM might be overkill.

3. **The Simple API for XML (SAX)** [12]. SAX uses an event-driven model and handles an XML file as a unidirectional stream of data. SAX parsing is usually faster and the memory footprint is much smaller compared to a DOM construct. The price to pay is limitation: you cannot write to an XML data file, reading is unidirectional (so you can't go back in the XML file if needed, and you have to start from the beginning again) and there is no representation of the structure of the XML data file.

For the scope of this tutorial, we will use SAX to display the exchange rate for a certain currency at a certain date [13], and XSLT to extract some data out of the SDMX-ML data file and create an HTML table with the rates for all currencies at a specific period [14].

Parsing the SDMX-ML data file using SAX

The first step of the process, before we use the SDMX-ML data file, is to validate it. Once this is done, we may use a SAX parser to extract the exchange rate for a specific currency at a specific point in time.

A SAX parser is event-based. It will call one method in the application when it encounters an element tag and another one when it encounters, for instance, some text. So it's up to the developer to write the call-back methods. Java provides classes and methods to work with SAX, but a tutorial of using SAX in Java is beyond the scope of this exercise [15].

Apart from the setup code needed (see the `main` method in the `SAXGetRate.java` file), the `startElement` method shows what we are after: when we find the series for the selected currency, we search for the observation that matched the supplied period and exit when the value has been found.

Just open a shell and try it out following this syntax:

```
bash$ java -cp tutorial.jar SAXGetRate ecb_exr1_compact.xml currency period
```

So, for example

```
bash$ java -cp tutorial.jar SAXGetRate ecb_exr1_compact.xml USD 2006-11-21
Exchange rate value for USD on 2006-11-21: 1.2814
```

[10] XSLT [http://www.w3.org/TR/xslt] is a standard endorsed by the W3C.
[11] DOM [http://www.w3.org/DOM/] is a standard endorsed by the W3C.
[12] Although not endorsed by the W3C, SAX [http://www.saxproject.org/] is a de facto standard for the XML industry.
[13] We have also created a small DOM class that performs the same operation so that we can compare how the 2 technologies perform for the same task.
[14] The code used for this purpose is very primitive and is merely made available to offer simple examples on how we can extract data out of an SDMX-ML data file. It should by no mean be considered quality code or even useful for any other purpose than the one described above.
[15] Interested readers may find the SUN J2EE tutorial [http://java.sun.com/j2ee/1.4/docs/tutorial/doc/] useful.

Using XSLT to create an (X)HTML table with the daily rates

We now want to generate an (X)HTML table containing the exchange rates for all the currencies at a specified period. We will also use the Structure Definition file so as to extract the name for the currencies, which is nicer than just displaying the currency code.

Again, an introduction to XSLT is out of the scope of this tutorial but as one will notice, the XSL script is fairly small. Apart from the output settings, we assign the value passed to the script for the desired period to a variable and we get the list of currencies out of the Structure Definition file (`ecb_exr1_structure.xml`). We then match the root of the SDMX-ML data file and output the basic HTML information (head, body, table, etc). We then match all series, and for each series, we get the observation value for the supplied date, and add it to the HTML table.

To generate the (X)HTML table, we use an XSLT processor, using the SDMX-ML data file (`ecb_exr1_compact.xml`) as the input file, `sdmxml2html.xsl` for the XSL file and the desired filename for the HTML table (for example `ecb_exr1_table.html`) as parameters. You can use the task `generateHTMLTable` supplied in the Ant build file to perform this step.

```
bash$ ant generateHTMLTable
Buildfile: build.xml

generateHTMLTable:
     [xslt] Processing ecb_exr1_compact.xml to ecb_exr1_table.html
     [xslt] Loading stylesheet sdmxml2html.xsl

BUILD SUCCESSFUL
Total time: 5 seconds
```

We have now finished with the 2 utilities that needed to be created for the data consumer and we now know enough to be able to build fully-fledge SDMX-based software.