X-definition 4.2

Java programming guide

Author: **Václav Trojan**
Version: **4.2.0.0**
Date: **2022-06-15**

Contents

Notice

Questions, remarks, and bug reports please send to:
xdef@syntea.cz.
The current version of X-definition can be downloaded at:
https://github.com/Syntea/xdef,
or at:
https://www.xdefinice.cz/en/.
Source data of all examples are available in the zip file in the directory „examples“. Download it at:
https://www.xdefinice.cz/en/download-sekce/
A detailed description of the X-definition is available at:
https://xdef.syntea.cz/tutorial/en/userdoc/xdef-4.1.pdf
The Java documentation is available at:
https://xdef.syntea.cz/tutorial/en/userdoc/xdef-javadoc.jar
A tutorial with which a reader can learn the use of the X-definition is available at:
https://xdef.syntea.cz/tutorial/en.
The downloaded zip file also contains the source files of examples presented in this text.
An introduction to the construction process is available at:
https://xdef.syntea.cz/tutorial/en/userdoc/xdef-4.1_construction_mode.pdf
A description of the X components is available at:
https://xdef.syntea.cz/tutorial/en/userdoc/xdef-4.1_X-component.pdf
The JAR file with compiled Java code is available at:
https://xdef.syntea.cz/tutorial/en/userdoc/xdef.jar
or at:
https://www.xdefinice.cz/en/download

1 Introduction

This text is an introduction to programming in the X-definition programming language. Part of this is a sample of practices on how to program typical tasks in Java. In several examples, derived from a simplified order processing task in a hypothetical system, we will show how to use the X-definition definitions in Java. We assume that the reader is familiar with the basics of programming in Java and is familiar with the XML language. We also assume that the reader has become familiar with programming in Java.

2 How to run X-definition

X-definition can be started in two different modes, either in a mode, that validates and processes the input XML data according to the specified X-definition (a particular model) or, using the specified model in the X-definition. The resulting XML document will be constructed according to this model.

In any case, we first have to compile the set of X-definitions and create the **XDPool** object. This object is reentrant (all values are constant after creation) and it is, therefore, appropriate to save this object in a static variable or "singleton" object. To run X-definitions, create an **XDDocument** object from the XDPool object.

The result of running X-definitions is either an XML element or a Java instance of the class in which the values of the processed XML document are accessible (the so-called **X-component**). Reports of errors and statuses during the process are written into the so-called "**reporter**". The reporter can store the records either in working memory (org.xdef.ArrayReporter class) or in a file (class org.xdef.FileReporterWriter). In the methods which run X-definition, we can specify a reporter as a parameter. If the reporter is not specified (we will state "null"), then if bugs are reported during processing, the program will throw an exception with the listed errors. If the reporter is specified, then you can test the presence of errors or warnings with methods "errors ()" or "ErrorWarnings ()" respectively.

2.1 Validation mode (and processing) of the input document

The validation and processing mode is run by the „xparse“ method on the instance of the XDDocument class. The input data is passed to the method by the first parameter. It is advisable to specify the reporter as the second parameter.

The program finds the appropriate model in the X-definition (which must be declared "root" in the header of the X-definition). Further processing takes place according to this model. During processing, the occurrence of individual elements is checked by quantifiers and text values are checked using the corresponding validation method. When processing in different situations, the appropriate statements declared in the X-script are called.

If an element is required, but it is not present in the input data, an error is reported and a command from the "onAbsence" section is called. If, on the other hand, the number of elements in the input data exceeds the maximum from the quantifier, an error is reported, and eventually, the "onExcess" section command is invoked. If the value of the data is found to be valid by the validation method, then, if the "onTrue" section is specified, the command of this section is called. If the data value is invalid, it will report an error and eventually, invoke the "onFalse" section command.

After reading all the attributes, the command "onStartElement" will be called (if specified). And the corresponding closing command at the end of the element processing is called "finally" (if specified).

Of particular importance is the "forget" section. This section does not contain any command, but it will cause the processed item to be released from the computer's memory. This is for processing large data that would otherwise not fit into processor working memory during processing.

2.2 Document construction mode

The document construction mode ("create mode") does not process an input XML document, creates one according to a specified model. The construction mode is run by the "xcreate" method on the instance of the

XDDocument class. The first parameter of the xcreate method specifies the construction model for the resulting XML output. Again, it is a good idea to specify a reporter parameter.

The individual parts of the document being created are constructed according to the given model based on the so-called context. Context data is used at the time of creation to construct the target XML object. The context either does or doesn't exist in the construction of each part of the document. If the "create" section command is specified in the X-script of an XML model, then the value of a result of the "create" section command becomes the context for creating this XML object (this value must be usable as a context). This value then becomes the default for the construction of all descendants. The required type of this value varies according to the type of item being created. The commonly used method of the "create" section is an XPath expression that works with the XML document assigned as the default context for the construction of a result (in this way you can "transform" an XML document from the context into the final form).
E.g. for text values and/or attributes, the context value is converted to a string. This value is then set as the value of the created item. If the context value is "null" or an empty string, the attribute or text node is not constructed.

The context of elements can be a value of the following types:

- Element …… the result is made up of the content of this element. If the value is "null" then the element is not created.
- integer …… the number of elements corresponding to the minimum of this number and maximum in the quantifier is created.
- String …….. as a context, an element is created with a text node from the given value (ie, if the desired child is a text node, it is created from that string)
- Container …….. if the number of elements in the container is 0, nothing is created. Otherwise, elements are selected and this number of elements is created, corresponding to the minimum of number of elements and the maximum in the section occurs.
- boolean …….. an element is created only if the value is true.
- null ..... the element is not created

Example with an explicit specification of the "create" section:

```
<A xd:script="create 1" x="string; create 'abc'">
  optional string; create 'def';
</A>
```

The result will be:

```
<A x="abc">def</A>
```

3 Tools for programming X-definitions in Java

3.1 Compilation of X-definition (creating XDPool object)

To work with X-definitions, we will need an XDPool object (described by **org.xdef.XDPool**). XDPool is a binary object created by the compilation of X-definitions from source form. For example, the XDPool object can be created using the static **org.xdef.XDFactory.compileXD(...)** method. The first parameter of this method is the value of "java.util.Properties", where you can set some processing parameters (see ... XXXX). If this parameter is "null," the values set by the system (System.getProperties() method) are used. Other parameters then specify the X-definition sources to be compiled.

The XDPool object is reentrant (all data stored in it is constant). This means that one instance of this object can be used multiple times, or it can be shared in multiple processes. Therefore, you can save the created XDPool to a file and create it from the stored data again. This can be used for efficient programming: the XDPool object can be prepared separately from the source shape and saved to a file. You can then create the XDPool from this file (you do not need the source form of X-definitions and XDPool creation is then much faster). Another possibility is to generate the source of a Java class from the XDPool object using the "**XDFactory.genXDPoolClass**" method, so you can obtain the XDPool object any time from this class can then be part of a project distributed in source form.

*Note: In some cases, you need to create an XDPool object from a variety of sources (for example, a combination of source files stored in a database, from the Internet and the local file system, etc.). In this case, we must first create an object described by the interface **org.xdef.XDBuilder** uses the XDFactory class and uses it to incrementally create translation using setSource methods. Finally, we create the XDPool object using the **genXDPool** method called from XDBuilder. However, this procedure is not necessary in most cases, and it is usually provided with the static "compileXD" method from the XDFactory class. See Java documentation for the XDBuilder class.*

3.2 Create an instance of the XDDocument object

To work with X-definitions, use the XDPool object to create an instance of the XDDocument object that is required to work with an XML document. You can create this object from the XDPool object by using the "**createXDDocument**" method. The parameter of this method specifies a specific default X-definition used for validation or construction mode. The XDDocument is no longer reentrant, and its instance refers to the processed data. The result of the process of the XDDocument is an XML document (or an X-component object). When processing the X-definition, a protocol (error information, warnings, etc.) will also be generated. The values of the variables declared in the X-definitions are stored In the XDDocument object. In addition to the resulting XML document, it is also possible to obtain variable values from the XDDocument (using the „**getVariable**“ method). A protocol with the error reports, warnings, etc. is stored in a so-called **reporter** passed to the X-definition processor using the parameter of a startup method.

You can also set some values to the XDDocument before starting X-definition. The values of variables declared in X-definition can be declared using the **setVariable** method. The context that will be used in the construction mode can be set using the **setXDContext** method. By using the setUserObject method, you can set a user object that can be used in external user methods to link the program to external user data. The initial processing starts with „xparse“ (XML document validation and processing) or „xcreate“ (XML object construction).

To parse and validate XML data use the method **xparse**. To parse and validate JSON/XON data use the method **jparse** and to parse and validate Windows INI/Properties data, use the method **iparse**.

3.3 Other Java objects in X-definitions

When the X-definition is processed it internally creates temporary control objects that relate to the currently processed part of the XML object. All of these objects are based on the common interface **org.xdef.proc.XXNode** (this also applies to XDDocument). The **org.xdef.proc.XXElement** object is created when processing an XML element and the **org.xdef.proc.XXData** object is created when processing text values (ie, attributes or text nodes). These objects exist dynamically only during processing. However, they can be accessed in external "user" Java methods called from the X-script. In these methods, it is often necessary to work with detailed information about

the state of the processed objects or to influence them. Therefore, the object XXNode (or XXElement or XXData) that is created when processing the appropriate XML object can be passed as a parameter to an external method. If the external method requires it, this object can be passed to the external method (it must be always specified as the first parameter, but this parameter is not specified in the X-script). It must be declared in the relevant external method and the X-definitions automatically pass this value).

The values of the parameters and variables declared in X-definitions are represented by the interface **org.xdef.XDValue**. For common Java values (numbers, String, etc.), the external procedure parameters can be listed as a parameter list in the usual way, and the compiler automatically ensures conversion from the inner form to the Java object. The second way to pass parameters to external methods is to specify the parameter as an array of XDValue values. In this case, an array of objects with required values is passed to the method according to the list of parameters specified in a method call in the X-script. Thus, one external method can be executed from methods with a different number of parameters in the X-script whose values are set as values of the array passed to the method.

In some special cases, the program needs to know the values present in the models in X-definitions (such as occurrence limits, etc.). These objects are accessible via the interface **org.xdef.model.XMDefinition**, **org.xdef.model.XMElement** and **org.xdef.model.XMData**. They are the objects corresponding to the models declared in the X-definitions, or parts thereof. Access to these objects is made possible from XXNode using the **getXMDefinition**, **getXMElement**, **getXMData** methods. The XDPool can be obtained using the **getXDPool** method.

Some processing parameters can be set using Properties. E.g. If "xdef_doctype" is set to "false," the X-definition does not allow the occurrence of "DOCTYPE," which is important from a security point of view, or the property "xdef_warnings" to "false" means the warning messages will not cause a program exception if reported. The language of the report message by the property "xdef_language" For more options see **org.xdef.XDConstants**.

The error and information messages are passed to the X-definition processor by a reporter that allows you to write the processing protocol in the form of **org.xdef.sys.Report** objects to a file or the memory of your computer.

The date and time values are implemented in the X-definitions processor by the **org.xdef.sys.SDatetime** classes. For intervals, the **org.xdef.sys.SDuration** class is used.
The class **org.xdef.sys.BNFGrammar** is used to work with text values described by the extended BNF grammar.

4 Examples of how to work with XML data

In this chapter, we discuss some examples based on a specific hypothetical list of orders in a form of XML. Our task processes an input file of orders and checks their accuracy. In the examples, we describe both the source text of the Java program as well as the X-definitions and input data. The files are available for download together with the X-definition Jar file at *https://github.com/Syntea/xdef, or* https://www.xdefinice.cz/en/download.

4.1 Task 1

Let's first check if the order structure is formally correct. The input data is in the "task1/input" directory. If no error is reported, the processed (checked) order will be stored in the "task1/output" directory. If an error is detected the file with errors will be stored in the "task1/errors /" directory.

The X-definitions and the Java source code used in each variant of this example are in the "src/task1 /" directory. In each of the variants of our example, we will show successive different ways of using X-definitions and supporting means.

4.1.1 Variant 1

Let's start with the simplest variant. Using X-definition, we check the correctness of the input data, and if no error is found, we write the data into the output directory. Otherwise, we write the error log file (in text form) into the directory "task1/output/Order_1err.txt".

The formally correct order is in the file "task1/Input /Order.xml":

```
<Order Number="123" CustomerCode="ALFA">
  <DeliveryPlace>
    <Address Street="Oldroad" House="5" City="NewTown" ZIP="32321" />
  </DeliveryPlace>
  <Item ProductCode="0002" Quantity="2" />
  <Item ProductCode="0003" Quantity="1" />
</Order>
```

X-definition describing the input data ("src/task1/Order1.xdef"):

```
<xd:def xmlns:xd="http://www.xdef.org/xdef/4.2" name="Order" root="Order">

<Order Number="int" CustomerCode="string(1,20)">
  <DeliveryPlace>
    <Address Street="string(2,100)"
      House="int(1,9999)"
      City="string(2,100)"
      ZIP="num(5)"/>
  </DeliveryPlace>
  <Item xd:script="occurs 1..10" ProductCode="num(4)" Quantity="int(1,1000)"/>
</Order>

</xd:def>
```

When creating a program, we first need to obtain an XDPool object compiled from the source X-definitions. Note because this object contains only constants, it can be stored in a static final variable. From it, we create an XDDocument needed for further work. We also prepare a "reporter" to write the error log file (the class ArrayReporter stores the log in the working memory of the computer). Using the "xparse" method, we start validation and processing the input data according to the X-definition. Then we will test whether any errors have been reported (i.e. whether the input data corresponds to the X-definition) and write the processed XML document into the appropriate directory (using the "KXmlUtils.writeXml" utility). If the validation isn't successful an error log is produced.

Java source (the file „src /task1/Order1.java"):

```
package task1;

import org.xdef.sys.ArrayReporter;
import org.xdef.xml.KXmlUtils;
import org.xdef.XDDocument;
import org.xdef.XDFactory;
import org.xdef.XDPool;
import java.io.PrintStream;
import org.w3c.dom.Element;
```

```
public class Order1 {
        // Compile the X-definition source to the static variable with XDPool object
        static final XDPool xpool = XDFactory.compileXD(null, "src/task1/Order1.xdef");

        public static void main(String... args) throws Exception {
                // Create an instance of the XDDocument object (from XDPool)
                XDDocument xdoc = xpool.createXDDocument("Order");

                // Prepare the error reporter
                ArrayReporter reporter = new ArrayReporter();

                // Run validation mode (you can also try task1/input/Order_err.xml)
                Element result = xdoc.xparse("task1/input/Order.xml", reporter);

                // Check if an error was reported
                if (reporter.errors()) {
                        // Print errors to the file
                        PrintStream ps = new PrintStream("task1/errors/Order_err.txt ");
                        reporter.printReports(ps);
                        ps.close();
                        System.err.println("Incorrect input data");
                } else {
                        // No errors, write the processed document to the file
                        KXmlUtils.writeXml("task1/output/Order.xml", result);
                        System.out.println("OK");
                }
        }
}
```

The output of the formally correct order will be in the file „task1/output/Order_1.xml“:

```
<?xml version="1.0" encoding="UTF-8"?>
<Order CustomerCode="ALFA" Number="123">
  <DeliveryPlace>
    <Address City="NewTown" House="5" Street="Oldroad" ZIP="32321"/>
  </DeliveryPlace><Item ProductCode="0002" Quantity="2"/>
  <Item ProductCode="0003" Quantity="1"/>
</Order>
```

If the input data contains an error (in our case, we changed the "Quantity" attribute in the first element "Item" is not numeric, see file *task1/input/Order_err.xml*):

```
...
  <Item CommodityCode="0002" Quantity="xx"/>
...
```

then the error log will be written to the file: *„task1/errors/Ordererr.txt“:*

```
E XDEF809: Incorrect value of 'int'; line=6; column=38;
source="file:/D:/cvs/DEV/java/examples/task1/input/Order_err.xml"; xpath=/Order/Item[1]/@Quantity; X-
position=Order#Order/Item/@Quantity
```

4.1.2 Variant 2: save compiled X-definitions (XDPool) to a binary file

Let's show how to proceed if we want to save the XDPool object created from the source X-definition in a binary file and then use it. XDPool implements an interface Serializable, so it is possible to use ObjectOutputStream and ObjectInputStream to write and read the compiled instance of XDPool. In the program below we will compile the X-definition to the XDPool object and save it to the file "src/task1/Order1.xp".

Java program „src/task1/Order1a_gen.java“ generates the binary file „src/task1/Order1a.xp“ from the X-definition:

```
package task1;

import org.xdef.XDFactory;
import org.xdef.XDPool;
import java.io.FileOutputStream;
import java.io.ObjectOutputStream;
import java.util.Properties;

public class Order1a_gen {

        public static void main(String... args) throws Exception {
                // Compile the XDPool object from the X-definition source file
                Properties props = new Properties();
                XDPool xpool = XDFactory.compileXD(props, "task1/output/Order1a.xp");
                // Write the XDPool object to the file
```

```
                ObjectOutputStream outstr= new ObjectOutputStream(
                        new FileOutputStream("src/task1/Order1a.xp"));
                outstr.writeObject(xpool);
                outstr.close();
        }
}
```

The next program creates the XDPool object from this file. Of course, the time to create an instance of the XDPool object will be significantly shorter than compilation from the source X-definition.
Note that with each new version of X-definitions, it is advisable to re-compile the binary file from XDPool translated from the new source code, since the formal one may not be compatible with the actual version.

Java program „src/task1/Order1a.java" reads XDPool from the file „src/task1/Order.xp" which was generated by the program „Order1a_gen":

```
package task1;

import org.xdef.sys.ArrayReporter;
import org.xdef.xml.KXmlUtils;
import org.xdef.XDDocument;
import org.xdef.XDPool;
import java.io.FileInputStream;
import java.io.ObjectInputStream;
import java.io.PrintStream;
import org.w3c.dom.Element;

public class Order1a {
        public static void main(String... args) throws Exception {
                // Read the XDPool object from the file
                ObjectInputStream in = new ObjectInputStream(
                        new FileInputStream("src/task1/Order1a.xp"));
                XDPool xpool = (XDPool) in.readObject();
                in.close();

                // Create an instance of the XDDocument object (from XDPool)
                XDDocument xdoc = xpool.createXDDocument("Order");

                // Prepare the error reporter
                ArrayReporter reporter = new ArrayReporter();

                // Run validation mode (you can also try task1/input/Order_err.xml)
                Element result = xdoc.xparse("task1/input/Order.xml", reporter);

                // Check if an error was reported
                if (reporter.errorWarnings()) {
                        // Print errors to the file
                        PrintStream ps = new PrintStream("task1/errors/Order_err.txt");
                        reporter.printReports(ps);
                        ps.close();
                        System.err.println("Incorrect input data");
                } else {
                        // No errors, write the processed document to the file
                        KXmlUtils.writeXml("task1/output/Order.xml", result);
                        System.out.println("OK");
                }
        }
}
```

4.1.3 Variant 3: Creating an error file using an external method

Let's modify our example now by manipulating errors using external Java methods and not by methods in the X-script. These external methods will be invoked from the X-script, but the code will be in the Java class. Because the error log is generated by an external method, there is no need to specify the element model with errors in the X-definition. The error data will be generated directly in the appropriate Java method. The price we pay for this is that the structure of the error document is not described in X-definition, and is dependent on the Java code.

Note that we added the declaration of the external methods used in the X-script to the X-definition. The methods that are called from the X-script must be static and their parameters must match the types that are specified in the declaration section.

Since the external methods must be static, it is necessary to allow external methods to work with instances of objects that are passed by the "getUserObject" method. This object must be stored in the XDDocument before running X-definition of Java using the "setUserObject" method (in our example, we use the XML document to save the errors). External Java methods called from a script must have certain properties (they must be declared "static" and "public").

The first parameter of an external method declared in Java can be of the XXNode type. This parameter is passed automatically from the X-definition. Even if it is specified in Java code this parameter is not declared in the method call in the X-script (however, it is declared in the declaration of the Java external method in the declaration section). In our example, we need it to get the necessary data about the processing state, such as the current position of the input source being processed, but also to obtain the connected user object (using the „getUserObject“ method).

Types of other parameters of the call statement in the X-script match the parameter types specified in the external method. The second parameter in our example corresponds to the error code number. The external Java method from this data creates an "Error" element and adds it as a child to the root element "Errors".

Java program (the file „src/task1/Order3.java“):

```
package task1;

import org.xdef.sys.ArrayReporter;
import org.xdef.xml.KXmlUtils;
import org.xdef.XDDocument;
import org.xdef.XDFactory;
import org.xdef.XDPool;
import org.w3c.dom.Element;

public class Order3 {

        public static void main(String... args) throws Exception {
                // Create an instance of the XDDocument object (from XDPool)
                // (external method "err" called from the X-script see below)
                XDPool xpool = XDFactory.compileXD(null, "src/task1/Order3.xdef");

                // Create an instance of the XDDocument object (from XDPool)
                XDDocument xdoc = xpool.createXDDocument("Order");

                // Prepare the error reporter
                ArrayReporter reporter = new ArrayReporter();

                // Prepare the XML element used to record errors
                Element errors =
                        KXmlUtils.newDocument(null, "Errors", null).getDocumentElement();
                xdoc.setUserObject(errors);

                // Run validation mode (you can also try task1/input/Order_err.xml)
                xdoc.xparse("task1/input/Order.xml", reporter);

                // Check errors
                if (errors.getChildNodes().getLength() > 0) {
                        // Write error information to the file
                        KXmlUtils.writeXml("task1/errors/Order_err.xml", errors);
                        System.err.println("Incorrect input data");
                } else {
                        // No errors, write the processed document to the file
                        KXmlUtils.writeXml("task1/output/Order.xml", xdoc.getElement());
                        System.out.println("OK");
                }
        }
}
```

X-definition (the file „src/task1/Order3.xdef“):

```
<xd:def xmlns:xd="http://www.xdef.org/xdef/4.2" name="Order" root="Order">

<xd:declaration>
  external method void task1.Order3ext.err(XXNode, XDValue[]);
</xd:declaration>

<Order
  Number="int; onFalse err(1); onAbsence err(2);"
  CustomerCode="string; onAbsence err(3);">

  <DeliveryPlace xd:script="onAbsence err(11);">
    <Address Street="string(2,100); onFalse err(12); onAbsence err(13);"
      House="int(1,9999); onFalse err(14); onAbsence err(15);"
      City="string(2,100); onFalse err(16); onAbsence err(17);"
      ZIP="num(5); onFalse err(18); onAbsence err(19);"/>
  </DeliveryPlace>

  <Item xd:script="occurs 1..10; onAbsence err(21); onExcess err(22)"
    ProductCode="num(4); onFalse err(23); onAbsence err(24)"
    Quantity="int(1,1000); onFalse err(25); onAbsence err(26)"/>

</Order>
```

```
</xd:def>
```

Java class with the external method "err" (the file „src/task1/Order3_ext.java“):

```
package task1;

import org.xdef.sys.SPosition;
import org.xdef.XDValue;
import org.xdef.proc.XXNode;
import org.w3c.dom.Document;
import org.w3c.dom.Element;

public class Order3ext {

        /** Add error item. */
        public static void err(XXNode xnode, XDValue[] params) {
                Document doc = ((Element) xnode.getUserObject()).getOwnerDocument();
                Element newElem = doc.createElement("Error");
                newElem.setAttribute("ErrorCode", params[0].toString());
                Element root = xnode.getElement().getOwnerDocument().getDocumentElement();
                newElem.setAttribute("Customer", root.getAttribute("CustomerCode"));
                SPosition pos = xnode.getSPosition();
                newElem.setAttribute("Line", String.valueOf(pos.getLineNumber()));
                newElem.setAttribute("Column", String.valueOf(pos.getColumnNumber()));
                doc.getDocumentElement().appendChild(newElem);
                xnode.clearTemporaryReporter(); // remove the error
        }
}
```

4.2 Task 2

This task is a modification of the previous one, it does not process a single order, but a file with many orders. The number of order items is not limited and may not even fit into the computer memory.

The input file with orders „task2/input/Orders..xml“:

```
<Orders id = "123456789">

<Order Number="123" CustomerCode="ALFA">
  <DeliveryPlace>
    <Address Street="LIBERECKA" House="5" City="LIBEREC" ZIP="32321"/>
  </DeliveryPlace>
  <Item ProductCode="0002" Quantity="2"/>
  <Item ProductCode="0003" Quantity="1"/>
</Order>

<Order Number="456" CustomerCode="BETA">
  <DeliveryPlace>
    <Address Street="NA SLUPI" House="9" City="PRAHA" ZIP="11000"/>
  </DeliveryPlace>
  <Item ProductCode="0019" Quantity="50"/>
</Order>

</Orders>
```

4.2.1 Variant 1: Using the "forget" command

We will use the "forget" command to solve this task. This command can be given in the X-script of an element and orders the release of an element from memory after it has been processed. In our task, however, we must ensure that the element is written to the output file before release. X-definition can handle this situation - we will connect the data stream to the XDDocument by using the method "setStreamWriter". This stream enables us to write the processed objects (before it is released from memory). In the first variant, the X-definition processor writes the error log.

X-definition (“src/task2/Orders1.xdef”):

```
<xd:def xmlns:xd="http://www.xdef.org/xdef/4.2" name="Orders" root="Orders">

<Orders id="num(9)">
  <Order xd:script="occurs +; forget"
    Number="int"
    CustomerCode="string(1,20)">
    <DeliveryPlace>
      <Address Street="string(2,100)"
        House="int(1,9999)"
        City="string(2,100)"
```

```
        ZIP="num(5)"/>
    </DeliveryPlace>

    <Item xd:script="occurs 1..10" ProductCode="num(4)" Quantity="int(1,1000)"/>
  </Order>
</Orders>

</xd:def>
```

The continuous writing of the processed document into the output file is managed by the settings in the "setStreamWriter" procedure. X-definition in this case continuously writes the processed elements to this output stream. With the "forget" command in the X-script, we tell the X-definition to release this element from memory at the end of the element processing. The error reports are written directly to the file (this is why this time we use the FileReportWriter class instead of the ArrayReporter class we used in the previous examples).

Program „src/task2/Orders1.java“:

```
package task2;

import org.xdef.sys.FileReportWriter;
import org.xdef.XDDocument;
import org.xdef.XDFactory;
import org.xdef.XDPool;
import java.io.File;
import java.io.FileOutputStream;
import java.io.OutputStream;

public class Orders1 {
        public static void main(String... args) throws Exception {
                // Compile the X-definition source to the XDPool object
                XDPool xpool = XDFactory.compileXD(null, "src/task2/Orders1.xdef");

                // Create an instance of the XDDocument object (from XDPool)
                XDDocument xdoc = xpool.createXDDocument("Orders");

                // The file where to write the result
                OutputStream out = new FileOutputStream("task2/output/Orders.xml");
                xdoc.setStreamWriter(out, "UTF-8", true);

                // The file with errors
                File errors = new File("task2/errors/Orders_err.txt");

                // Prepare the error reporter
                FileReportWriter reporter = new FileReportWriter(errors);

                // Run validation mode (you can also try task2/input/Order_err.xml)
                xdoc.xparse("task2/input/Orders.xml", reporter);

                // close the output stream.
                out.close();
                reporter.close();

                // Check if an error was reported
                if (reporter.errorWarnings()) {
                        System.err.println("Incorrect input data");
                } else {
                        System.out.println("OK");
                }
        }
}
```

4.2.2 Variant 2: writing the correct orders and the wrong orders separately to the files

In this variant we will show the possibility to divide the output of the processed items into two files: in one we will write only correct orders and in the other, we will write the incorrect ones. We will also show you a more complicated use of external methods and a connected user object.

In the X-definition, the "writeResult" method is called in the X-script of the "Order" element in the "finally" section. Instead of setting the automatic output of the processed input items into the output file, we will write the processed elements using this method. The errors will be processed by the "err" method. Finally, we have to close the streams by the "closeAll" method, which closes the output files in case they are not empty.

In the declaration section, we declare three counters with the initial value "0". They will store the number of errors and the number of correct orders. The "errCount" counter counts errors, and the error number is then stored in the "errCountOld" counter so that we can see if the new error has been found in the processed order

(and therefore not written to the "output" stream). In the "count" counter we can see the number of the processed correct order items.

Both, the "output" and "error" streams are written using the "XmlOutStream" objects that have implemented resources for the continuous writing of XML objects. The element can be written as a whole by the „writeElement“ method. The second option is to divide the writing of the element header by the "writeElementStart" method, then to write the child nodes of the element by the "writeElement" or "writeText" methods, and finally to terminate the writing by the "writeElementEnd" method. The „XMLOutStream“ class is set to create the output file in the first writing only.

First, we need to write the root element header by the "writingElementStart" method, and at the end of the process called the "writeElementEnd" method. The individual error items are written to the corresponding data streams by the "writeElement" method.

Note that we have included the text of the declaration section in the CDATA section so that we can freely use the "<", "&" characters anywhere in the code, without the need for alternate entities for these operators.

File names are passed to X-definition by the program using external variables "outFile" and "errFile".

X-definition (“src/task2/Orders2.xdef”):

```
<xd:def xmlns:xd="http://www.xdef.org/xdef/4.2" name="Orders" root="Orders">

<xd:declaration>
<![CDATA[
  /* File names of files with customers and commodity codes. */
  external String outFile, errFile;
  int errCount = 0, errCountOld = 0, count = 0;
  XmlOutStream outStream, errStream;

  /* write file with errors */
  void err(int code) {
    if (errCount++ == 0) {
      errStream = new XmlOutStream(errFile);
          /* Document header and start of the root element */
      errStream.writeElementStart(new Element("Errors"));
    }
    Element e = new Element('Error'); /* prepare element. */
    /* Set attributes. */
    e.setAttribute("Customer", xpath("/Orders/Order/@CustomerCode").toString());
    e.setAttribute("ErrorCode", toString(code));
    e.setAttribute("Line", toString(getSourceLine()));
    e.setAttribute("Column", toString(getSourceColumn()));
    errStream.writeElement(e); /* write the element with error information */
  }

  /* Write result */
  void writeResult() {
    if (errCount != errCountOld) { /* error in order */
      errCountOld = errCount; /* save the number of errors, to know if a new error occurred */
      return; /* write nothing */
    }
    if (count++ == 0) {
      /* this is the first item*/
      outStream = new XmlOutStream(outFile, "windows-1250"); /* prepare data stream */
      outStream.writeElementStart(getRootElement()); /* write document header and root element */
    }
    outStream.writeElement(getElement()); /* write the processed order */
  }

  /* Close the output stream and the file with errors */
  void closeAll() {
    if (errCount > 0) { /* check if errors were reported */
      errStream.writeElementEnd(); /* write root element end tag */
      errStream.close(); /* close the data stream */
    }
    if (count > 0) { /* Check if a correct order exists */
      outStream.writeElementEnd(); /* write root element end tag */
      outStream.close(); /* close the data stream */
    }
  }
]]>
</xd:declaration>

<Orders id="num(9); onFalse err(98)"
    xd:script="finally closeAll();">
  <Order xd:script = "occurs +; onAbsence err(99); finally writeResult()"
    Number="int; onFalse err(1); onAbsence err(2);"
    CustomerCode="string; onAbsence err(3);">
```

```
    <DeliveryPlace xd:script="onAbsence err(11);">
      <Address Street="string(2,100); onFalse err(12); onAbsence err(13);"
        House="int(1,9999); onFalse err(14); onAbsence err(15);"
        City="string(2,100); onFalse err(16); onAbsence err(17);"
        ZIP="num(5); onFalse err(18); onAbsence err(19);"/>
    </DeliveryPlace>
    <Item xd:script="occurs 1..10; onAbsence err(21); onExcess err(22)"
      ProductCode="num(4); onFalse err(23); onAbsence err(24)"
      Quantity="int(1,1000); onFalse err(25); onAbsence err(26)"/>
  </Order>
</Orders>

</xd:def>
```

Program „src/task2/Orders2.java“:

```
package task2;

import org.xdef.sys.ArrayReporter;
import org.xdef.XDDocument;
import org.xdef.XDFactory;
import org.xdef.XDPool;

public class Orders2 {
        public static void main(String... args) throws Exception {
                // Compile X-definition to XDPool
                XDPool xpool = XDFactory.compileXD(null, "src/task2/Orders2.xdef");

                // Create an instance of the XDDocument object (from XDPool)
                XDDocument xdoc = xpool.createXDDocument("Orders");

                // Set external variables
                xdoc.setVariable("outFile", "task2/output/Orders.xml");
                xdoc.setVariable("errFile", "task2/errors/Orders_err.xml");

                // Prepare the error reporter
                ArrayReporter reporter = new ArrayReporter();

                // Run validation mode (you can also try task2/input/Order_err.xml)
                xdoc.xparse("task2/input/Orders.xml", reporter);

                // Throw an exception if unexpected errors detected
                reporter.checkAndThrowErrors();

                // Check reported errors
                if (xdoc.getVariable("errCount").intValue() != 0) {
                        System.err.println("Incorrect input data");
                } else {
                        System.out.println("OK");
                }
        }
}
```

4.2.3 Variant 3: Using the external Java methods

In this variant, we will again show the possibility to divide the output into two files. In the first one, we will write only the correct orders and in the other, we will write the wrong ones. We will also demonstrate a more complex use of external methods and a connected user object, in which we have programmed the methods from the previous sample using the external methods in Java.

X-definition records the output object and reports errors using methods that are programmed in the external class.

X-definition (“src/task2/Orders3.xdef”):

```
<xd:def xmlns:xd="http://www.xdef.org/xdef/4.2" name="Orders" root="Orders">

<xd:declaration>
  external method void task2.Orders3ext.writeOrder(XXNode);
  external method void task2.Orders3ext.err(XXNode, long);
</xd:declaration>

 <Orders id="num(9); onFalse err(0)">
   <Order xd:script="occurs +; onAbsence err(0); finally writeOrder(); forget"
          Number="int; onFalse err(1); onAbsence err(2);"
          CustomerCode="string; onAbsence err(3);">
     <DeliveryPlace xd:script="onAbsence err(11);">
       <Address Street="string(2,100); onFalse err(12); onAbsence err(13);"
          House="int(1,9999); onFalse err(14); onAbsence err(15);"
          City="string(2,100); onFalse err(16); onAbsence err(17);"
```

```
          ZIP="num(5); onFalse err(18); onAbsence err(19);"/>
     </DeliveryPlace>
     <Item xd:script="occurs 1..10; onAbsence err(21); onExcess err(22)"
          ProductCode="num(4); onFalse err(23); onAbsence err(24)"
          Quantity="int(1,1000); onFalse err(25); onAbsence err(26)"/>
   </Order>
 </Orders>
</xd:def>
```

Instead of setting a continuous write of the processed document into the output file, we will write the processed elements using the "writeOrder" method in the external "Order3ext" class. The external class instance is stored in the "writer" variable and attached using the "setUserObject" method to the XDDocument object. Eventually, we finalize the process by the "closeAll" method, which terminates the writing to both the output file and the error file. We do not prepare the reporter this time and we instead set null as the second parameter of the "xparse" method. Therefore, when an error occurs, which is not processed explicitly by the X-script the program throws a RuntimeException.

If no exception is thrown we close the files (by the command "writer.closeAll ();") which we created and check if there are any errors written by the user object (command "writeErrorsNumber()").

Program „src/task2/Orders3.java“ :

```
package task2;

import org.xdef.XDDocument;
import org.xdef.XDFactory;
import org.xdef.XDPool;

public class Orders3 {
        public static void main(String... args) throws Exception {
                // Compile X-definition to XDPool
                XDPool xpool = XDFactory.compileXD(null, "src/task2/Orders3.xdef");

                // Create an instance of the XDDocument object (from XDPool)
                XDDocument xdoc = xpool.createXDDocument("Orders");

                // Create an instance of Orders3ext
                Orders3ext writer = new Orders3ext("task2/output/Orders.xml",
                        "task2/errors/Orders_err.xml");
                xdoc.setUserObject(writer);

                // Run validation mode (you can also try task2/input/Order_err.xml)
                xdoc.xparse("task2/input/Orders.xml", null);

                // Close all streams
                writer.closeAll();

                if (writer.errNum() != 0) {
                        System.err.println("Incorrect input data");
                } else {
                        System.out.println("OK");
                }
        }
}
```

The external class for continuous writing ensures that the correct order is written to the recorder in the "_outputWriter" variable and the error information in the variable "_errorWriter". The writer constructor is created when the first order or error information is written. When creating a recorder, you must first write to the stream the XML header and the start tag of the root element (element name and its attributes) using the "writeElementStart" method. The individual child elements are written using the "writeNode" method. In the end, we still have to write the root element's end tag by the "writeElementEnd" method (see the "closeAll" method in the Java source code.

Java source of the class with external methods („src/task2/Orders3ext.java“):

```
package task2;

import org.xdef.sys.SPosition;
import org.xdef.xml.KXmlUtils;
import org.xdef.xml.KXpathExpr;
import org.xdef.XDFactory;
import org.xdef.XDXmlOutStream;
import org.xdef.proc.XXNode;
import java.io.IOException;
import javax.xml.xpath.XPathConstants;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
```

```
public class Orders3ext {

        private final String _outputFile;       // Output file
        private final String _errorFile;        // Error file
        private XDXmlOutStream _outputWriter;   // writer for output
        private XDXmlOutStream _errorWriter;    // writer for errors
        private Document _errorDoc;             // XML document with errors
        private Document _objDoc;               // XML document with orders
        private final KXpathExpr _xpath;        // prepared xpath expression
        private int _errCount;                  // error counter
        private int _errCountOld;               // previous value of the error counter
        private int _count;                     // counter of correct orders

        // Create an instance of this class.
        public Orders3ext(String outputFile, String errorFile) throws IOException {
                _outputFile = outputFile; // output file name.
                _errorFile = errorFile; // error file name.
                // writers will be created when an item to be written occurs
                _errorWriter = _outputWriter = null;
                // Prepare XPath expression to get customer code from an order
                // Because of the command "forget" it will be in the processed document
                // only one (the processed) order.
                // This XPath expression will be executed when an error item is generated.
                _xpath = new KXpathExpr("/Orders/Order[1]/@KodZakaznika");
                // Clear counters
                _errCount = _errCountOld = _count = 0;
        }

        // Write the order (only if no error was reported
        public static void writeOrder(XXNode xnode) {
                // Get "User object" (i.e. the instance of this class).
                Orders3ext x = (Orders3ext) xnode.getUserObject();
                if (x._errCount != x._errCountOld) { // an error was reported?
                        // set the old error counter (i.e. no errors reported for the next item)
                        x._errCountOld = x._errCount;
                } else {
                        // No error reported, so write the order of the result.
                        Element el = xnode.getElement();
                        if (x._count == 0) { // check if nothing was written yet
                                // Create a writer and insert the XML header and the root element
                                try {
                                        x._outputWriter =
                                                XDFactory.createXDXmlOutStream(x._outputFile,
                                                "windows-1250", true);
                                } catch (IOException ex) {
                                        throw new RuntimeException(ex.getMessage());
                                }
                                x._objDoc = el.getOwnerDocument();
                                x._outputWriter.writeElementStart(x._objDoc.getDocumentElement());
                        }
                        x._count++; // increase the counter of correct orders.
                        // write the processed order
                        x._outputWriter.writeNode(el);
                }
        }

        // Create the writeOrder and set the variable "_error".
        public static void err(XXNode xnode, long code) {
                // Get "User object" (i.e. the instance of this class).
                Orders3ext x = (Orders3ext) xnode.getUserObject();
                // Create the XML writer for errors (if it was not created yet)
                if (x._errCount == 0) {
                        try {
                                x._errorWriter = XDFactory.createXDXmlOutStream(x._errorFile,
                                        "windows-1250", true);
                        } catch (IOException ex) {
                                throw new RuntimeException(ex.getMessage());
                        }
                        x._errorDoc = KXmlUtils.newDocument(null,"Errors",null);
                        // write the XML header and the root element.
                        x._errorWriter.writeElementStart(x._errorDoc.getDocumentElement());
                }
                x._errCount++; // increase error counter
                // Create the element to be written.
                Element el = x._errorDoc.createElement("Error");
                el.setAttribute("ErrorCode", String.valueOf(code));
                String customer =
                        (String) x._xpath.evaluate(xnode.getElement(), XPathConstants.STRING);
                el.setAttribute("Customer", customer);
                SPosition pos = xnode.getSPosition();
                el.setAttribute("Line", String.valueOf(pos.getLineNumber()));
                el.setAttribute("Column", String.valueOf(pos.getColumnNumber()));
```

```
                x._errorWriter.writeNode(el);
        }

        // Get the number of errors
        public int errNum() {return _errCount;}

        // Close created files
        public void closeAll() {
                // close result output stream (if something was written)
                if (_outputWriter != null) {
                        _outputWriter.closeStream(); // write root end tag and close the stream
                }
                // close error output stream (if something was written)
                if (_errorWriter != null) {
                        _errorWriter.closeStream(); // write root end tag and close the stream
                }
        }
}
```

4.3 Task 3

Now, let's check the correctness of the customer and commodity code in the input data against data stored in external XML files. The input data is the same as in the first example.

Input order „task3/input/Order.xml“:

```
<Order Number="123" CustomerCode="ALFA">
  <DeliveryPlace>
    <Address Street="LIBERECKÁ" House="5" City="LIBEREC" ZIP="32321"/>
  </DeliveryPlace>
  <Item ProductCode="0002" Quantity="2"/>
  <Item ProductCode="0003" Quantity="1"/>
</Order>
```

The file with customers „task3/input/Customers..xml“:

```
<Customers>
  <Customer CustomerCode="ALFA">
     <Company Name="ALFA S.R.O." ID="1234578"/>
     <Address Street="KLADENSKÁ" House="5" City="KLADNO" ZIP="54321"/>
  </Customer>
  <Customer CustomerCode="BETA">
     <Company Name="BETA A.S" ID="1234580"/>
     <Address Street="MĚLNICKÁ" House="5" City="MĚLNÍK" ZIP="45321"/>
  </Customer>
</Customers>
```

The file with products „task3/input/Products..xml“:

```
<Products>
  <Product Code="0001" Name="bicycle" Price="150.50"/>
  <Product Code="0002" Name="motorcycle" Price="2340.00"/>
  <Product Code="0003" Name="car" Price="7777.00"/>
</Products>
```

4.3.1 Variant 1: automatic generation of error code

In the first variant, as usual, we demonstrate the simplest way to generate the error code automatically. The XML data with customer codes and product numbers are passed through external variables. We get the customer code and product code in the script using the "xpath" method used in the validation methods "customerCode" and "productCode" (both declared in the declaration section). Note that the result of the method "xpath" is a Container. On Container objects, the "getLength" method returns the number of sequential elements in the object. We expect that if the value is correct, an appropriate element is found in the xpath argument. The value we look for using the xpath expression is obtained by the "getText" method, which returns a string with the value of the currently processed attribute or the text node. We validate the value of the code with the declared methods. If the result is "true," the value is correct and if it is "false" the code value is incorrect. Note that the method "error" used in validation methods puts the error message to the report file and returns the boolean value "false".

X-definition „src/task3/Order1.xdef“:

```
<xd:def xmlns:xd="http://www.xdef.org/xdef/4.2" name="Order" root="Order">

<xd:declaration>
  /* Information about the customers and products. */
  external Element customers, products;
```

```
  /* Check if the customer exists in the customer list. */
  boolean customerCode() {
    if (xpath('Customer[@CustomerCode=&quot;' + getText() + '&quot;]',
                  customers).getLength()!=0) return true;
    return error("Customer " + getText() + " not exists");
  }
  /* Check if the product exists in the products list. */
  boolean productCode() {
    if (xpath('Product[@Code=&quot;' + getText() + '&quot;]',
                products).getLength() != 0) return true;
    return error("Product " + getText() + " not exists");
  }
</xd:declaration>

<Order Number="int"
  CustomerCode="customerCode()" >

  <DeliveryPlace>
    <Address Street = "string(2,100)"
      House = "int(1,9999)"
      City = "string(2,100)"
      ZIP = "num(5)"/>
  </DeliveryPlace>

  <Item xd:script = "occurs 1..10"
    ProductCode = "productCode()"
    Quantity = "int(1,1000)"/>
</Order>

</xd:def>
```

The Java program is virtually the same as in the first example, we just had to add the external variables "products" and "customers" to the XDDocument. These variables are of the Element type. Note that the "setVariable" method in Java code automatically converts a file into an Element.

Program „src/task3/Order1.java“ :

```
package task3;

import org.xdef.sys.ArrayReporter;
import org.xdef.xml.KXmlUtils;
import org.xdef.XDDocument;
import org.xdef.XDFactory;
import org.xdef.XDPool;
import java.io.PrintStream;

public class Order1 {
        public static void main(String... args) throws Exception {
                // Compile the X-definition source to the XDPool object
                XDPool xpool = XDFactory.compileXD(null, "src/task3/Order1.xdef");

                // Create an instance of the XDDocument object (from XDPool)
                XDDocument xdoc = xpool.createXDDocument("Order");

                // set variables "products" and "customers"
                xdoc.setVariable("products", "task3/input/Products.xml");
                xdoc.setVariable("customers", "task3/input/Customers.xml");

                // Prepare error reporter
                ArrayReporter reporter = new ArrayReporter();

                // Run validation mode (you can also try task3/input/Order_err.xml)
                xdoc.xparse("task3/input/Order.xml", reporter);

                // Check if an error was reported
                if (reporter.errorWarnings()) {
                        // Print errors to the file
                        PrintStream ps = new PrintStream("task3/errors/Order_err.txt");
                        reporter.printReports(ps);
                        ps.close();
                        System.err.println("Incorrect input data");
                } else {
                        // write processed document
                        KXmlUtils.writeXml("task3/output/Order_123.xml", xdoc.getElement());
                        System.out.println("OK");
                }
        }
}
```

The result is similar to the first example.

4.3.2 Variant 2: declaration of the validation method

In this variant we make a change: we do the type checking in the validation method. The declared type is a method that - in our case - results in a boolean value. We will show that the error message can be processed using the "error" method, which writes the error message into the reporter and returns the Boolean value "false". The Java program will be the same as in the previous variant, so let's start with the X-definition:

X-definition „src/task3/Order2..xdef“:

```
<xd:def xmlns:xd="http://www.xdef.org/xdef/4.2" name="Order" root="Order">

<xd:declaration>
  /* Information about the customers and products. */
  external Element customers, products;

  boolean customer() {
    String s = getText();
    Container c = xpath('Customer[@CustomerCode="' + s + '"]', customers);
    return c.getLength()==0 ? error("Incorrect customer code: " + s) : true;
  }

  boolean item() {
    String s = getText();
    Container c = xpath('Product[@Code="' + s + '"]', products);
    return c.getLength()==0 ? error("Incorrect item code: " + s) : true;
  }
</xd:declaration>

<Order Number="int" CustomerCode= "customer()">

  <DeliveryPlace>
    <Address Street="string(2,100)"
      House="int(1,9999)"
      City="string(2,100)"
      ZIP="num(5)"/>
  </DeliveryPlace>

  <Item xd:script = "occurs 1..10"
    ProductCode = "item()"
    Quantity = "int(1,1000)"/>
</Order>

</xd:def>
```

In the event of an error, the error message log will contain information with the text in the "error" method:

The file with the error information is saved in „task3/errors/Order_123err.txt“ :

```
Incorrect customer code: AGFA; line=2; column=35;
source='file:/D:/cvs/DEV/java/examples/task3/input/Order_err.xml'; pos=82; xpath=/Order/@CustomerCode;
X-position=Order#Order
E XDEF515: Value error; line=2; column=35;
source="file:/D:/cvs/DEV/java/examples/task3/input/Order_err.xml"; xpath=/Order/@CustomerCode;
X-position=Order#Order/@CustomerCode;
```

4.3.3 Variant 3: use a file with error message texts in language mutations

This variant is the same as the previous one, but we will show the option of attaching error message texts. The advantage is that these texts and their language mutations can be provided separately by the program. The method "err" uses the X-definition reporter, and the report models can be passed to the program as a file with properties (see the Java code). In the "err" method, we must, in this case, use a message identifier as the first parameter, the second parameter may be the default message text or an empty string and the third parameter may be the modification string with parameters (if required). If the message identifier is found, the content of the second parameter is replaced by the text from the found report. Otherwise, it will use this parameter to build the report. If we know the message exists, the second parameter may be an empty string or null.

X-definition „src/task3/Order2a.xdef“:

```
<xd:def xmlns:xd="http://www.xdef.org/xdef/4.2" name="Order" root="Order">

<xd:declaration>
<![CDATA[
  /* Information about the customers and products. */
  external Element customers, products;

  /* Validation of customer code. */
  boolean customer() {
```

```
    String s = getText();/* get the value of the attribute. */
    /* find customer description and save it to Container. */
    Container c = xpath('Customer[@CustomerCode="' + s + '"]', customers);
    /* Check if the customer was found. If yes, return true; otherwise
     * call the method error, which writes the error report and returns false. */
    return c.getLength()==0
        ? error("POBJ001", "Customer code: &{0}", "&{0}" + s) : true;
  }

  /* Validate product code from Item. */
  boolean item() {
    String s = getText(); /* get attribute value. */
    /* Find product description and save it to Container. */
    Container c = xpath('Product[@Code="' + s + '"]', products);
    /* Check if the code was found. If yes, return true; otherwise
     * call the method error, which writes the error report and returns false. */
    return c.getLength()==0
        ? error("POBJ002", "Product code: &{0}", "&{0}" + s) : true;
  }
]]>
</xd:declaration>

<Order Number="int" CustomerCode= "customer()">

  <DeliveryPlace>
    <Address Street="string(2,100)"
      House="int(1,9999)"
      City="string(2,100)"
      ZIP="num(5)"/>
  </DeliveryPlace>

  <Item xd:script = "occurs 1..10"
    ProductCode = "item()"
    Quantity = "int(1,1000)"/>
</Order>

</xd:def>
```

Note that in the message texts the "&{#SYS000}" link is added, which allows you to write the rows, the column, the name of the source file, and the XPath position, in the object currently being processed (by the parameters "&{line}", "&{column}", "&{source}" etc). Properties "_prefix" and "_language" are obligatory and specify the prefix of messages and the ISO name of the language in the message file. The file must be saved in the UTF-8 character code.

Reports in the Czech language „src/task3/Order_ces.properties“:

```
# Prefix of messages.
_prefix=POBJ

# ISO name of the language.
_language=ces

# ISO name of the default language.
_defaultLanguage=eng

# *************** Messages: ***************
POBJ001=Chybný kód zákazníka: "&{0}"&{#SYS000}
POBJ002=Chybné číslo položky: "&{0}"&{#SYS000}
POBJ003=Chyba ve vstupních datech
POBJ004=Vstupní data jsou zapsána
```

Reports in the English language „src/task3/Order_eng.properties“:

```
# Prefix of messages.
_prefix=POBJ

# ISO name of the language.
_language=eng

# ISO name of the default language.
_defaultLanguage=eng

# *************** Messages: ***************
POBJ001=Incorrect customer code: "&{0}"&{#SYS000}
POBJ002=Invalid item identifier: "&{0}"&{#SYS000}
POBJ003=Input data error
POBJ004=Input data saved
```

In the Java program, the first set by the "setProperty(...)" method to access the files with the language message mutations. We can also set the language in which the messages are displayed (if the language is not set, the

system setting is used. If the language (or the local language message identifier) is not found in the tables, the default language (i.e. English) is used. If even no such message is found, the default text may be specified by the second parameter of the "error" method). We will show that we can use also the Java code as a reporting system (see the method "Report.error" and "Report.info").

Program „src/task3/Order2a.java“ :

```
package task3;

import org.xdef.sys.ArrayReporter;
import org.xdef.sys.Report;
import org.xdef.xml.KXmlUtils;
import org.xdef.XDConstants;
import org.xdef.XDDocument;
import org.xdef.XDFactory;
import org.xdef.XDPool;
import java.io.PrintStream;

public class Order2a {
        public static void main(String[] args) throws Exception {
                // Compile the X-definition source to the XDPool object
                XDPool xpool = XDFactory.compileXD(null, "src/task3/Order2a.xdef");

                // Create an instance of the XDDocument object (from XDPool)
                XDDocument xdoc = xpool.createXDDocument("Order");

                // Set files with reports
                xdoc.setProperty(XDConstants.XDPROPERTY_MESSAGES + "POBJ",
                        "src/task3/POBJ*.properties");

                // Set the actual language for the reporter (you can also try to set "ces")
                xdoc.setProperty(XDConstants.XDPROPERTY_MSGLANGUAGE, "eng"); // English

                // Set external variables "products" and "customers"
                xdoc.setVariable("products", "task3/input/Products.xml");
                xdoc.setVariable("customers", "task3/input/Customers.xml");

                // Prepare error reporter
                ArrayReporter reporter = new ArrayReporter();

                // Run the validation mode (you can also try task3/input/Order_err.xml)
                xdoc.xparse("task3/input/Order.xml", reporter);

                // Check errors
                if (reporter.errorWarnings()) {
                        // print errors to a file
                        PrintStream ps = new PrintStream("task3/errors/Order_err.txt");
                        reporter.printReports(ps); //print errors
                        ps.close();
                        // print the message to the system console
                        Report rep = Report.error("POBJ003", null);
                        System.err.println(rep.toString());
                } else {
                        // Write the processed document to the file
                        KXmlUtils.writeXml("task3/output/Order.xml", xdoc.getElement());
                        // print the message to the system console
                        Report rep = Report.info("POBJ004", null);
                        System.out.println(rep.toString());
                }
        }
}
```

In the event of an error, the error message log will contain information with the text in the "error" method (the language is set to "eng" - English):

The output of the incorrect item „task3/errors/Order_err.txt“ in English:

```
E POBJ001: Incorrect customer code: 'AGFA'; line=2; column=35;
source="file:/D:/cvs/DEV/java/examples/task3/input/Order_err.xml"; xpath=/Order/@CustomerCode;     X-
position=Order#Order
```

To try generating an error file using methods, let's do the following exercise.

4.4 Task 4

In this task, we will re-validate the data, and add some information to the resulting document. Again, we will work with an input file of virtually unlimited size We will add a data check as in the previous example, but the task will be somewhat more complicated. The customer address element is not part of the input file this time, but we will

add it from the data stored in the customer information file. Besides, we'll add item information which we calculate as a multiple of the number of items and the cost per item we get from the item information data. Thus, the attribute "Price" will increase in item elements. We only write the correct orders to the output file and the incorrect ones to the error file. For simplicity, we assume the formal accuracy of the input data (all required data is in the correct format). We leave the error file for simplicity as text. This time the input data will not contain the "DeliveryPlace" element.

Input data with orders: „task4/input/Orders.xml“:

```
<Orders id="123456789">
  <Order Number="123" CustomerCode="ALFA">
    <Item ProductCode="0002" Quantity="2"/>
    <Item ProductCode="0003" Quantity="1"/>
  </Order>
  <Order Number="124" CustomerCode="BETA">
    <Item ProductCode="0001" Quantity="5"/>
  </Order>
</Orders>
```

4.4.1 Variant 1: no external Java methods

In the X-definition code, we add the missing element "DeliveryPlace" and add the "Price" attribute to the order items. We will describe the "Order" input data model, and because the output form of an order differs from the input data form, we describe the output order as a separate model that will be used to construct the result. XML data with a list of valid customers and items will be passed to the respective external variables "products" and "customers" by the program. The program also passes the output channel for writing the result to the variable "output".

*X-definition **„src/task4/Orders1.xdef“:***

```
<xd:def xmlns:xd="http://www.xdef.org/xdef/4.2" name="Orders" root="Orders">

<xd:declaration>
  external Element products, /* information about products. */
        customers; /* information about customers. */
  external XmlOutStream output; /* Output stream (set by the external program). */

  Container c; /* information about customer created by the method "customer". */
  int errors = 0, errorsOld = 0, errorsOld1 = 0, count = 0; /* counters */

  /* Check customer code */
  boolean customer() {
    String s = getText(); /* get attribute value. */
    c = xpath('Customer[@CustomerCode="' + s + '"]', customers); /* Find the customer. */
    /* Check if the customer found */
    if (c.getLength()==0) {
      /* Customer not found, increase error counter and report an error. */
      errors++;
      return error("Incorrect customer code: " + s); /* sets error message and returns false. */
    }
    return true; /* Customer found, OK */
  }

  /* Check the item code */
  boolean item() {
      String s = getText(); /* get attribute value. */
      /* Find the description of the item. */
      Container c = xpath('Product[@Code="' + s + '"]', products);
      if (c.getLength()==0) { /* Item found? */
        /* Item not found, increase the error counter and report the error. */
        errors++;
        return error("Incorrect item number: " + s);
      }
      return true; /* Item was found, OK */
  }

  /* Write order. */
  void writeObj() {
    if (errors != errorsOld || errorsOld1 != errors()) {
      /* An error occurred */
      errorsOld = errors; /* save the counter to "errorsOld"; write nothing. */
      errorsOld1 = errors();
    } else {
      /* Is it the first record? */
      if (count++ == 0) {
         /* nothing was written yet, write XML header and root element. */
         output.setIndenting(true); /* set output indentation. */
         output.writeElementStart(getRootElement());
```

```
      }
      /* Create an object from the context with the actual order and write it. */
      output.writeElement(xcreate('Order', getElement()));
    }
  }

  /* Close output. */
  void closeAll() {
    /* Something was written?. */
    if (count != 0) {
      /* yes, close output. */
      output.close();
    }
  }

  /* Calculate the total cost of the item of an order. */
  float price() {
    /* Get quantity */
    int number = parseInt(xpath("@Quantity")); /* Get quantity from the input data. */
    String code = from("@ProductCode"); /* Get commodity code from the input data. */
    /* Find the product (we already know it exists - see customer()). */
    Element el1 = xpath('Product[@Code="' + code + '"]', products).getElement(0);
    float price = parseFloat(el1.getAttribute("Price")); /* get price of one product. */
    return price * number; /* Compute total cost and return it. */
  }
  uniqueSet checkObjId int();
</xd:declaration>

<Orders xd:script="finally output.close();" id="string(9)">
  <Order xd:script="occurs +; finally writeObj(); forget"
      Number="checkObjId.ID()"
      CustomerCode="customer()">
    <Item xd:script="occurs 1..100;" ProductCode="item()" Quantity="int"/>
  </Order>
</Orders>

<Order Number="string" CustomerCode="string">
  <DeliveryPlace xd:script="create c" >
    <Address Street="string(2,100)"
      House="int(1,9999)"
      City="string(2,100)"
      ZIP="num(5)"/>
  </DeliveryPlace>
  <Item xd:script="occurs 1..100;"
    ProductCode="string();"
    Quantity="int();"
    Price="float; create price();" />
</Order>

</xd:def>
```

The program to run the task is similar to the previous case.

Program „src/task4/Orders1.java“ :

```
package task4;

import org.xdef.sys.ArrayReporter;
import org.xdef.XDDocument;
import org.xdef.XDFactory;
import org.xdef.XDPool;
import java.io.PrintStream;
import java.util.Properties;

public class Orders1 {
        public static void main(String... args) throws Exception {
                // compile XDPool from the X-definition source
                Properties props = new Properties();
                XDPool xpool = XDFactory.compileXD(props, "src/task4/Orders1.xdef");

                // Create an instance of the XDDocument object (from XDPool)
                XDDocument xdoc = xpool.createXDDocument("Orders");

                // set variables "products", "customers" and "output"
                xdoc.setVariable("products", "task4/input/Products.xml");
                xdoc.setVariable("customers", "task4/input/Customers.xml");
                xdoc.setVariable("output",
                        XDFactory.createXDXmlOutStream("task4/output/Orders.xml", "UTF-8", true));

                // prepare the error reporter
                ArrayReporter reporter = new ArrayReporter();

                // run validation mode (you can also try task4/input/Order_err.xml)
                xdoc.setProperties(props);
                xdoc.xparse("task4/input/Orders.xml", reporter);
```

```
                // check errors
                if (reporter.errorWarnings()) {
                        // write log file with errors
                        PrintStream ps = new PrintStream("task4/errors/Orders_err.txt");
                        reporter.printReports(ps); //print errors
                        ps.close();
                        System.err.println("Incorrect input data");
                } else {
                        System.out.println("OK");
                }
        }
}
```

The output of the program with the added address and price will be stored in the file "task4/output/Orders.xml":

```
<Orders id="123456789">
  <Order CustomerCode="ALFA"
    Number="123">
    <DeliveryPlace>
      <Address City="KLADNO"
        House="5"
        Street="KLADENSKÁ"
        ZIP="54321"/>
    </DeliveryPlace>
    <Item Price="4680.0"
      ProductCode="0002"
      Quantity="2"/>
    <Item Price="3455.0"
      ProductCode="0003"
      Quantity="1"/>
  </Order>
  <Order CustomerCode="BETA"
    Number="124">
    <DeliveryPlace>
      <Address City="MĚLNÍK"
        House="5"
        Street="MĚLNICKÁ"
        ZIP="45321"/>
    </DeliveryPlace>
    <Item Price="602.5"
      ProductCode="0001"
      Quantity="5"/>
  </Order>
</Orders>
```

4.4.2 Variant 2: with external Java methods

X-definition „src/task4/Orders2.xdef“:

```
<xd:def xmlns:xd="http://www.xdef.org/xdef/4.2" name="Orders" root="Orders">

<xd:declaration>
  external method {
    boolean task4.Orders2ext.customer(XXData);
    boolean task4.Orders2ext.item(XXData);
    String task4.Orders2ext.price(XXNode);
    void task4.Orders2ext.closeAll(XXNode);
    void task4.Orders2ext.writeObj(XXNode);
  }
  Element address;
  uniqueSet checkObjId int();
</xd:declaration>

<Orders xd:script="finally closeAll()" id="num(9)">
  <Order xd:script="occurs +; finally writeObj(); forget"
      Number="checkObjId.ID()"
      CustomerCode="customer()">
    <Item xd:script="occurs 1..100;"
      ProductCode="item()"
      Quantity="int"/>
   </Order>
</Orders>

<Order Number="int" CustomerCode="string">
  <DeliveryPlace>
    <Address xd:script="create address" Street="string(2,100)"
      House="int(1,9999)"
      City="string(2,100)"
      ZIP="num(5)"/>
  </DeliveryPlace>
  <Item xd:script="occurs 1..100;"
    ProductCode="string;"
    Price="float; create price()"
    Quantity="int"/>
```

```
</Order>

</xd:def>
```

Java program „src/task4/Orders2.java“:

```
package task4;

import org.xdef.sys.ArrayReporter;
import  org.xdef.XDDocument;
import  org.xdef.XDFactory;
import  org.xdef.XDPool;
import java.io.PrintStream;

public class Orders2 {
        public static void main(String[] args) throws Exception {
                // compile the XDPool from the X-definition source
                XDPool xpool = XDFactory.compileXD(null, "src/task4/Orders2.xdef");

                // Create an instance of the XDDocument object (from XDPool)
                XDDocument xdoc = xpool.createXDDocument("Orders");

                // set the instance of Orders2ext as user object
                xdoc.setUserObject(new Orders2ext(xpool, // the instance of Orders2ext
                  "task4/input/Products.xml", // file with information about commodity items
                  "task4/input/Customers.xml", // file with the information about customers
                  "task4/output/Orders.xml")); // output file.

                // Prepare the error reporter
                ArrayReporter reporter = new ArrayReporter();

                // run validation mode (you can also try task4/input/Orders_err.xml)
                xdoc.xparse("task4/input/Orders.xml", reporter);

                // Check errors
                if (reporter.errorWarnings()) {
                        // write log file with errors
                        PrintStream ps = new PrintStream("task4/errors/Orders_err.txt");
                        reporter.printReports(ps); //print errors
                        ps.close();
                        System.err.println("Incorrect input data");
                } else {
                        System.out.println("OK");
                }
        }
}
```

Java source code with external methods „src/task4/Orders2ext.java“:

```
package task4;

import org.xdef.sys.ArrayReporter;
import org.xdef.xml.KXmlOutStream;
import org.xdef.xml.KXmlUtils;
import org.xdef.XDDocument;
import org.xdef.XDPool;
import org.xdef.proc.XXData;
import org.xdef.proc.XXNode;
import java.io.IOException;
import javax.xml.xpath.XPath;
import javax.xml.xpath.XPathConstants;
import javax.xml.xpath.XPathExpression;
import javax.xml.xpath.XPathFactory;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;

/** External methods called from Order2.xdef */
public class Orders2ext {

        Element _items, _customers;
        int _errors, _errorsOld, _count;
        KXmlOutStream _output;
        XDPool _xpool;

        private final XPath _xp = XPathFactory.newInstance().newXPath();

        public Orders2ext(XDPool xpool,
                String items,
                String customers,
                String output) throws IOException {
                _items = KXmlUtils.parseXml(items).getDocumentElement();
                _customers = KXmlUtils.parseXml(customers).getDocumentElement();
                _output = new KXmlOutStream(output, "UTF-8", true);
                _xpool = xpool;
```

```
                _errorsOld = _errors = _count = 0;
        }

        public static boolean customer(XXData xdata) {
                /* Get the instance of this class. */
                Orders2ext u = (Orders2ext) xdata.getUserObject();
                String s = xdata.getTextValue(); /* get attribute value. */
                try {
                        /* Find the customer. */
                        XPathExpression xExpr =
                                u._xp.compile("Customer[@CustomerCode='" + s + "']");
                        NodeList nl =
                                (NodeList) xExpr.evaluate(u._customers, XPathConstants.NODESET);
                        if (nl == null || nl.getLength() == 0) {
                                // the customer not found, increase error counter and report error.
                                u._errors++;
                                xdata.error("Incorrect customer code: " + s, null);
                                return false; /* returns false -> incorrect. */
                        }
                        return true; /* Customer found, OK */
                } catch (Exception ex) {
                        u._errors++;
                        xdata.error("Unexpected exception: " + ex, null);
                        return false;
                }
        }

        public static boolean item(XXData xdata) {
                /* Get the instance of this class. */
                Orders2ext u = (Orders2ext) xdata.getUserObject();
                String s = xdata.getTextValue(); /* get attribute value. */
                try {
                        /* Find the description of the Item according to the code. */
                        XPathExpression xExpr = u._xp.compile("Product[@Code='" + s + "']");
                        NodeList nl =
                                (NodeList) xExpr.evaluate(u._items, XPathConstants.NODESET);
                        if (nl == null || nl.getLength() == 0) {
                                // Item not found, increase the error counter and report an error.
                                u._errors++;
                                xdata.error("Incorrect item number: " + s, null);
                                return false;
                        }
                        return true; /* Item was found, OK */
                } catch (Exception ex) {
                        u._errors++;
                        xdata.error("Unexpected exception: " + ex, null);
                        return false;
                }
        }

        public static void writeObj(XXNode xnode) {
                /* Get the instance of this class. */
                Orders2ext u = (Orders2ext) xnode.getUserObject();
                if (u._errors != u._errorsOld) {
                        /* a new error occurs, do not write record */
                        u._errorsOld = u._errors; /* save error counter to "errorsOld" */
                } else {
                        /* Check if this is the first record. */
                        if (u._count++ == 0) {
                                /* first time, so write the root element. */
                                u._output.setIndenting(true); /* set output indentation */
                                u._output.writeElementStart(
                                xnode.getElement().getOwnerDocument().getDocumentElement());
                        }
                        /* Prepare XDDocument for the construction according to model "Order". */
                        XDDocument xdoc = u._xpool.createXDDocument("Orders");
                        xdoc.setUserObject(u);
                        Element el = xnode.getElement();
                        xdoc.setXDContext(el);
                        try {
                                String s = el.getAttribute("CustomerCode");
                                XPathExpression xExpr = u._xp.compile(
                                        "Customer[@CustomerCode='" + s + "']/Address");
                                NodeList nl = (NodeList) xExpr.evaluate(
                                        u._customers, XPathConstants.NODESET);
                                Element adresa = (Element) nl.item(0);
                                xdoc.setVariable("address", adresa);
                                /* Create the object and write it. */
                                ArrayReporter reporter = new ArrayReporter();
                                u._output.writeNode(xdoc.xcreate("Order", reporter));
                        } catch (Exception ex) {
                                // do nothing, an error was already reported when parsed
                        }
```

```
                }
        }

        public static void closeAll(XXNode xnode) {
                /* Get the instance of this class. */
                Orders2ext u = (Orders2ext) xnode.getUserObject();
                /* Check if a record was written. */
                if (u._count != 0) {
                        /* Yes, close the stream. */
                        u._output.writeElementEnd();
                        u._output.closeStream();
                }
        }

        public static String price(XXNode xnode) {
                try {
                        /* Get the instance of this class. */
                        Orders2ext u = (Orders2ext) xnode.getUserObject();
                        Element el = xnode.getXDContext().getElement(); /* Actual context. */
                        String s = el.getAttribute("ProductCode"); /* get commodity code. */
                        /* Find the item description. */
                        XPathExpression xExpr = u._xp.compile("Product[@Code='" + s + "']");
                        /* We already know that it exists. */
                        NodeList nl =
                                (NodeList) xExpr.evaluate(u._items, XPathConstants.NODESET);
                        Element el1 = (Element) nl.item(0);
                        /* get quantity as a number */
                        int count = Integer.parseInt(el.getAttribute("Quantity"));
                        /* compute price */
                        float price = Float.parseFloat(el1.getAttribute("Price")) * count;
                        /* Return it as a string */
                        return String.valueOf(price);
                } catch (Exception ex) {return "-1"; /* this never should happen! */}
        }
}
```

5 JSON/XON /Properties/Windows INI data

In this chapter, we discuss some examples of data in JSON, XON, Properties, or Windows INI format. Our task processes the input data and checks its accuracy. In the examples, we describe again both the source text of the Java program as well as the X-definition. The files, including input data, are available in a Jar file at http://www.xdefinice.cz/en/download.

Note that because JSON format is a subset of XON format, the description of the model or the processed data may be either in the JSON or in the XON format.

5.1 Task5 (JSON)

Let's have the following JSON data *(see „task5/input/jsonExample.json"):*

```
{"date": "2020-02-22",
  "cities": [
    {"from": ["Brussels",
        {"to": "London", "distance": 322},
        {"to": "Paris", "distance": 265}
      ]
    },
    {"from": ["London",
        {"to": "Brussels", "distance": 322},
        {"to": "Paris", "distance": 344}
      ]
    }
  ]
}
```

The model of this JSON data is written in the element xd:xon named "distances". The complete X-definition (see *„src/task5/jsonExample.xdef"):*

```
<xd:def xmlns:xd="http://www.xdef.org/xdef/4.2" name="jsonExample" root="distances">
  <xd:xon xd:name="distances">
    { "date": "date()",
      "cities": [
        {%script = "occurs +",
          "from": [
            "string()",
            {%script = "occurs +",
               "to":       "jstring()",
               "distance": "int(0, 9999)"
            }
          ]
        }
      ]
    }
  </xd:xon>
</xd:def>
```

Note that the model of JSON data contains a JSON text where the description of values is a string with an X-script description. If you want to add an X-script command to maps or arrays you have to add it as a directive %script = "X-script…".

In the Java program for parsing JSON data, we use the method „jparse" instead of "xparse". The result of parsing is an XON object. You can print this object in JSON format using the static method "toJsonString" from the class "org.xdef.xon.XonUtils".

The complete Java program (see *„src/task5/JsonExample.java"):*

```
package task5;

import java.io.File;
import java.io.FileOutputStream;
import java.io.OutputStreamWriter;
import org.xdef.sys.ArrayReporter;
import org.xdef.XDDocument;
import org.xdef.XDFactory;
import org.xdef.XDPool;
import java.io.PrintStream;
import java.io.Writer;
import java.util.Properties;
import org.xdef.xon.XonUtils;
```

```
public class JsonExample {
        public static void main(String... args) throws Exception {
                // compile the XDPool object from the X-definition source
                Properties props = new Properties();
                XDPool xpool = XDFactory.compileXD(props, "src/task5/jsonExample.xdef");
                // Create an instance of the XDDocument object (from XDPool)
                XDDocument xdoc = xpool.createXDDocument("jsonExample");
                // prepare then error reporter
                ArrayReporter reporter = new ArrayReporter();
                // run validation mode (you can also try task5/input/Order_err.xml)
                xdoc.setProperties(props);
                Object xon = xdoc.jparse("task5/input/jsonExample.json", reporter);
                // check errors
                if (reporter.errorWarnings()) {
                        new File("task5/errors").mkdirs();
                        // write log file with errors
                        PrintStream ps = new PrintStream("task5/errors/json.txt");
                        reporter.printReports(ps); //print errors
                        ps.close();
                        System.err.println("Input data error; see task5/errors/json.txt");
                } else {
                        System.out.println("OK. See task5/output/result.json");
                        new File("task5/output").mkdirs();
                        // Store the parsed result
                        Writer out = new OutputStreamWriter(
                                new FileOutputStream("task5/output/result.json"), "UTF-8");
                        out.write(XonUtils.toJsonString(xon, true));
                        out.close();
                }
        }
}
```

5.2 Task5 (XON)

Let's have the following XON data *(see „task5/input/xonExample.xon"):*

```
{date: D2020-02-22,
  cities: [
    {from: [
        "Brussels",
        { to: "London", distance = 322I},
        { to: "Paris", distance = 265I}
      ]
    },
    {from: [
        "London",
        { to: "Brussels", distance = 322I},
        { to: "Paris", distance = 344I}
      ]
    }
  ]
}
```

The model of this XON data is written in the element xd:xon with the name "distances". Note that the model is written to xd:xon element (in fact you can always use JSON or XON formats are equivalent).

The complete X-definition (see *„src/task5/xonExample.xdef"):*

```
<xd:def xmlns:xd="http://www.xdef.org/xdef/4.2" name="xonExample" root="distances">
  <xd:xon xd:name="distances">
    { date: "date()",
      cities: [
        {%script = "occurs +",
          from: [
            "string()",
            {%script = "occurs +",
               to:       "jstring()",
               distance: "int(0, 9999)"
            }
          ]
        }
      ]
    }
  </xd:xon>
</xd:def>
```

Note that the model of XON data again contains a JSON text where the description of values is a string with an X-script description. The description of an XON model can be either in the JSON format interchangeably.

In the Java program for parsing JSON data, we use the method „jparse“ instead of “xparse”. The result of parsing is an XON object. You can print this object in JSON format using the static method “toXonString” from the class “org.xdef.xon.XonUtils”.

The complete Java program (see *„src/task5/XonExample.java“):*

```
package task5;

import java.io.File;
import java.io.FileOutputStream;
import java.io.OutputStreamWriter;
import org.xdef.sys.ArrayReporter;
import org.xdef.XDDocument;
import org.xdef.XDFactory;
import org.xdef.XDPool;
import java.io.PrintStream;
import java.io.Writer;
import java.util.Properties;
import org.xdef.xon.XonUtils;

public class XonExample {
        public static void main(String... args) throws Exception {
                // compile the XDPool object from the X-definition source
                Properties props = new Properties();
                XDPool xpool = XDFactory.compileXD(props, "src/task5/xonExample.xdef");
                // Create an instance of the XDDocument object (from XDPool)
                XDDocument xdoc = xpool.createXDDocument("xonExample");
                // prepare the error reporter
                ArrayReporter reporter = new ArrayReporter();
                xdoc.setProperties(props);
                Object xon = xdoc.jparse("task5/input/xonExample.xon", reporter);
                // check errors
                if (reporter.errorWarnings()) {
                        // write log file with errors
                        PrintStream ps = new PrintStream("task5/errors/xon.txt");
                        new File("task5/errors").mkdirs();
                        reporter.printReports(ps); //print errors
                        ps.close();
                        System.err.println("Input data error; see task5/errors/xon.txt");
                } else {
                        System.out.println("OK. See task5/output/result.xon");
                        // Store the parsed result
                        new File("task5/output").mkdirs();
                        Writer out = new OutputStreamWriter(
                                new FileOutputStream("task5/output/result.xon"), "UTF-8");
                        out.write(XonUtils.toXonString(xon, true));
                        out.close();
                }
        }
}
```

6 PROPERTIES/INI data

In this chapter, we discuss some examples with data in the properties or INI format. The code in the task processes the input data and checks its accuracy. In the examples, we describe again both the source text of the Java program as well as the X-definitions and input data which are available in the form of files that you can download data from examples at http://www.xdefinice.cz/en/download.
Note that the result of processed data is in a map where keys are in string format and values are the results of parsed items. Models of Properties or INI data are written as a text of element "xd:ini". For parsing both, the properties or INI data use the method "iparse" of the "XDDocument" class.

6.1 Task5 (Properties)

Let's have the following Properties data *(see „task5/input/propsExample.properties"):*

```
############# TRS configuration #############

# TRS user name
TRSUser = John Smith

# user directory
Home = D:/TRS_Client/usr/Smith
# authority(SECURITY | SOFTWARE | CLIENT | UNREGISTERED)
Authority=CLIENT
# Maximal item size (10000 .. 15000000)
ItemSize=4000000
# Receiver sleep time in seconds (1 .. 3600).
ReceiverSleep=1

# Remote server
RemoteServerURL=http://localhost:8080/TRS/TRSServer
SeverIP = 123.45.67.8
SendMailHost = smtp.synth.cz
MailAddr = jira@synth.cz
Signature = 12afe0c1d246895a990ab2dd13ce684f012b339c
```

The model of this Properties data is written in the element xd:ini with the name "congig". The complete X-definition (see *„src/task5/propsExample.xdef"):*

```
<xd:def xmlns:xd="http://www.xdef.org/xdef/4.2" name="propsExample" root="TRSconfig">
  <xd:ini xd:name="TRSconfig">
    TRSUser = string()
    Home = file()
    Authority = enum("SECURITY", "SOFTWARE", "CLIENT", "UNREGISTERED")
    ItemSize = int(10000, 15000000)
    ReceiverSleep = int(1, 3600)
    RemoteServerURL = url()
    SeverIP = ipAddr()
    SendMailHost = domainAddr()
    MailAddr = emailAddr()
    Signature = SHA1()
  </xd:ini>
</xd:def>
```

Note that the model of Properties data contains a text in the Properties format where the description of values is a string with an X-script description. If you want to add an X-script command you write it as a directive %sctipt = "…".

In the Java program for parsing Properties data, you use the method „iparse". The result of parsing is an object of Java.util.Map<String, Object>. You can print this object in Properties format using the static method "toIniString" from the class "org.xdef.xon.XonUtils".

The complete Java program (see *„src/task5/PropsExample.java"):*

```
package task5;

import java.io.File;
import java.io.FileOutputStream;
import java.io.OutputStreamWriter;
import org.xdef.sys.ArrayReporter;
import org.xdef.XDDocument;
import org.xdef.XDFactory;
import org.xdef.XDPool;
import java.io.PrintStream;
```

```
import java.io.Writer;
import java.util.Map;
import java.util.Properties;
import org.xdef.xon.XonUtils;

public class PropsExample {
        public static void main(String... args) throws Exception {
                // compile the XDPool object from the X-definition source
                Properties props = new Properties();
                XDPool xpool = XDFactory.compileXD(props,"src/task5/propsExample.xdef");
                // Create an instance of the XDDocument object (from XDPool)
                XDDocument xdoc = xpool.createXDDocument("propsExample");
                // prepare the error reporter
                ArrayReporter reporter = new ArrayReporter();
                xdoc.setProperties(props);
                Map<String, Object> ini =
                        xdoc.iparse("task5/input/propsExample.properties", reporter);
                // check errors
                if (reporter.errorWarnings()) {
                        // write log file with errors
                        new File("task5/errors").mkdirs();
                        PrintStream ps =
                                new PrintStream("task5/errors/properties.txt");
                        reporter.printReports(ps); //print errors
                        ps.close();
                        System.err.println("Input data error; see task5/errors/props.txt");
                } else {
                        System.out.println("OK. See task5/output/result.props");
                        // Store the parsed result
                        new File("task5/output").mkdirs();
                        Writer out = new OutputStreamWriter(
                                new FileOutputStream("task5/output/result.properties"), "ASCII");
                        out.write(XonUtils.toIniString(ini));
                        out.close();
                }
        }
}
```

6.2 Task5 (Windows INI)

Let's have the following Windows INI data *(see „task5/input/iniExample.ini"):*

```
############# TRS configuration #############
# TRS user name
TRSUser = John Smith
[User]
# user directory
Home = D:/TRS_Client/usr/Smith
# authority(SECURITY | SOFTWARE | CLIENT | UNREGISTERED)
Authority=CLIENT
# Maximal item size (10000 .. 15000000)
ItemSize=4000000
# Receiver sleep time in seconds (1 .. 3600).
ReceiverSleep=1
[Server]
# Remote server
RemoteServerURL=http://localhost:8080/TRS/TRSServer
SeverIP = 123.45.67.8
SendMailHost = smtp.synth.cz
MailAddr = jira@synth.cz
Signature = 12afe0c1d246895a990ab2dd13ce684f012b339c
```

The model of this INI data is written in the element xd:ini with the name "TRSconfig". Note that the model is written to xd:ini element to the description of the section "server" is added X-script specifying that this section is optional.

The complete X-definition (see *„src/task5/iniExample.xdef"):*

```
<xd:def xmlns:xd="http://www.xdef.org/xdef/4.2" name="iniExample" root="TRSconfig">
  <xd:ini xd:name="TRSconfig">
    TRSUser = string()
    [User]
      Home = file()
      Authority = enum("SECURITY", "SOFTWARE", "CLIENT", "UNREGISTERED")
      ItemSize = int(10000, 15000000)
      ReceiverSleep = int(1, 3600)
    [Server] %script = optional
      RemoteServerURL = url()
      SeverIP = ipAddr()
      SendMailHost = domainAddr()
      MailAddr = emailAddr()
```

```
      Signature = SHA1()
  </xd:ini>
</xd:def>
```

Note that the model of INI data contains the text in the INI format where the description of values is a string with an X-script description. If you want to add an X-script command you write it as a directive %sctipt = "…".

In the Java program for parsing Properties data use the method „iparse“. The result of parsing is an object of Java.util.Map<String, Object>. You can print this object in INI format using the static method “toIniString” from the class “org.xdef.xon.XonUtils”. The complete Java program (see *„src/task5/IniExample.java“):*

```
package task5;

import java.io.FileOutputStream;
import java.io.OutputStreamWriter;
import org.xdef.sys.ArrayReporter;
import org.xdef.XDDocument;
import org.xdef.XDFactory;
import org.xdef.XDPool;
import java.io.PrintStream;
import java.io.Writer;
import java.util.Map;
import java.util.Properties;
import org.xdef.xon.XonUtils;

public class IniExample {
        public static void  args) throws Exception {
                // compile XDPool from the X-definition
                Properties props = new Properties();
                XDPool xpool = XDFactory.compileXD(props, "src/task5/iniExample.xdef");
                // Create an instance of the XDDocument object (from XDPool)
                XDDocument xdoc = xpool.createXDDocument("iniExample");
                // prepare error reporter
                ArrayReporter reporter = new ArrayReporter();
                xdoc.setProperties(props);
                Map<String, Object> ini =
                        xdoc.iparse("task5/input/iniExample.ini", reporter);
                // check errors
                if (reporter.errorWarnings()) {
                        // write log file with errors
                        PrintStream ps = new PrintStream("task5/errors/ini_err.txt");
                        reporter.printReports(ps); //print errors
                        ps.close();
                        System.err.println("Incorrect input data");
                } else {
                        System.out.println("OK. See task5/output/result.ini");
                        // Store the parsed result
                        Writer out = new OutputStreamWriter(
                                new FileOutputStream("task5/output/result.ini"), "ISO8859-2");
                        out.write(XonUtils.toIniString(ini));
                        out.close();
                }
        }
}
```

7 X-components

X-components are generated as Java source code and therefore need to be generated in some directory first and then compile with a Java compiler.

7.1 Task6

Let's have an XML document describing the city, streets, and tenants *(see „task6/input/town.xml“)*:

```
<Town Name='Nonehill'>
  <Street Name='Long'>
    <House Num='1'>
      <Person FirstName='John' LastName='Smith'></Person>
      <Person FirstName='Jane' LastName='Smith'></Person>
    </House>
    <House Num='2'/>
    <House Num='3'>
      <Person FirstName='James' LastName='Smith'></Person>
    </House>
  </Street>
  <Street Name='Short'>
    <House Num='1'>
      <Person FirstName='Jeremy' LastName='Smith'></Person>
    </House>
  </Street>
</Town>
```

The X-definition describing the above XML data (*see „src/task6/input/townA.xdef“*):

```
<xd:def xmlns:xd="http://www.xdef.org/xdef/4.2" name="A" root="Town">
  <Town Name="string">
     <Street xd:script="*;" Name="string">
        <House xd:script="*; ref House"/>
     </Street>
  </Town>

  <House Num="int" Address="optional string;">
     <Person xd:script="*; ref Person" />
  </House>

  <Person FirstName="string" LastName="string();"/>

</xd:def>
```

In the following X-definition, the "%class" commands are used to specify which X-components will be generated. Note that we also create the interface „Citizen“ from the component „Person“ and this interface is implemented in the class „Person“. This allows access to the class „Person“ with this interface (*see „src/task6/input/townB.xdef“*):

```
<xd:def xmlns:xd="http://www.xdef.org/xdef/4.2" name="B">
<xd:component>
  /* X-Components generated from the X-definition A */
  %class task6.components1.City %link A#Town;
  %class task6.components1.House %link A#House;
</xd:component>
</xd:def>
```

Java program that generates the source code of X-components to the directory „src/task6/components1“. Note it is also created the file „src/task6/components1/Town1.xp“ containing compiled XDPool. This is recommended because X-components are generated from a specific instance of XDPool.

See *„src/task6/input/ GenComponents1.java“*:

```
package task6;

import java.io.File;
import java.io.FileOutputStream;
import java.io.ObjectOutputStream;
import org.xdef.XDFactory;
import org.xdef.XDPool;

/** Compile X-definitions and create the class with compiled XDPool. */
public class GenComponents1 {
        public static void main(String... args) throws Exception {
                // 1. delete the directory with X-componets
```

```
                File components = new File("src/task6/components1");
                if (components.exists()) {
                        for (File x: components.listFiles()) {
                                x.delete();
                        }
                        components.delete();
                }

                // 2. Compile X-definitions
                XDPool xPool = XDFactory.compileXD(null,  //use System properties
                        "src/task6/townA.xdef",  // X-definition files
                        "src/task6/townB.xdef");

                // 3. generate X-components
                xPool.genXComponent(new File("src/"), "UTF-8", false, false);
                System.out.println("X-components are generated to\n"
                        + components.getCanonicalPath());

                // 4. save XDPool to the file "src/task6/components/Town1.xp"
                ObjectOutputStream os = new ObjectOutputStream(
                        new FileOutputStream(new File(components, "Town1.xp")));
                os.writeObject(xPool);
                os.close();
                System.out.println("XDPool is saved to\n"
                        + components.getCanonicalPath() + File.separator + "Town1.xp");
        }
}
```

We display only important parts of generated X-components. The class Street is the inner class of the class City and the class Person is an inner class of the class House:

```
================================================
X-component City (contains inner class Street)
================================================
public class City implements org.xdef.component.XComponent{
  public String getName() {return _Name;}
  public java.util.List<City.Street> listOfStreet() {
    return _Street;
  }
...
  public static class Street implements org.xdef.component.XComponent{
    public String getName() {return _Name;}
    public java.util.List<task6.components.House> listOfHouse() {
      return _House;
    }
    ...
  }
}

================================================
X-component House (contains inner class Person)
================================================
public class House implements org.xdef.component.XComponent{
  public Integer getNum() {return _Num;}
  public String getAddress() {return _Address;}
  public java.util.List<Person> listOfPerson() {
    return _Person;
  }
  public void setNum(Integer x){_Num=x;}
  public void setAddress(String x){_Address=x;}
...
  public static class Person implements org.xdef.component.XComponent{
    public String getFirstName() {return _FirstName;}
    public String getLastName() {return _LastName;}
    ...
  }
}
```

All generated Java files must be compiled with a Java compiler before the next step.

The following Java program processes the input XML document and creates the X-component instance from it. Using the X-component "City", data about streets, houses, and tenants in the city are printed. Then the address is added to each house and an XML document is created from the X-component and written to the file "test6/output/data1.xml". See Java program *„src/task6/input/Town1.java“*:

```
package task6;

import java.io.File;
import java.io.FileInputStream;
import java.io.ObjectInputStream;
import org.w3c.dom.Element;
import org.xdef.XDDocument;
import org.xdef.XDPool;
```

```
import org.xdef.xml.KXmlUtils;
import task6.components1.City;
import task6.components1.House;

public class Town1 {

        public static void main(String... args) throws Exception {
                // 1. Read the compiled XDPool object from the file
                ObjectInputStream ois = new ObjectInputStream(
                        new FileInputStream("src/task6/components1/Town1.xp"));
                XDPool xPool = (XDPool) ois.readObject();
                ois.close();7

                // 2. Create XDDocument
                XDDocument xd = xPool.createXDDocument("A");

                // 3. Create an instance of the X-component City (unmarchall)
                // (Note the generated X-components are in the package "components".)
                City city = (City)xd.xparseXComponent("task6/input/town.xml",null,null);

                // 4. Print out the contents of the object City
                System.out.println("City " + city.getName());
                for (City.Street street: city.listOfStreet()) {
                        System.out.println("Street " + street.getName() + ":");
                        for (House house: street.listOfHouse()) {
                                System.out.print("House No. " + house.getNum() + ". ");
                                if (house.listOfPerson().size() > 0) {
                                        System.out.println("Tenants :");
                                        for (House.Person citizen: house.listOfPerson()) {
                                                System.out.println(citizen.getFirstName()
                                                        + " " + citizen.getLastName());
                                        }
                                } else {
                                        System.out.println("No tenants in this house.");
                                }
                        }
                }

                // 5. Update the address for each house.
                for (City.Street street: city.listOfStreet()) {
                        for (House house: street.listOfHouse()) {
                                house.setAddress(city.getName() + ", " + street.getName()
                                        + " " + house.getNum());
                        }
                }

                // 6. Save XML with addresses to the file data1.xml
                Element el = city.toXml();
                new File("task6/output").mkdirs();
                KXmlUtils.writeXml("task6/output/data1.xml", el, true, false);
                System.out.println("\nElement City written to: test6/output/data1.xml");
        }
}
```

The file „test9/output/data1.xml“ is a stored XML document created from X-component.

7.2 Task6 (variant with transformation)

In this example, we will use the input XML data and the X-definition from the previous paragraph. From the input data, we create a list of tenants in the given city. The X-definition describing this list has the form (see *„src/task6/townC.xdef“*):

```
<xd:def xmlns:xd="http://www.xdef.org/xdef/4.2" name="C" root="Residents">
  <Residents>
    <Resident xd:script="*; create from('//Person');"
      GivenName="string; create from('@FirstName')"
      FamilyName="string; create from('@LastName')"
      Address="string; create from('../@Address')" />
  </Residents>
</xd:def>
```

Description of X-components (see *„src/task6/townD.xdef“*)::

```
<xd:def xmlns:xd="http://www.xdef.org/xdef/4.2" name="D">
<xd:component>
  %class task6.components2.City %link A#Town;
  %class task6.components2.House %link A#House;
  /* Generate interface according to X-component A#Person */
  %interface task6.components2.Citizen %link A#Person;
  /* The generated X-component "Person" implements the interface
     "task6.components2.Citizen" */
```

```
  %class task6.components2.Person implements task6.components2.Citizen
     %link A#Person;
  /* XKomponents generated from the X-definition C */
  %class task6.components2.Tenants %link C#Residents;
  /* Binds */
  %bind FirstName %link C#Residents/Resident/@GivenName;
  %bind LastName %link C#Residents/Resident/@FamilyName;
  %bind Address %link C#Residents/Resident/@Address;
</xd:component>
</xd:def>
```

The following Java program generates the source code of X-components and the file Town1.xp. See *„src/task6/GenComponents2.java“*:

```
package task6;

import java.io.File;
import java.io.FileOutputStream;
import java.io.ObjectOutputStream;
import org.xdef.XDFactory;
import org.xdef.XDPool;

/** Compile X-definitions and create the class with compiled XDPool. */
public class GenComponents2 {
        public static void main(String... args) throws Exception {
                // 1. delete the directory with X-components
                File components1 = new File("src/task6/components2");
                if (components1.exists()) {
                        for (File x: components1.listFiles()) {
                                x.delete();
                        }
                        components1.delete();
                }

                // 2. Compile the X-definitions
                XDPool xPool = XDFactory.compileXD(null,  //use System properties
                        "src/task6/townA.xdef",  // X-definition files
                        "src/task6/townC.xdef",
                        "src/task6/townD.xdef");

                // 3. generate X-components
                xPool.genXComponent(new File("src/"), "UTF-8", false, false);
                System.out.println("X-components are generated to\n"
                        + components1.getCanonicalPath());

                // 4. save XDPool to the file "src/task6/components1/Town2.xp"
                ObjectOutputStream os = new ObjectOutputStream(
                        new FileOutputStream(new File(components1, "Town2.xp")));
                os.writeObject(xPool);
                os.close();

                System.out.println("XDPool and X-components are generated to\n"
                        + components1.getCanonicalPath());
                System.out.println("XDPool is saved to\n"
                        + components1.getCanonicalPath() + File.separator + "Town2.xp");
        }
}
```

After running this program, the interface „Citizen“ and the X-component „Tenants“ are added to the set of X-components in the directory *„src/task6/components2“*:

```
Interface Citizen
================================================
public interface Citizen extends org.xdef.component.XComponent {
  public String getFirstName();
  public String getLastName();
...
}

================================================
X-component Tenants (contains inner class Resident)
================================================
public class Tenants implements org.xdef.component.XComponent{
  public java.util.List<Tenants.Resident> listOfResident() {
    return _Resident;
  }
  ...
  public void addResident(Tenants.Resident x) {
    if (x!=null) _Resident.add(x);
  }
  public static class Resident implements org.xdef.component.XComponent{
    public String getFirstName() {return _FirstName;}
    public String getLastName() {return _LastName;}
```

```
    public String getAddress() {return _Address;}
    ...
  }
}
```

All generated Java files must be compiled with a Java compiler before the next step.

The following Java program processes the input XML document and creates the X-component instance from it. Using the X-component "City", data about streets, houses, and tenants in the city are printed. Then the address is added to each house and an XML document is created from the X-component and written to the file "*test6/output/data2.xml*". See Java program *„src/task6/Town2.java“*:

```
package task6;

import java.io.FileInputStream;
import java.io.ObjectInputStream;
import org.xdef.xml.KXmlUtils;
import org.w3c.dom.Element;
import org.xdef.XDDocument;
import org.xdef.XDPool;
import org.xdef.component.XComponentUtil;
import task6.components2.House;
import task6.components2.City;
import task6.components2.Tenants;

public class Town2 {

        public static void main(String... args) throws Exception {
                // 1. Get compiled XDPool from the file
                ObjectInputStream ois = new ObjectInputStream(
                        new FileInputStream("src/task6/components2/Town2.xp"));
                XDPool xpool = (XDPool) ois.readObject();
                ois.close();

                // 2. Create XDDocument
                XDDocument xd = xpool.createXDDocument("A");

                // 3. Create the instance of the X-component City (unmarchall)
                // (Note the generated X-components are in the package "components".)
                City city = (City)xd.xparseXComponent("task6/input/town.xml",null,null);

                // 4. Update the address for each house.
                for (City.Street street: city.listOfStreet()) {
                        for (House house: street.listOfHouse()) {
                                house.setAddress(city.getName() + ", " + street.getName()
                                        + " " + house.getNum());
                        }
                }

                // 5. Transform it to the X-component Tenants
                Tenants tenants =
                        (Tenants) XComponentUtil.toXComponent(city, xpool, "C#Residents");

                // 6. Print tenants from the object "tenants"
                for (Tenants.Resident x: tenants.listOfResident()) {
                        System.out.println(x.getFirstName()
                           + " " + x.getLastName() + "; " + x.getAddress());
                }

                // 7. save the transformed version to the file data2.xml
                Element el = tenants.toXml();
                KXmlUtils.writeXml("task6/output/data2.xml", el, true, false);
                System.out.println("Tenants written to: task6/output/data2.xml");
        }
}
```

The generated file *„task6/output/data2.xml“*:

```
<Residents>
  <Resident Address="Nonehill, Long 1" FamilyName="Smith" GivenName="John"/>
  <Resident Address="Nonehill, Long 1" FamilyName="Smith" GivenName="Jane"/>
  <Resident Address="Nonehill, Long 3" FamilyName="Smith" GivenName="James"/>
  <Resident Address="Nonehill, Short 1" FamilyName="Smith" GivenName="Jeremy"/>
</Residents>
```

7.3 Task6 (using X-component to construct data)

X-component is a Java class. It is therefore possible to create an instance of this class and use setters to set the values in it. From the X-component it is then possible to create an XML object or e.g. XON, JSON, etc. Let's use the

X-component "task6.components2.Tenants" from the previous task as an example. First, create the instance of the root X-component Tenants. The resident is added to the list of residents in the method „addResident“. After filling in the values, let's create XML data from the X-component (see the method „toXml“). Note that there was no need to use X-definition in this program.

See Java program *„src/task6/ CreateResidentList.java“*:

```
package task6;

import java.util.List;
import org.w3c.dom.Element;
import org.xdef.xml.KXmlUtils;
import task6.components2.Tenants;
import task6.components2.Tenants.Resident;

public class CreateResidentList {

        /** Add residents to the list of residents in the X-component Tenants. */
        private static void addResident(Tenants tenants,
                String firstName, String lastName, String address) {
                // 1. Create an instance of X/component Resident.
                Resident resident = new Resident();

                // 2. Set values of Resident
                resident.setFirstName(firstName);
                resident.setLastName(lastName);
                resident.setAddress(address);

                // 3. Add created the resident to the list of residents
                tenants.addResident(resident);
        }

        public static void main(String... args) throws Exception {
                // 1. Create an instance of X-component (model of element Residents)
                Tenants tenants = new Tenants();
                List<Resident> list = tenants.listOfResident();
                Resident resident = new Resident();
                resident.setFirstName("Michal");
                resident.setLastName("Kotek");
                resident.setAddress("Praha, Balbinova 3");
                list.add(resident);

                // 2. Add residents to the X-component
                addResident(tenants, "Janes", "Smith", "Nonehill, Long 3");
                addResident(tenants, "Jeremy", "Smith", "Nonehill, Short 1");
                addResident(tenants, "Jane", "Smith", "Newmill, Innis st. 15");

                // 2. Add residents to the list of residents
                for (Tenants.Resident x: tenants.listOfResident()) {
                        System.out.println(x.getFirstName()
                           + " " + x.getLastName() + "; " + x.getAddress());
                }

                // 4. Create XML with residents to the file data3.xml
                Element el = tenants.toXml();
                KXmlUtils.writeXml("task6/output/data3.xml", el, true, false);
                System.out.println("\nCreated data written to test6/output/data3.xml");
        }
}
```

The generated file "test6/output/data3.xml":

```
<Residents>
  <Resident Address="Praha, Balbinova 3" FamilyName="Kotek" GivenName="Michal"/>
  <Resident Address="Nonehill, Long 3" FamilyName="Smith" GivenName="Janes"/>
  <Resident Address="Nonehill, Short 1" FamilyName="Smith" GivenName="Jeremy"/>
  <Resident Address="Newmill, Innis st. 15" FamilyName="Smith" GivenName="Jane"/>
</Residents>
```

Note that constructed XML object is not validated with the X-definition. If you want to validate it you must create the XDDocument from XDPool and invoke the method „xparse“ with the created XML.

8 Lexicon

The "lexicon" technology allows you to work with X-definition with Java mutations of XML documents. Several language versions can be described in the X-definition. The name of the language is in the "language" attribute in the header of the "xd:lexicon" section. The lexicon technology allows reading sets of the attribute and element names according to the corresponding lexicon section. Keep in mind that these are only the names of the items in the XML document and not the conversion of the data contained in these items.

8.1 Task7

This example will show the use of the lexicon in X-definitions. We have two versions of the input data in English and German.

The English version (see *„task7/input/town_eng.xml“*):

```
<Town Name='Nonehill'>
  <Street Name='Long'>
    <House Num='1'>
      <Person FirstName='John' LastName='Smith'></Person>
      <Person FirstName='Jane' LastName='Smith'></Person>
    </House>
    <House Num='2'/>
    <House Num='3'>
      <Person FirstName='James' LastName='Smith'></Person>
    </House>
  </Street>
  <Street Name='Short'>
    <House Num='1'>
      <Person FirstName='Jeremy' LastName='Smith'></Person>
    </House>
  </Street>
</Town>
```

The German version (see *„task7/input/town_deu.xml“*):

```
<Stadt Name='Nonehill'>
  <Straße Name='Long'>
    <Haus Nummer='1'>
      <Person Vorname='John' Nachname='Smith'></Person>
      <Person Vorname='Jane' Nachname='Smith'></Person>
    </Haus>
    <Haus Nummer='2'/>
    <Haus Nummer='3'>
      <Person Vorname='James' Nachname='Smith'></Person>
    </Haus>
  </Straße>
  <Straße Name='Short'>
    <Haus Nummer='1'>
      <Person Vorname='Jeremy' Nachname='Smith'></Person>
    </Haus>
  </Straße>
</Stadt>
```

The following X-definition describes the structure of the object, which corresponds to the English version. So in the lexicon section of the English port, just write the default="true" attribute. Entries in the "lexicon" section must contain direct X-positions in the models, they must not refer to positions in the referenced parts of the model (eg. „town#Town/Street/House/@Num“ or „town#Town/Street/HousePerson“ would not be correct).

The lexicon for the German and Czech versions describes the names that correspond to the corresponding positions in models. (see *„src/task7/town.xdef“*):

```
<xd:def xmlns:xd="http://www.xdef.org/xdef/4.2" root="Town" name="town">

  <Town Name="string">
     <Street xd:script="*;" Name="string">
        <House xd:script="*; ref _house"/>
     </Street>
  </Town>

  <_house Num="int" Address="optional string;">
     <Person xd:script="*; ref _person" />
  </_house>
```

```
  <_person FirstName="string" LastName="string();"/>

  <xd:lexicon language="eng" default="yes"/>

  <xd:lexicon language="deu">
    town#Town =                  Stadt
    town#Town/@Name =            Name
    town#Town/Street =           Straße
    town#Town/Street/@Name =     Name
    town#Town/Street/House =     Haus
    town#_house/@Num =           Nummer
    town#_house/@Address =       Adresse
    town#_house/Person =         Person
    town#_person/@FirstName =    Vorname
    town#_person/@LastName =     Nachname
  </xd:lexicon>

  <xd:lexicon language="ces">
    town#Town =                  Město
    town#Town/@Name =            Jméno
    town#Town/Street =           Ulice
    town#Town/Street/@Name =     Jméno
    town#Town/Street/House =     Dům
    town#_house/@Num =           Číslo
    town#_house/@Address =       Adresa
    town#_house/Person =         Osoba
    town#_person/@FirstName =    Jméno
    town#_person/@LastName =     Příjmení
  </xd:lexicon>

</xd:def>
```

Use the following Java program to read the data in the English and German versions and finally create an XML document with the Czech version. In the Java program, it is enough to set the name of the language version by the method "setLexiconLanguage". For conversion („translation“) of XML data from one language to the other is used the method „xtranslate“. See *„src/task7/Town.java“*:

```
package task7;

import java.io.File;
import org.w3c.dom.Element;
import org.xdef.XDDocument;
import org.xdef.XDFactory;
import org.xdef.XDPool;
import org.xdef.sys.ArrayReporter;
import org.xdef.xml.KXmlUtils;

public class Town {

        public static void main(String... args) throws Exception {
                // 1. Compile X-definitions
                XDPool xPool = XDFactory.compileXD(null, "src/task7/town.xdef");

                XDDocument xd;
                Element el;
                ArrayReporter reporter = new ArrayReporter();

                // 3. Create XDDOcument, set language, and process the localized version
                // for the English language
                xd = xPool.createXDDocument("town");
                xd.setLexiconLanguage("eng");
                el = xd.xparse("task7/input/town_eng.xml", reporter);
                if (reporter.errors()) {
                        System.err.println("Error on English version");
                }

                // 3. Create XDDOcument, set language, and process the localized version
                // for the English language
                xd = xPool.createXDDocument("town");
                xd.setLexiconLanguage("eng");
                reporter.clear();
                el = xd.xparse("task7/input/town_eng.xml", reporter);
                if (reporter.errors()) {
                        System.err.println("Error on English version");
                } else {
                        System.out.println("eng OK");
                }

                // 4. Create XDDOcument, set language, and process the localized version
                // for the German language
                xd = xPool.createXDDocument("town");
                xd.setLexiconLanguage("deu");
```

```
                reporter.clear();
                el = xd.xparse("task7/input/town_deu.xml", reporter);
                if (reporter.errors()) {
                        System.err.println("Error on German version");
                } else {
                        System.out.println("deu OK");
                }

                // 5. Create XDDOcument and translate the localized version to Czech.
                // result write to the file "task7/output/town_ces.xml"
                xd = xPool.createXDDocument("town");
                reporter.clear();
                el = xd.xtranslate("task7/input/town_deu.xml", "deu", "ces", reporter);
                new File("task7/output").mkdirs();
                if (reporter.errors()) {
                        System.err.println("Error on translation from 'deu' to 'ces'");
                } else {
                        KXmlUtils.writeXml("task7/output/town_ces.xml", el, true, false);
                        System.out.println(
                                "town_deu transated to ces (see task7/output/town_ces.xml)");
                }

                // 6. Check translated Czech version
                xd = xPool.createXDDocument("town");
                xd.setLexiconLanguage("ces");
                reporter.clear();
                xd.xparse("task7/output/town_ces.xml", reporter);
                if (reporter.errors()) {
                        System.err.println("errors in translation");
                }
        }
}
```

The Czech version of the document is written to the file *„task7/output/town_ces.xml“*:

```
<Město Jméno="Nonehill">
  <Ulice Jméno="Long">
    <Dům Číslo="1">
      <Osoba Jméno="John" Příjmení="Smith"/>
      <Osoba Jméno="Jane" Příjmení="Smith"/>
    </Dům>
    <Dům Číslo="2"/>
    <Dům Číslo="3">
      <Osoba Jméno="James" Příjmení="Smith"/>
    </Dům>
  </Ulice>
  <Ulice Jméno="Short">
    <Dům Číslo="1">
      <Osoba Jméno="Jeremy" Příjmení="Smith"/>
    </Dům>
  </Ulice>
</Město>
```