X-thesaurus 3.2

Brief introduction

Jindřich Kocman, Václav Trojan

Version: **3.2**
Date: **2019-01-24**

Inventory of changes

X-definition 3.2

Default version X-definition 3.1

Contents

1 Notice

Questions, remarks and bug reports please send to: xdef@syntea.cz.
The actual version of X-definition you can download from: http://www.xdef.cz

2 Introduction

This document describes the basic use of the "X-thesaurus" technology in X-definition. It is designed for programmers who need to work with XML data in different local languages. X-thesaurus is built on top of X-definition and it enables to work with XML data modified to different local languages. For using X-thesaurus it is required a basic knowledge about X-definition.

2.1 Terms and abbreviations

X-definition	1. The language used for description of the structure, content, processing and construction of XML objects. 2. XML element in the X-definition language.
X-components	Technology for creating of Java classes from the X-definitions
X-component	Java class that was created by the X-component technology.
XComponent	Java interface org.xdef,component.XComponent that is implemented by X-components.
Unmarshalling	Populate an object with data from the XML.
Marshalling	Data conversion of an object into XML. Opposite of the unmarshalling.
xd:component	XML element that contains a list of descriptions of X-component in the X-definition
XDPosition	XDPosition is a description of a location of a model in the set of X-definitions (compiled to XDPool)

3 X-thesaurus

The X-thesaurus is a XML document describing the local names of XML elements, attributes and enumeration items in the models from X-definitions according to given local language. Each model of element in the X-definition corresponds to one generated X-component. The values of attributes, elements, and text values in the X-component, are accessible by the methods get***NAME*** and set***NAME***, where "***NAME***" is the name of the attribute or of element in X-definition. The conversion of the value of an attribute or of a text node described in the X-definition and the type of object in Java is given by the following table:

X-definition datatype	X-component
olean	`java.lang.Boolean`
byte, short, int, long (and derived types)	`java.lang.Long`
float, double	`java.lang.Double`
decimal, dec	`java.math.BigDecimal`
datetime, xdatetime, gDate, gTime, gYear, gMoth, gYearMonth, gMonthDay	`org.xdef,sys.SDatetime` (also available as `java.util.Calendar`, `java.util.Date`, `java.sql.Timestamp`)
duration	`org.xdef,sys.SDuration`
base64Binary, hexBinary, base64, hex	`byte[]`
enum	`Java.lang.String` or `Enum` in Java
other datatypes	`java.lang.String`

The values of child elements are represented by X-component objects. If the maximum number of quantifier is higher than one then it is represented by java.util.List< correspondent X-component>. Also if there occur more than one text value at one position it will be represented as java.util.List<correspondent datatype>.

All types are objects (not the primitive values), so it is possible to test in the program if the value exists in the X-component object (e.g. optional values may not exist). If the element or a value may have more occurrences, it creates an array of values (implemented using java.util.List) and instead of a setter is generated the method add***NAME***. Because there are more possibilities to work with datatypes of datetime the getters are generated in more variants: timestampOf***NAME***, calendarOf***NAME***, and dateOf***NAME***.

Note some of commands in the X-definition affect the creation of X-component

- *ignore – the X-definition object is ignored, code is not generated in X-component.*
- *forget – the elements are deleted from the memory after processing of the document. However, the corresponding X-component is created*

Example

Let us have a master X-definition describing an insurance contract:

```
<Contract
    VIN   = "required int()"
    Date  = "required date()">
  <Owner
    Name        = "required string()"
    CompanyID   = "required num(8)" />
  <Keeper  xd:script="occurs 1.."
    PersonalID = "required num(10)" />
</Contract>
```

Generated X-component will look as follows (the constructors and the methods of the interface org.xdef,component.XComponent are not displayed):

```
package user.kocman.manual;

public class Contract implements org.xdef,component.XComponent{
  public org.xdef,sys.SDatetime getDate() {return _Date;}
  public java.util.Date dateOfDate() {
    return org.xdef,sys.SDatetime.getDate(_Date);
  }
```

```
  public java.sql.Timestamp timestampOfDate() {
    return org.xdef,sys.SDatetime.getTimestamp(_Date);
  }
  public java.util.Calendar calendarOfDate() {
    return org.xdef,sys.SDatetime.getCalendar(_Date);
  }
  public Long getVIN() {return _VIN;}
  public Contract.Owner getOwner() {return _Owner;}
  public java.util.List<Contract.Keeper> listOfKeeper() {return _Keeper;}
  public void setDate(org.xdef,sys.SDatetime x) {_Date = x;}
  public void setDate(java.util.Date x) {
    _Date = x==null ? null : new org.xdef,sys.SDatetime(x);
  }
  public void setDate(java.sql.Timestamp x) {
    _Date = x==null ? null : new org.xdef,sys.SDatetime(x);
  }
  public void setDate(java.util.Calendar x) {
    _Date = x==null ? null : new org.xdef,sys.SDatetime(x);
  }
  public void setVIN(Long x) {_VIN = x;}
  public void setOwner(Contract.Owner x) {_Owner = x;}
  public void addKeep;er(Contract.Keeper x) {
    if (x!=null) _Keeper.add(x);
  }
  private org.xdef,sys.SDatetime _Date;
  private Long _VIN;
  private Contract.Owner _Owner;
  private final java.util.List<Contract.Keeper> _Keeper =
    new java.util.ArrayList<Contract.Keeper>();
  // Constructors and implementation of the interface XComponent ...

public static class Owner implements org.xdef,component.XComponent{
  public String getICO() {return _ICO;}
  public String getName() {return _Name;}
  public void setICO(String x) {_ICO = x;}
  public void setName(String x) {_Name = x;}
  public String xposOfICO(){return XD_XPos + "/@ICO";}
  public String xposOfName(){return XD_XPos + "/@Name";}
  private String _ICO;
  private String _Name;
  // Constructors and implementation of the interface XComponent ...
}
public static class Keeper implements org.xdef,component.XComponent{
  public String getPersonalID() {return _PersonalID;}
  public void setPersonalID (String x) {_PersonalID = x;}
  public String xposOfRC(){return XD_XPos + "/@PersonalID ";}
  private String _PersonalID;
  // Constructors and implementation of the interface XComponent ...
}
}
```

3.1 Instance of an X-component

Each X-component implements Java interface `org.xdef,component.XComponent` that allows conversion from/to the XML data format. The X-component instance can be created when parsing the XML according to the model in the X-definition when it comes to populate the appropriate data from the XML (operation unmarshalling). X-component can also be created using the constructor, and populate it with the data by Java program. From an instance of an X-component you can create the XML element by calling the method `org.xdef.XComponent.toXml()` (marshalling operation).

3.2 XDPosition

XDPosition is a description of a location of a model in the set of X-definitions (compiled to XDPool object). It consists of the name of X-definition, followed by the "#" and the name of the model. A model can be followed by the "/" character, and either the name of the nested element "$text" for a text value, or "@", and the name of the attribute. If the name contains a prefix of namespace it is written with the prefix, as listed in the X-definition. If in a set of descendants of an element contains more nodes with the same name, it is described by the serial number of this item in the square brackets (the numbers start from one: "[1]"). The first element is taken if the number is not specified (i.e. "[1]" may not be written).

In the following example are described the XDPositions of individual items:

```
<xd:def name = "Model">
  <A>                          Model#A
    <B                         Model#A/B
      b = "string()" />        Model#A/B/@b
    <C />                      Model#A/C
    <B>                        Model#A/B[2]
      required string();       Model#A/B/$text
    </B>
  </A>
</xd:def>
```

3.3 Commands to generate X-components

The list of commands used for creating the X-component is written in the X-definition as the text of the element <xd:component>. The <xd:component> element may be written in the separate X-definition or it can be part of any other X-definition. Each command in the list is terminated with ";".

3.3.1 Command %class

The command `%class` defines a Java class which is generated according to the model of the element in the X-definition. The %class keyword is followed by the fully qualified class name and by the keyword `%link` that specifies XDPosition of the model in the set of X-definitions, from which the X-component is generated. If the X-component extends a Java class or it implements an Java interface then you can specify "`extends SuperClassName implements InterfaceName`" after the name of Java class, where SuperClassName and InterfaceName must be the fully-qualified names. The syntax is the same as in the declaration of a class in the Java language.

Example 1

Source X-definition:

```
<xd:def xmlns:xd="http://www.xdef.org/3.2"
   xd:name="Vehicle"
   xd:root="Vehicle">

    <Vehicle
          VIN  = "required string()"
    />

    <xd:component>
        %class cz.syntea.tutorial.Vehicle
               extends cz.syntea.tutorial.PreVehicle
               implements cz.syntea.tutorial.IVehicle
           %link Vehicle#Vehicle;
    </xd:component>
</xd:def>
```

Generated X-component:

```
package cz.syntea.tutorial;

public class Vehicle extends cz.syntea.tutorial.PreVehicle
  implements cz.syntea.tutorial.IVehicle, org.xdef.component.XComponent{
  public String getVIN() {return _VIN;}
  public void setVIN(String x) {_VIN = x;}
  private String _VIN;
  // Constructors and implementation of the interface XComponent ...
}
```

3.3.2 Command %bind

By the command `%bind` can be set the new name of an item in an X-definition (the attribute model, element model or text node model and the corresponding names of the getters and setters). The keyword %bind is followed with the name that will be applied instead of the automatically generated. After the specification of the name the command continues with the keyword `%from` which is followed by the list of XDPositions (separated by comma) to which the statement relates. The same name can be used in more models. The getters and setters will automatically be adjusted to match the newly assigned name (see Example 2). If the generated Java class has an ancestor, you can use the %bind

command to bind the getter and setter defined in the ancestor. In this case a given variable, including getters and setters will not be generated and it uses the implementation of these methods in the ancestor.

Example 2

Let's have a truck that extends the vehicle defined in the previous example. The generated X-component does not contain getters and setters of the items that are inherited:

```
<xd:def xmlns:xd="http://www.xdef.org/3.2"
    xd:name="Truck"
    xd:root="Truck">

    <Truck xd:script = "ref Vehicle#Vehicle"
          MaxWeight = "required int()"
    />

    <xd:component>
        %bind VIN %with cz.syntea.tutorial.Vehicle %link Truck#Truck/@VIN;
        %class cz.syntea.tutorial.Truck
                extends cz.syntea.tutorial.Vehicle
            %link Truck#Truck;
    </xd:component>
</xd:def>
```

Generated X-component:

```
package cz.syntea.tutorial;
public class Truck extends cz.syntea.tutorial.Vehicle
  implements org.xdef.component.XComponent{
  public Long getMaxWeight() {return _MaxWeight;}
  public void setMaxWeight(Long x) {_MaxWeight = x;}
  private Long _MaxWeight; getVIN()
  // The methods getVIN and setVIN are already implemented in the superclass Vehicle!

  // Constructors and implementation of the interface XComponent ...
}
```

3.3.3 Command %interface

The command `%interface` is used when the (final) model takes the structure of another (referenced) model and, where appropriate, it also adds the additional attributes, text values or elements. To make the X-components generated from the models behave like the X-component created from the referenced ones, you can create an interface from the given model. This interface can then be added for generating of final models (see 3.3.1). The interface command starts with the keyword %interface, which is followed by the fully qualified name of the interface and by keyword `%link` followed by XDPosition of the model in the project.

Example 3

Personal car shares the part of the structure of the Vehicle. This structure was pulled out and the element Personal links to it.

```
<xd:def  xmlns:xd="http://www.xdef.org/3.2"
    xd:name="Personal"
    xd:root="Personal">

    <Vehicle
        VIN  = "required string()"
    />

    <Personal xd:script = "ref Vehicle"
        MaxPersons = "required int()"
    />

    <xd:component>
        %interface cz.syntea.tutorial.IVehicle %link Personal#Vehicle;
        %class cz.syntea.tutorial.Personal
                implements cz.syntea.tutorial.IVehicle
            %link Personal#Personal;
    </xd:component>

</xd:def>
```

The generated interface declares all the getters and setters of Vehicle:

```
package cz.syntea.tutorial;

public interface IVehicle extends org.xdef.component.XComponent {
  public String getVIN();
  public void setVIN(String x);
  public String xposOfVIN();
}
```

The X-component Personal implements the interface IVehicle:

```
package cz.syntea.tutorial;

public class Personal implements cz.syntea.tutorial.IVehicle,org.xdef.component.XComponent{
  public Long getMaxPersons() {return _MaxPersons;}
  public String getVIN() {return _VIN;}
  public void setMaxPersons(Long x) {_MaxPersons = x;}
  public void setVIN(String x) {_VIN = x;}
  private Long _MaxPersons;
  private String _VIN;
  // Constructors and implementation of the interface XComponent ...
}
```

3.3.4 Command %ref

It often happens that the project (XDPool) is generated from more X-definitions. In the case the X-component is generated from given X-definition, but XDPool is different (for example, there are some X-definition extra, missing etc.), it can be used the already created X-component and it's necessary to prevent its new generation (for example, if the X-component is located in another Jar file). To refer to the already generated X-component is provided by the command `%ref` with the fully qualified name of the already generated X-component and with the keyword `%link` with the XDPosition of the model in the XDPool.

Example

In the Jar file, which is in the current classpath, already is the generated X-component `cz.syntea.test.prexd.Vehicle` from Example 1. Then, when you create a set of new X-components, the already generated X-component is used and it is not generated again.

```
<xd:def xmlns:xd="http://www.xdef.org/3.2"
    xd:name="Fleet"
    xd:root="Fleet">

    <Fleet>
        <Personal  xd:script = "ref Vehicle#Vehicle; occurs 0.."/>
        <Van xd:script = "ref Vehicle#Vehicle; occurs 0.."/>
    </Fleet>

    <xd:component>
        %ref cz.syntea.tutorial.Vehicle   %link Vehicle#Vehicle;
        %class cz.syntea.tutorial.Fleet %link Fleet#Fleet;
    </xd:component>
</xd:def>
```

```
package cz.syntea.tutorial;

public class Fleet implements org.xdef.component.XComponent{
  public java.util.List<Vehicle> listOfPersonal () {return _Personal;}
  public java.util.List<Vehicle> listOfVan() {return _Van;}
  public void addPersonal (Vehicle x) {
    if (x != null) {
        if (x.xGetXPos() == null)
          x.xInit(this, "Personal", null, "Fleet#Fleet/Personal");
      _Personal.add(x);
    }
  }
  public void addVan(Vehicle x) {
    if (x != null) {
        if (x.xGetXPos() == null)
          x.xInit(this, "Van", null, "Fleet#Fleet/Van");
      _Van.add(x);
```

```
    }
  }
  private final java.util.List<Vehicle> _Personal = new java.util.ArrayList<Vehicle>();
  private final java.util.List<Vehicle> _Van = new java.util.ArrayList<Vehicle>();
  // Constructors and implementation of the interface XComponent ...
}
```

3.3.5 Command %enum

If the X-definition the data type enum is specified, its value in the X-component is represented by default as a String. However, in case we want to have a choice in the code only from the allowed values, the data value of the enum is possible to generate as Java enum type. The data type must be defined in the Script section <xd:declaration>. Enum will be generated by using the command %enum followed by the fully qualified name of the enum class and the name of the datatype.

Example

```
<xd:def xmlns:xd="http://www.xdef.org/3.2" xd:name="Vehicle2" xd:root="Vehicle2">

    <Vehicle2
          VIN   = "required string()"
          Color = "required color()"
    />

    <xd:declaration>
        type color enum('white', 'blue', 'green');
    </xd:declaration>

    <xd:component>
        %enum cz.syntea.tutorial.Color color;
        %class cz.syntea.tutorial.Vehicle2 %link Vehicle2#Vehicle2;
    </xd:component>
</xd:def>
```

```
package cz.syntea.tutorial;
public enum Color implements org.xdef.component.XCEnumeration {
        white,
        blue,
        green;
        @Override
        public final Object itemValue() {return name();}
        @Override
        public final String toString() {return name();}
        public static final Color toEnum(final Object x) {
                if (x != null)
                        for(Color y: values())
                                if (y.itemValue().toString().equals(x.toString())) return y;
                return null;
        }
}
```

```
package cz.syntea.tutorial;

public class Vehicle2 implements org.xdef.component.XComponent{
  public cz.syntea.tutorial.Color getColor() {return _Color;}
  public String getVIN() {return _VIN;}
  public void setColor(cz.syntea.tutorial.Color x) {_Color = x;}
  public void setVIN(String x) {_VIN = x;}
  private cz.syntea.tutorial.Color _Color;
  private String _VIN;
  // Constructors and implementation of the interface XComponent ...
}
```

4 How to work with the X-components

4.1 Generation of X-components

To generate X-components from X-definitions you need the object org.xdef.XDPool (you can create it for instance, by the method org.xdef.XDFactory.compileXD or to load the compiled XDPool from a file by the method

org.xdef.XDFactory.readXDPool etc.). The X-components are generated using the method org.xdef.component.GenXComponent.genXComponent():

```
XDPool xdPool = XDFactory.compileXD(null, "resources/manual/Vehicle.xdef");
GenXComponent.genXComponent(
                xdPool,                                          // XDPool
                new File("src/main/java").getAbsolutePath(),  // directory where to generate
                null                                             // character set (default UTF-8)
        );
```

The code above generates the source code of X-components defined in the passed XDPool. After the compilation of the generated Java classes, you can use them.

4.2 Create an instance of X-component from XML data (unmarshalling)

XDPool is fully reentrant object, so it can be saved as a static final variable that is statically initiated and therefore it is accessible to all programs that work with it.

Example of creating an X-component from XML:

```
public static final XDPool XD_POOL = XDFactory.compileXD(null, "/manual/Vehicle.xdef");
...

ArrayReporter reporter = new ArrayReporter();
File source = new File("src/main/resources/manual/Vehicle.xml");
...

// 1. Create XDDocument
XDDocument doc = XD_POOL.createXDDocument("Vehicle");
// 2. Create X-component with the method parseXComponent().
XComponent xc = doc.parseXComponent(source, Vehicle.class, reporter);
// 3. The method parseXComponent returns the X-component created according to the command %class.
//    We can cast it therefore to the class defined by the %class command and to use
//    the getters and setters: getMujObjekt(), ... setMujObjekt(...) ...
Vehicle vehicle = (Vehicle) xc;
```

4.3 Create XML from X-component (unmashalling)

From the X-component, it is possible to create an XML element that corresponds to the values that are in the X-component. To do this, use the method toXml():

```
Element el = xc.toXml();
```

4.4 Transformation to another X-component

X-components allow you to create a new X-component that has a different structure (a variant of the construction mode) from the particular X-component.

Example:

```
XComponent oldxc;                                    // original X-component
...
XDPool xp;                                           // given XDPool
XComponent newxc = XComponentUtil.toXComponent(stara, // new X-component
    xp,                                              // XDPool
    "def#model");                                    // XDPosition of the model in the new X-component
```

5 Create X-components and work with them

Let us have X-definition, which describes a town in which there are streets and houses in which are the tenants (the file src/data/Town.xdef):

```
<xd:def xmlns:xd="http://www.xdef.org/3.2"
    xd:name="Town"
    xd:root="Town " >

    <Town Name = "required string()">
        <Street xd:script="occurs 0..;"
                Name = "required string()" >
```

```
            <House xd:script="occurs 0..; ref House" />
        </Street>
    </Town>

    <House Num     = "required int()"
           Address = "optional string()">
        <Person xd:script="occurs 0..; ref Person" />
    </House>

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

</xd:def>
```

XML data are in the file "src/data/Town.xml" (processed by X-definition "Town"):

```
<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>
```

TheX-definition in the XML file " src/Town_XC.xdef" describes generation of X-components and of the interface "Citizen":

```
<xd:def xmlns:xd="http://www.xdef.org/3.2"
   xd:name="Town-XC">

    <xd:component>
        /*****************************************************
        *   X-komponents generated from the X-definition Town  *
        *****************************************************/
        %class town.Town %link Town#Town;
        %class town.House %link Town#House;

        /* Create the interface according to the X-component from Town#Person */
        %interface town.Citizen %link Town#Person;
        /* The X-component has the specified interface */
        %class town.Tenant implements town.Citizen %link Town#Person;
    </xd:component>
</xd:def>
```

Let's show how to create the file in which the XDPool with the compiled X-definitions is saved. Then we'll generate the X-components from it:

```
public class Example_XC {
    public static void main(String[] args) throws Exception {
       // 1. Compile X-definitions
        XDPool xdPool = XDFactory.compileXD(null,       // null -> use System properties as default
            "src/data/*.xdef");                         // X-definitions

        // 2. save XDPool to the file
        xdPool.writeXDPool(new File("resources/XDPool.dat"));

        // 3. generate X-components
        GenXComponent.genXComponent(xdPool,     // XDPool
            new File("src").getAbsolutePath(),  // directory where to generate
            null);                              // character set (default UTF-8)
        }
}
```

Now we can write the program that will use the X-component. We first set the static variable XP (using the method getXDPool). From the input file "resources/manual/Town.xml" we create the instance of X-component Town and then we print its contents. Finally we add the addresses of houses the the X-component and we write it to the file manual/Town-processed. xml:

```
public class Example_XC1 {
    // read compiled XDPool from the file to XP
    public static final XDPool XP;
    static {
        try {
            XD = XDFactory.readXDPool("resources/XDPool.dat");
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }

    public static void main(String[] args) throws Exception {
        // 1. create XDDocument
        XDDocument xd = XD.createXDDocument("Town");

        // 2. create the instance of X-component Town (unmarchall) from XML data
        File townFile = new File("src/Example_XC1.xml");
        Town town = (Town) xd.parseXComponent(townFile, town.Town.class, null);

        // 3. Print contents of the X-component Town
        System.out.println("Town " + town.getName());
        for (Town.Street street: town.listOfStreet()) {
            System.out.println("Street " + street.getName() + ":");
            for (House house: street.listOfHouse()) {
                System.out.print("House " + house.getNum() + ". ");
                if (dum.listOfPerson().size() > 0) {
                    System.out.println("Tenants :");
                    for (Person person: house.listOfPerson()){
                        System.out.println(person.getFirstName() + " " + person.getLastName());
                    }
                } else {
                    System.out.println("No tenants in this house");
                }
            }
        }

        // 4. Add to each house the address.
        for (Town.Street street: town.listOfStreet()) {
            for (House house: ulice.listOfHouse()) {
                house.setAddress(town.getName() + ", " + street.getName() + " " + house.getNum());
            }
        }

        // 5. Save the XML document with address to the file Town_processed.xml (marshall)
        Element el = town.toXml();
        KXmlUtils.writeXml("resources/Town_processed.xml", el, true, false);
    }

}
```

5.1 Example of transformation

Let's try to define the X-definition describing another XML data containing a list of tenants in the city and let us use the clause `create` to describe how to create a new transformed file (according to the X-definition "Tenants ") from the input data:

```
<xd:def xmlns:xd="http://www.xdef.org/3.2"
        xd:name = "Tenants"
        xd:root = "Tenants">

    <Tenants>
        <Tenant xd:script="occurs 0..; create from('//Person');"
              GivenName  = "required string(); create from('@FirstName')"
              FamilyName = "required string(); create from('@LastName')"
              Address    = "required string(); create from('../@Address')" />
    </Tenants>
```

```
    <xd:component>
        /*****************************************
        *  X-components of the X-definition Tenants *
        *****************************************/
        %class cz.syntea.tutorial.Tenants %link Tenants#Tenants;

        %bind FirstName %link Tenants#Tenants/Tenant/@GivenName;
        %bind LastName  %link Tenants#Tenants/Tenant/@FamilyName;
    </xd:component>

</xd:def>
```

Transformation of the X-component Town according to the model Tenants by the method org.xdef.component.CompomentUtil.toXComponent(...):

```
public class Example_XC2 {

    public static void main(String[] args) throws Exception {
        // 1. create XDPool a XDDocument
        XDDocument xd = Example_XC1.XD.createXDDocument("Town");

        // 2. create the instance of the X-component Town (unmarchall)
        File townFile = new File("resources/Town_processed.xml");
        Town town = (Town) xd.parseXComponent(townFile, Town.class, null);

        // 3. create transformation to the X-component Tenants
        Tenants tenants =
           (Tenants) XComponentUtil.toXComponent(town, Example1.XD, "Residents#Residents");

        // 4. save data to the file Residents.xml
        Element el = tenants.toXml();
        KXmlUtils.writeXml("resources/Residents.xml", el, true, false);

        // 5. print the list of tenants
        for (Tenants.Resident x: tenants.listOfResident()) {
            System.out.println(x.getFirstName() + " " + x.getLastName() + "; " + x.getAddress());
        }
    }
}
```

The source code of examples above is distributed together with downloaded X-definition files:

*http://www.xdefinice.cz/en/*download/