X-komponenty 3.1

Uživatelská příručka

Napsal: **J. Kocman, V. Trojan**

Verze: **3.1**
Ze dne: **01.11.2017**

Soupis změn

Výchozí verze

Obsah

1 Úvod

Tento dokument je uživatelskou příručkou popisující základní použití technologie "X-komponent". Je určen pro programátory, kteří potřebují převádět data z XML do objektů a zpět. X-komponenty jsou nadstavbou nad X-definicemi a rozšiřují její možnosti o další způsob práce s XML. Pro použití X-komponent je nutné mít základní znalosti o X-definicích.

Upozornění: *Tento manuál se týká implementace X-komponent v X-definicích verze 3.1 build 3.1.2.5*

Dotazy, poznámky a hlášení chyb poslílejte na adresu: *xdef@syntea.cz*.

Aktuální verzi X-definic můžete stáhnout na adrese: *http://www.xdef.cz*.

1.1 Používané pojmy a zkratky

X-definice	1. Jazyk, který umožňuje popsat strukturu, obsah a zároveň i zpracování obsahu XML objektů. 2. XML element jehož obsah je zapsán v jazyce X-definic.
X-komponenty	Technologie pro vytváření javovských tříd na základě struktury X-definic.
X-komponenta	Javovská třída či objekt, který byl vytvořen technologií X-komponent.
XComponent	Javovské rozhraní `cz.syntea.xc.XComponent`, které implementují všechny X-komponenty.
Unmarshalling	Naplnění objektů daty z XML. Opak marshallingu.
Marshalling	Převod dat z objektů do XML. Opak unmarshallingu.
xd:component	XML element v X-definici obsahující popis X-komponent.

2 X-komponenty

X-komponenta je vygenerovaná javovská třída, která svou strukturou odpovídá určitému modelu X-definice. Každému elementu v X-definici odpovídá jedna X-komponenta. Hodnoty atributů, elementů a textových hodnot jsou v X-komponentě přístupné pomocí metod get*Nazev*() a set*Nazev*(), kde *Nazev* je jméno atributu nebo elementu v X-definici. Převod mezi hodnotou atributu nebo textové hodnoty v X-definici a objektem v Javě je dán následující tabulkou.

X-definice	X-komponenta
boolean	`java.lang.Boolean`
short, int, long	`java.lang.Long`
float, double	`java.lang.Double`
dec, decimal	`java.math.BigDecimal`
datetime, xdatetime	`cz.syntea.xdef.sys.SDatetime`
duration	`cz.syntea.xdef.sys.SDuration`
base64Binary, hexBinary	`byte[]`
enum	`Java.lang.String` nebo `Enum` v Javě
ostatní	`java.lang.String`

Všechny typy jsou objekty, což umožňuje u nepovinných atributů testovat, zda při převodu X-komponenty do XML mají být generovány či nikoliv. Pokud bude mít element více opakování, vytvoří se pole hodnot (implementované pomocí `java.util.List`).

Příklad

Mějme vzorovou X-definici popisující pojistnou smlouvu:

```
<Smlouva
    VIN           = "required int()"
    DatumUzavreni = "required datetime('yyyyMMdd')">
  <Vlastnik
    NazevFirmy    = "required string()"
    ICO           = "required num(8)" />
  <Provozovatel  xd:script="occurs 1.."
    RC            = "required num(10)" />
</Smlouva>
```

Vygenerovaná X-komponenta bude vypadat následovně (konstruktory a metody z rozhraní cz.syntea.xdef.component.XComponent nejsou zobrazeny):

```
package cz.syntea.user.kocman.manual;

public class Smlouva implements cz.syntea.xdef.component.XComponent{
  public cz.syntea.xdef.sys.SDatetime getDatumUzavreni() {return _DatumUzavreni;}
  public java.util.Date dateOfDatumUzavreni() {
    return cz.syntea.xdef.sys.SDatetime.getDate(_DatumUzavreni);
  }
  public java.sql.Timestamp timestampOfDatumUzavreni() {
    return cz.syntea.xdef.sys.SDatetime.getTimestamp(_DatumUzavreni);
  }
  public java.util.Calendar calendarOfDatumUzavreni() {
    return cz.syntea.xdef.sys.SDatetime.getCalendar(_DatumUzavreni);
  }
  public Long getVIN() {return _VIN;}
  public Smlouva.Vlastnik getVlastnik() {return _Vlastnik;}
  public java.util.List<Smlouva.Provozovatel> listOfProvozovatel() {return _Provozovatel;}
  public void setDatumUzavreni(cz.syntea.xdef.sys.SDatetime x) {_DatumUzavreni = x;}
  public void setDatumUzavreni(java.util.Date x) {
    _DatumUzavreni=x==null ? null : new cz.syntea.xdef.sys.SDatetime(x);
  }
  public void setDatumUzavreni(java.sql.Timestamp x) {
    _DatumUzavreni=x==null ? null : new cz.syntea.xdef.sys.SDatetime(x);
  }
  public void setDatumUzavreni(java.util.Calendar x) {
    _DatumUzavreni=x==null ? null : new cz.syntea.xdef.sys.SDatetime(x);
  }
  public void setVIN(Long x) {_VIN = x;}
```

```
  public void setVlastnik(Smlouva.Vlastnik x) {_Vlastnik = x;}
  public void addProvozovatel(Smlouva.Provozovatel x) {
    if (x!=null) _Provozovatel.add(x);
  }
  private cz.syntea.xdef.sys.SDatetime _DatumUzavreni;
  private Long _VIN;
  private Smlouva.Vlastnik _Vlastnik;
  private final java.util.List<Smlouva.Provozovatel> _Provozovatel =
    new java.util.ArrayList<Smlouva.Provozovatel>();
  //Konstruktory a implementace rozhraní XComponent

public static class Vlastnik implements cz.syntea.xdef.component.XComponent{
  public String getICO() {return _ICO;}
  public String getNazevFirmy() {return _NazevFirmy;}
  public void setICO(String x) {_ICO = x;}
  public void setNazevFirmy(String x) {_NazevFirmy = x;}
  public String xposOfICO(){return XD_XPos + "/@ICO";}
  public String xposOfNazevFirmy(){return XD_XPos + "/@NazevFirmy";}
  private String _ICO;
  private String _NazevFirmy;
  //Konstruktory a implementace rozhraní XComponent
}
public static class Provozovatel implements cz.syntea.xdef.component.XComponent{
  public String getRC() {return _RC;}
  public void setRC(String x) {_RC = x;}
  public String xposOfRC(){return XD_XPos + "/@RC";}
  private String _RC;
  //Konstruktory a implementace rozhraní XComponent
}
}
```

2.1 Instance X-komponenty

Každá X-komponenta implementuje rozhraní `cz.syntea.xdef.component.XComponent`, které umožnuje převod z a do XML. Instance X-komponenty může být vytvořena při parsování XML podle modelu v X-definici, kdy dojde i k jejímu naplnění příslušnými daty z XML (operace unmarshalling). X-komponentu lze vytvořit i pomocí bezparametrického konstruktoru a daty ji naplnit ručně. Z instance X-komponenty je možné vytvořit XML element zavoláním metody `cz.syntea.xdef.component.XComponent.toXml()` (operace marshalling).

2.2 XDPozice

XDPozice je popis místa v kolekci X-definic (XDPoolu). Skládá se ze jména X-definice, po kterém následuje „#“ a jméno modelu. Za modelem může následovat znak „/“ a buď jméno vnořeného elementu, „text()“ pro textovou hodnotu, nebo „@“ a jméno atributu. Pokud mají jména prefix, tak se zapisují s prefixem tak, jak byla uvedena v X-definici. Pokud se v množině potomků elementu vícekrát vyskytuje uzel se stejným jménem, doplní se do hranatých závorek pořadové číslo (čísluje se od jedničky „[1]“). Pokud se pořadí neuvede, bude se brát pouze první element („[1]“ se nemusí zapisovat). V následujícím příkladě jsou pro ukázku uvedeny XDPozice všech prvků v X-definici:

```
<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>
```

2.3 Příkazy pro generování X-komponent

Příkazy pro tvorbu X-komponent se zapisují do sekce `<xd:component>`, která musí být uvedena v X-definici. Sekce `<xd:component>` může být v samostatné X-definici nebo může být součástí libovolné jiné X-definice. Každý příkaz je ukončen znakem „;“.

2.3.1 Příkaz %class

Příkazem `%class` se definuje Javovská třída, která se má vygenerovat podle určitého elementu v X-definicích. Za klíčovým slovem `%class` následuje plně kvalifikované jméno třídy a klíčové slovo `%link`, které určuje pozici modelu v XDPoolu, pro který se má daná X-komponenta vygenerovat. Pokud má X-komponenta rozšiřovat nějakou třídu, či implementovat rozhraní, lze plně kvalifikované jméno rozšířit o `extends` *`JmenoSuperClass`* `implements` *`JmenoInterface`*, kde *`JmenoSuperClass`* a *`JmenoInterface`* musí být znovu plně kvalifikovaná jména. Syntaxe je stejná jako v deklaraci třídy definované v zdrojovém kódu Java.

Příklad 1

Zdrojová X-definice:

```
<xd:def  xmlns:xd="http://www.syntea.cz/xdef/3.1" xd:name="Vozidlo" xd:root="Vozidlo">

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

    <xd:component>
        %class cz.syntea.tutorial.Vozidlo
               extends cz.syntea.tutorial.PreVozidlo
               implements cz.syntea.tutorial.IVozidlo
           %link Vozidlo#Vozidlo;
    </xd:component>
</xd:def>
```

Vygenerovaná X-komponenta:

```
package cz.syntea.tutorial;

public class Vozidlo extends cz.syntea.tutorial.PreVozidlo
  implements cz.syntea.tutorial.IVozidlo, cz.syntea.xdef.component.XComponent{
  public String getVIN() {return _VIN;}
  public void setVIN(String x) {_VIN = x;}
  private String _VIN;
  //Konstruktory a implementace rozhraní XComponent
}
```

2.3.2 Příkaz %bind

Jméno objektu (atributu, textové hodnoty či elementu a odpovídající jméno getterů a setterů) lze nastavit příkazem `%bind`. Po klíčovém slově `%bind` následuje jméno proměnné, které bude použito místo automaticky vygenerovaného. Dále následuje příkaz `%from`, po kterém jsou uvedeny XDPozice všech prvků oddělených čárkou, kterých se příkaz týká. Stejné jméno může být použito ve více modelech. Gettery a settery budou automaticky upraveny tak, aby odpovídaly nově přiřazenému jménu (viz Příklad 1).
Pokud vygenerovaná třída má předka, lze pomocí `%bind` navázat objekt na getter a setter definovaný v předkovi. V tomto případě se daná proměnná včetně getterů a setterů nebude vůbec generovat a bude využívat implementaci těchto metod předka (viz Příklad 2).

Příklad 2

Máme nákladní vozidlo, které rozšiřuje vozidlo z předchozího příkladu. Vygenerovaná X-komponenta nebude obsahovat gettery a settery pro položky, které se zdědily:

```
<xd:def  xmlns:xd="http://www.syntea.cz/xdef/3.1"
    xd:name="Nakladni"
    xd:root="Nakladni">

    <Nakladni xd:script = "ref Vozidlo#Vozidlo"
          MaxNaklad = "required int()"
    />

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

Vygenerovaná X-komponenta.

```
package cz.syntea.tutorial;
public class Nakladni extends cz.syntea.tutorial.Vozidlo
  implements cz.syntea.xdef.component.XComponent{
  public Long getMaxNaklad() {return _MaxNaklad;}
  public void setMaxNaklad(Long x) {_MaxNaklad = x;}
  private Long _MaxNaklad;
  //Konstruktory a implementace rozhraní XComponent
}
```

2.3.3 Příkaz %interface

Často se vyskytuje případ, kdy (koncový) model přebírá strukturu jiného (referenčního) modelu a případně ji i doplňuje o další atributy, textové hodnoty či elementy. Aby se X-komponenty vygenerované z koncových modelů mohly chovat jako X-komponenta vytvořená z referenčního, lze z daného modelu vytvořit interface. Tento interface pak lze přidat u generování koncových modelů (viz 2.3.1). Interface se vytvoří klíčovým slovem `%interface`, za kterým je plně kvalifikované jméno interfaceu a klíčové slovo `%link` s pozicí modelu v XDPoolu.

Příklad 3

Osobní automobil má stejnou část struktury jako Vozidlo. Tato struktura byla vytažena ven a element OsobniVozidlo se na ni odkazuje.

```
<xd:def  xmlns:xd="http://www.syntea.cz/xdef/3.1"
    xd:name="Osobni"
    xd:root="Osobni">

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

    <Osobni xd:script = "ref Vozidlo"
            Pocetmist = "required int()"
    />

    <xd:component>
        %interface cz.syntea.tutorial.IVozidlo %link Osobni#Vozidlo;
        %class cz.syntea.tutorial.Osobni
                implements cz.syntea.tutorial.IVozidlo
            %link OsobniVozidlo#Osobni;
    </xd:component>
</xd:def>
```

Vygenerovaný interface pak má definovány veškeré gettery a settery. X-komponenta Osobni je vygenerovány standardně jako v předchozích příkladech, pouze implementuje rozhraní IVozidlo.

```
package cz.syntea.tutorial;

public interface IVozidlo extends cz.syntea.xdef.component.XComponent {
  public String getVIN();
  public void setVIN(String x);
  public String xposOfVIN();
}
```

```
package cz.syntea.tutorial;

public class Osobni implements cz.syntea.tutorial.IVozidlo,cz.syntea.xdef.component.XComponent{
  public Long getPocetmist() {return _Pocetmist;}
  public String getVIN() {return _VIN;}
  public void setPocetmist(Long x) {_Pocetmist = x;}
  public void setVIN(String x) {_VIN = x;}
  private Long _Pocetmist;
  private String _VIN;
  //Konstruktory a implementace rozhraní XComponent
}
```

2.3.4 Příkaz %ref

Často se stává, že XDPool je generován z různého množství X-definic. V případě, že X-komponenta je vygenerována danou X-definicí, ale XDPool je jiný (například tam jsou nějaké X-definice navíc), lze se na již vytvořenou X-komponentu odvolat a zamezit jejímu opětovnému vygenerování (například pokud se daná X-komponenta nachází v jiném jaru). K odkazu na již vygenerovanou X-komponentu se používá příkaz `%ref` s plně kvalifikovaným jménem již vygenerované X-komponenty a klíčovým slovem `%link` s pozicí modelu v XDPoolu.

Příklad

V jar souboru, který máme v classpath, je již vygenerována X-komponenta `cz.syntea.test.prexd.Vozidlo` (z Příkladu 1). X-definici, která tuto X-komponentu vygenerovala, máme zahrnutou v XDPoolu. Pak při tvorbě nových X-komponent se pak použije již vygenerovaná X-komponenta a negeneruje se znovu.

```
<xd:def xmlns:xd="http://www.syntea.cz/xdef/3.1"
    xd:name="Flotila"
    xd:root="Flotila">

    <Flotila>
        <Osobni  xd:script = "ref Vozidlo#Vozidlo; occurs 0.."/>
        <Dodavka xd:script = "ref Vozidlo#Vozidlo; occurs 0.."/>
    </Flotila>

    <xd:component>
        %ref cz.syntea.tutorial.Vozidlo   %link Vozidlo#Vozidlo;
        %class cz.syntea.tutorial.Flotila %link Flotila#Flotila;
    </xd:component>
```

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

```
package cz.syntea.tutorial;

public class Flotila implements cz.syntea.xdef.component.XComponent{
  public java.util.List<Vozidlo> listOfOsobni () {return _Osobni;}
  public java.util.List<Vozidlo> listOfDodavka() {return _Dodavka;}
  public void addOsobni (Vozidlo x) {
    if (x != null) {
        if (x.xGetXPos() == null)
          x.xInit(this, "OsobniAutomobil", null, "Flotila#Flotila/Osobni");
      _Osobni.add(x);
    }
  }
  public void addDodavka(Vozidlo x) {
    if (x != null) {
        if (x.xGetXPos() == null)
          x.xInit(this, "Dodavka", null, "Flotila#Flotila/Dodavka");
      _Dodavka.add(x);
    }
  }
  private final java.util.List<Vozidlo> _Osobni = new java.util.ArrayList<Vozidlo>();
  private final java.util.List<Vozidlo> _Dodavka = new java.util.ArrayList<Vozidlo>();
  //Konstruktory a implementace rozhraní XComponent
}
```

2.3.5 Příkaz %enum

Pokud je v X-definici použit datový typ enum nebo list, převádí se do X-komponent standardně jako String.
V případě, že chceme i v kódu mít na výběr pouze z povolených hodnot, lze z datového typu enum a list vytvořit enum v kódu Javy. Datový typ musí být definovaná mimo model v sekci <xd:declaration>. Enum se vygeneruje pomocí příkazu %enum následovaný plně kvalifikovaným jménem třídy a názvem datového typu.

Příklad

```
<xd:def xmlns:xd="http://www.syntea.cz/xdef/3.1" xd:name="Vozidlo2" xd:root="Vozidlo2">

    <Vozidlo2
          VIN   = "required string()"
          Barva = "required barva()"
    />

    <xd:declaration>
        type barva enum('bila', 'modra', 'zelena');
    </xd:declaration>

    <xd:component>
        %enum cz.syntea.tutorial.Barva barva;
        %class cz.syntea.tutorial.Vozidlo2 %link Vozidlo2#Vozidlo2;
    </xd:component>
</xd:def>
```

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

```
package cz.syntea.tutorial;
```

```
public class Vozidlo2 implements cz.syntea.xdef.component.XComponent{
  public cz.syntea.tutorial.Barva getBarva() {return _Barva;}
  public String getVIN() {return _VIN;}
  public void setBarva(cz.syntea.tutorial.Barva x) {_Barva = x;}
  public void setVIN(String x) {_VIN = x;}
  private cz.syntea.tutorial.Barva _Barva;
  private String _VIN;
  //Konstruktory a implementace rozhraní XComponent
}
```

Poznámka - některé příkazy v X-definici ovlivňují i vytvořenou X-komponentu:

- ignore – takto označené prvky X-definice při zpracování ignoruje. Nevytváří se ani odpovídající X-komponenta nebo getter a setter
- forget – elementy se po zpracování vymažou z paměti i ze zpracovávaného dokumentu. Odpovídající X-komponenta se ale vytvoří

3 Jak pracovat s X-komponentami

3.1 Generování X-komponent

Abychom vygenerovali X-komponenty podle X-definic, musíme mít vytvořen objekt `cz.syntea.xdef.XDPool`. Ten se může vytvořit např. metodou `cz.syntea.xdef.XDFactory.compileXD()`. Druhou možností je XDPool je načíst již zkompilovaný XDPool (metoda `cz.syntea.xdef .XDFactory.readXDPool()`). X-komponenty se vygenerují pomocí metody `cz.syntea.xdef.component.GenXComponent.genXComponent()` (viz následující ukázka kódu).

```
XDPool xdPool = XDFactory.compileXD(null, "resources/manual/Vozidlo.xdef");
GenXComponent.genXComponent(
                xdPool,                                          // XDPool
                new File("src/main/java").getAbsolutePath(),  // adresář kam se bude generovat
                null                                             // znakova sada  (defaultne UTF-8)
        );
```

Pomocí shora vytvořeného progamu vygenerujeme zdrojové kódy X-komponent definované v předaném XDPoolu. Po překladu Java kompilátorem vzniknou třídy, které můžeme používat.

3.2 Vytvoření instance X-komponenty z XML (unmarshalling)

XDPool je plně reentrantní objekt, proto je vhodné jej uložit jako statickou finální proměnnou, která je staticky iniciovaná a je přístupná pro všechny programy, které s ní pracují. Např. příkazem:

Příklad vytvoření X-komponenty z XML:

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

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

// 1. Vytvoříme obvyklým způsobem objekt XDDocument
XDDocument doc = XD_POOL.createXDDocument("Vozidlo");
// 2. X-komponentu vytvoříme metodou parseXComponent().
XComponent xc = doc.parseXComponent(source, Vozidlo.class, reporter);
// 3. Metoda parseXComponent vrací instanci XComponent vyvořenou podle příkazu %class.
//    Můžeme ji tedy přetypovat na třídu, kterou jsme definovali příkazem %class a pak
//    používat příslušné gettery a settery: getMujObjekt(), ... setMujObjekt(...) ...
Vozidlo vozidlo = (Vozidlo) xc;
```

3.3 Vytvoření XML z X-komponenty (unmashalling)

Z X-komponenty je možné vytvořit XML element, který odpovídá hodnotám, které jsou v instanci X-komponenty. K tomu slouží příkaz toXml():

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

3.4 Transformace na jinou X-komponentu

X-komponenty umožňují vytvořit z X-kompnenty novou X-komponentu, která má jinou strukturu (obdoba konstrukčního režimu). K tomu slouží statická metoda `toXComponent` vve třídě `XComponentUtil`.

Příklad:

```
XComponent stara;                                       // původní X-komponenta
...
XDPool xp;                                              // použitý XDPool
XComponent newxc = XComponentUtil.toXComponent(stara, // původní X-kompnenta
    xp,                                                 // XDPool
    "def#model");                                       // XDPozice modelu nové X-komponenty
```

4 Příklad

4.1 Vytvoření X-komponent a práce s nimi

Mějme X-definici, ktera popisuje město, ve kterém jsou ulice a domy v nichž jsou nájemníci (soubor manual/Town.xdef):

```
<xd:def xmlns:xd="http://www.syntea.cz/xdef/3.1"
    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>
```

Data jsou uložena v následujícím XML (soubor manual/Town.xml):

```
<Town Name="Nicovice">
  <Street Name="Dlouha">
    <House Num="1">
      <Person FirstName="Jan" LastName="Novak"></Person>
      <Person FirstName="Jana" LastName="Novakova"></Person>
    </House>
    <House Num="2"/>
    <House Num="3">
      <Person FirstName="Josef" LastName="Novak"></Person>
    </House>
  </Street>
  <Street Name="Kratka">
    <House Num="1">
      <Person FirstName="Pavel" LastName="Novak"></Person>
    </House>
  </Street>
</Town>
```

X-definice s popisem X-komponent (soubor manual/Town-XC.xdef):

```
<xd:def xmlns:xd="http://www.syntea.cz/xdef/3.1" xd:name="Town-XC">

    <xd:component>
        /************************************************
        *     X-komponenty generovane z X-definice Town   *
        ************************************************/
        %class cz.syntea.tutorial.Mesto %link Town#Town;
        %class cz.syntea.tutorial.Dum %link Town#House;

        /* Vytvorime interface, který odpovídá X-komponente z Town#Person */
        %interface cz.syntea.tutorial.Obcan %link Town#Person;
        /* X-komponenta bude mit implementovan tento interface */
        %class cz.syntea.tutorial.Osoba implements cz.syntea.tutorial.Obcan %link Town#Person;
    </xd:component>
</xd:def>
```

Z uvedených X-definic X-komponenty vygenerujeme binární soubor, ve kterém bude uložen XDPool a dále vygenerujeme soubory s java source X-komponent:

```
public static void main(String[] args) throws IOException {
    XDPool xdPool = XDFactory.compileXD(null,
            "data/manual/Town.xdef",
            "data/manual/Town-XC.xdef"
        );
    xdPool.writeXDPool(new File("resources/manual/XDPool.dat"));
    GenXComponent.genXComponent(
            xdPool,                                         // XDPool
            new File("src/main/java").getAbsolutePath(),  // adresář kam se bude generovat
            null                                            // znakova sada  (defaultne UTF-8)
    );
}
```

Nyní můžeme napsat program, bude používat X-komponenty. XDPool uložíme do statické proměnné XP (ze třídy Pool metodou getXDPool). Ze vstupního souboru manual/Town.xml vytvoříme instanci X-komponenty Mesto a vytiskneme její obsah. Pak doplníme adresu každého domu a XML zapíšeme do souboru manual/Town-processed.xml:

```
public class Ukazka1 {
    // načteme již zkompilovaný XDPool
    public static final XDPool XD_POOL;
    static {
        try {
            XD_POOL = XDFactory.readXDPool("resources/manual/XDPool.dat");
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }

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

        // 2. vytorime instanci X-komponenty Mesto (unmarchall)
        File mestoFile = new File("src/main/resources/manual/Town.xml");
        Mesto mesto = (Mesto) xd.parseXComponent(mestoFile, Mesto.class, null);

        // 3. Vytiskneme obshah X-komponenty Mesto
        System.out.println("Mesto " + mesto.getName());
        for (Mesto.Street ulice: mesto.listOfStreet()) {
            System.out.println("Ulice " + ulice.getName() + ":");
            for (Dum dum: ulice.listOfHouse()) {
                System.out.print("Dum c. " + dum.getNum() + ". ");
                if (dum.listOfPerson().size() > 0) {
                    System.out.println("Najemnici :");
                    for (Obcan osoba: dum.listOfPerson()){
                        System.out.println(osoba.getFirstName()
                                + " " + osoba.getLastName());
                    }
                } else {
                    System.out.println("Dum nema najemniky");
                }
            }
        }

        // 4. Doplnime adresu ke kazdemu domu.
        for (Mesto.Street ulice: mesto.listOfStreet()) {
            for (Dum dum: ulice.listOfHouse()) {
                dum.setAddress(mesto.getName() + ", " + ulice.getName()
                        + " " + dum.getNum());
            }
        }

        // 5. Ulozime XML s doplnenymi adresami do souboru Town_processed.xml (marshall)
        Element el = mesto.toXml();
        KXmlUtils.writeXml("resources/manual/Town_processed.xml", el, true, false);
    }

}
```

4.2 Ukázka transformace

Zkusme definovat pomocí X-definice nové XML, obsahující seznam nájemníků ve městě a popišme pomocí klauzule `create` popsat, jak se má vytvořit ze vstupních dat (soubor Residents.xdef):

```
<xd:def xmlns:xd="http://www.syntea.cz/xdef/3.1"
        xd:name = "Residents"
        xd:root = "Residents">

    <Residents>
        <Resident 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')" />
    </Residents>
</xd:def>
```

Nyní popíšeme X-komponenty pro práci s X-definicí Residents – (soubor Residents-XC.xdef):

```
<xd:def xmlns:xd="http://www.syntea.cz/xdef/3.1"
        xd:name = "Residents-XC">

    <xd:component>
        /************************************************
        *  X-komponenty generovane z X-definice Residents *
        ************************************************/
        %class cz.syntea.tutorial.Najemnici %link Residents#Residents;

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

</xd:def>
```

Transformaci z komponenty Město do tvaru podle modelu Najemnici provedeme příkazem XCompomentUtil.to`XComponent(...)`:

```
public class Ukazka2 {

    public static void main(String[] args) throws Exception {
        // 1. ziskame XDPool a XDDocument
        XDDocument xd = Ukazka1.XD_POOL.createXDDocument("Town");

        // 2. vytorime instanci X-komponenty Mesto (unmarchall)
        File mestoFile = new File("resources/manual/Town_processed.xml");
        Mesto mesto = (Mesto) xd.parseXComponent(mestoFile, Mesto.class, null);

        // 3. vytvorime transformaci do X-komponent Najemnici
        Najemnici najemnici = (Najemnici) XComponentUtil.toXComponent(
                mesto, Ukazka1.XD_POOL, "Residents#Residents");
        // 4. uloxime do souboru Residents.xml
        Element el = najemnici.toXml();
        KXmlUtils.writeXml("resources/manual/Residents.xml", el, true, false);

        // 5. Vytiskneme najemniky
        for (Najemnici.Resident x: najemnici.listOfResident()) {
            System.out.println(x.getFirstName()
                    + " " + x.getLastName() + "; " + x.getAddress());
        }
    }
}
```