X-definition 4.2

Introduction to the Construction Mode

Author: **Václav Trojan**

Version: **4.2.0.0**
Date: **2022-06-15**

Contents

Tables

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

The prerequisite to reading this text is that the reader is familiar with the validation mode of X-definition and understands the X-definition Script language. Here we describe step by step the construction mode of X-definition. Hopefully, the reader will finally appreciate the benefits and simplicity of this way of creating XML data.

While in the validation mode, the activity of the X-definition processor is controlled by XML input data, whereas in the construction mode, the processor is controlled by the X-definition itself. This means that the output XML data is created according to the information recorded in the X-definition. Thus, instead of processing the input document where the models corresponding to the input data are searched for in X definition (i.e. the process is "controlled" by the input data), in the construction mode X definition is used as a formula for the construction of the result (i.e., the process is in this case "controlled" by the X-definition). Commands that provide the data needed to create the resulting data are specified in the "create" sections of the Script of the models. Let's note in this context that when creating the resulting document, the specification of the "create" section can even be omitted, and then an implicit action is performed to provide the construction of the resulting data. It is important to note that the code of the "create" section is executed only in the construction mode and is ignored in the validation mode. However, in the beginning, we will show the behavior of the X-definitions consistently with the explicit specification of the "create" section of the Script.

Note: Below the examples are the Internet addresses where you can click and run the appropriate X-definition given in the example and even modify it.

Table 1 - Used terms and abbreviation

Term	Explanation
XML	Extensible Markup Language
X-definition	1. XML-based language designed for the description of XML objects 2. XML file containing data written in X-definition language
validation mode	the mode of processing X-definition which processes validation of input XML data
construction mode	the mode of processing X-definition which returns, as a result, XML data (or JSON object or X-component)
quantifier	minimal and maximal occurrence of an item in the model
attribute	XML attribute
element	XML element
text node	the child node of an XML element containing a string value
model	description of XML element in X-definition language
Script	the programming language used in models and declaration parts of the X-definition
context	data internally used for the construction of XML objects
data source	data which are set as the context for the actual model
string	a sequence of characters allowed in XML data
ResultSet	An object which is the result of a relational database query command
NamedValue	Pair of name (a string) and a value
Container	An object containing a sequence of data values and a map of named values
reporter	An object containing messages reported during the processing of X-definition
X-definition processor	the Java program, which implements the processing of X-definition

3 How to run Construction mode in Java

In the Java program, we run the design mode from the "org.xdef.XDDocument " object using the "xcreate" method. The result of this method is the "org.w3c.dom.Element" object. The parameter of this method is the "javax.xml.namespace.QName" object, which refers to the model according to which the result is to be created (or, if the model does not have a namespace URI, it is enough to enter a string with the model name). Of course, the X definition must have such a model. The second parameter of the "xcreate" method is a reporter (the "org.xdef.ReportWriter" object), which writes errors found during the creation process. If the reporter value is "null" and some errors are reported, then the run of the X processor definitions is terminated except "RuntimeException" which is text containing the error messages.

A typical sequence of Java commands to run the construction mode:

```
XDPool xpool = XDFactory. compileXD(...);
XDDocument xd = xpool. createXDDocument(name); /* name of X-definition */
QName qname = new QName(uri, name); /* reference to model */
...
ArrayReporter reporter = new ArrayReporter();
Element elem = xd.xcreate(qname, reporter);
if (reporter.errors()) {
 // processing of errors
}
```

If we want to set the default source data for the construction mode (we speak about "context" - see Chapter 6. The context for XML construction), before calling the method "xcreate" we call the "setXDContext(...)" method with the parameter with an object containing context data:

```
xd.setXDContext("project/data.xml");
xd.xcreate(qname, reporter);
```

4 Explicit Specification of Data Source for Construction

First, we assume that in the construction mode, the "create" section is mandatory and that the objects are created from the values returned by the command in this section.

Let's take the simplest example and ask what may be the result of the command in the corresponding "create" section so that the X-definition processor can create the desired element. In other words, what the processor must know (what value it expects in the "create" section) to construct the "A" element described in the following model:

```
<xd:def xmlns:xd = "http://www.xdef.org/xdef/4.2">
  <A xd:script="create ???"/>
</xd:def>
```

Since "A" does not contain any additional data, it is clear that the only thing the processor needs to know is if the element "A" is to be created or not. In this case, the Boolean value should be enough. The X-definition can thus take the form:

```
<xd:def xmlns:xd = "http://www.xdef.org/xdef/4.2">
   <A xd:script="create true"/>
</xd: def>
```

http://xdef.syntea.cz/tutorial/en/example/C201.html

The result will be:

```
<A/>
```

Simple! Try the Internet link and see for yourself. If the result of the "create" command were "false", the "A" element would not be created (and the processor would report the error that the required element was not created). You can try to modify the text in the window with the X-definition.

However, generally, the "boolean" type of value is not sufficient for all cases. Take a more complicated example. In element "A" we will require a certain number of "B" elements as child nodes of "A":

```
<xd:def xmlns:xd = "http://www.xdef.org/xdef/4.2">
   <A xd:script="create true">
      <B xd:script="occurs *; create ???"/>
   </A>
```

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

If we wrote "true" in the creation section of the "B" element, we would have to wait a long time for the result and the program would end with a memory overflow because the elements corresponding to the occurrence condition would gradually be created. Do not even try it. But we can solve the problem easily. We simply write the number of elements we require. So our X-definition will look like this:

```
<xd:def xmlns:xd = "http://www.xdef.org/xdef/4.2">
   <A xd:script="create 1">
      <B xd:script="occurs *; create 3"/>
   </A>
</xd:def>
```

http://xdef.syntea.cz/tutorial/en/example/C202.html

Thus, three elements "B" and the result will be:

```
<A><B/><B/><B/></A>
```

An attentive reader can also modify the X-definition as follows:

```
<xd:def xmlns:xd = "http://www.xdef.org/xdef/4.2">
   <A xd:script="create true">
      <B xd:script="occurs *; create 3"/>
   </A>
</xd:def>
```

http://xdef.syntea.cz/tutorial/en/example/C203.html

As we have seen, to create an element, we generally need to know the condition if an element is to be created. If the result of the command in the "create" section is a "Container" object or a sequence of items (e.g. from "xpath", "xquery", database table rows, etc.), we can look at this value as an iterator and the individual elements from it are created.

So, another option is to get by the "create" command a set of values according to which the result will be constructed. For example, we can use the "Container" type, which, as we know, may contain a sequence of items. Each item in the sequential part of the Container object is used (if it is not null) as a basis for creating a result. Modify our example by giving a set of items to the "create" section (remember that the entry "[false, 0, 'abc'] is a constructor for the "Container" with three items: false, 0, and the string "abc"):

```
<xd:def xmlns:xd = "http://www.xdef.org/xdef/4.2">
   <A xd:script="create true">
      <B xd:script="occurs *; create [false, 0, 'abc']"/>
   </A>
</xd:def>
```

http://xdef.syntea.cz/tutorial/en/example/C204.html

Three elements "B" are created again. The container has three items, the result will be generated from all values of the Container (even the value "false" or 0) except the "null" value. Therefore the result will be:

```
<A><B/><B/><B/></A>
```

The following X-definitions do not create any element "B" since the Container is empty:

```
<xd:def xmlns:xd = "http://www.xdef.org/xdef/4.2">
   <A xd:script="create true">
      <B xd:script="occurs *; create []"/>
   </A>
</xd:def>
```

http://xdef.syntea.cz/tutorial/en/example/C205.html

The result will be:

```
<A/>
```

It is obvious that the same result is achieved, for example, by the following X-definition:

```
<xd:def xmlns:xd = "http://www.xdef.org/xdef/4.2">
   <A xd:script="create true">
      <B xd:script="occurs *; create [null, null, null]"/>
   </A>
</xd:def>
```

http://xdef.syntea.cz/tutorial/en/example/C206.html

Now, let us ask ourselves what happens if we put an XML element as a data source into the create section. Let's look at the X-definition:

```
<xd:def xmlns:xd = "http://www.xdef.org/xdef/4.2">
   <A xd:script="create new Element('X')"/>
</xd:def>
```

http://xdef.syntea.cz/tutorial/en/example/C207.html

The program has an object, even an XML element. Let's say at the beginning that the processor does not care for the name of the element (except for the element "xd: any" - if the reader goes on reading, he'll see why). Let's have an element "X" that we pass as a data source for the design of the desired element "A". The result will again be:

```
<A/>
```

The reader will surely answer the question of what will be the result from the following X-definition:

```
<xd:def xmlns:xd = "http://www.xdef.org/xdef/4.2">
   <A xd:script="create null">
</xd:def>
```

http://xdef.syntea.cz/tutorial/en/example/C208.html

What happens when the data source is a string? So:

```
<xd:def xmlns:xd = "http://www.xdef.org/xdef/4.2">
   <A xd:script="create 'blabla'">
      required string();
   </A>
</xd:def>
```

http://xdef.syntea.cz/tutorial/en/example/C209.html

Even in this case, we get the "A" element (unless the string value was "null"). Let's examine another interpretation now. The processor produces from the string an auxiliary element that will have a text value taken from the string value. This auxiliary element will then be used as a data source for the construction of the element in a given model (this property makes it easy to create text-based elements). So, the result will be:

```
<A>blabla</A>
```

Let's describe the situation that is one of the most typical variants for real usage. Imagine that we want to create our XML element from another XML element. The source data for the construction of an element can be provided either as a result of an XPath expression or as a result of a database statement.

First, we show you how to proceed using the XPath command. Let's take the following XML element as a data source for the construction of the following result:

```
<X>
  <A/>
  <Y/>
  <Z/>
  <A/>
  <Z/>
</X>
```

The desired result is the element "A" whose descendants "B" will be created from all elements "A" from the data source. We also require the "C" element to be created from the second occurrence of the element "Z". The desired result will therefore be:

```
<A>
  <B/>  (to be created for the first "A")
  <B/>  (to be created for the second "A")
  <C/>  (to be created for the second "Z")
</A>
```

When designing the X definition, we will use the Script method "xpath(...)", with the XPath expression being the first parameter, and as the second parameter, we will use the element on which the XPath expression is to be performed. Let's note that the result of the "xpath" method is an object Container (we have already encountered this variant) created from the NodeList object that is the result of the XPath expression:

```
<xd:def xmlns:xd = "http://www.xdef.org/xdef/4.2">
<xd:declaration>
   <![CDATA[
      /* Create XML Element to "source" variable. */
      Element source = xparse("<X><A/><Y/><Z/><A/><Z/></X>");
   ]]>
</xd:declaration>
<A xd:script="create source"> /* "source" is the value for construction. */
   <B xd:script="occurs 1..*; create xpath('/X/A', source)"/>
   <C xd:script="occurs 1; create xpath('/X/Z[2]', source)"/>
</A>
</xd:def>
```

http://xdef.syntea.cz/tutorial/en/example/C210.html

Element A is created from element X (as we already know, the name does not matter). All elements "B" are created from all the elements "A" in the source and the element "C" is created from the second occurrence of the element "Z". The result will therefore be the element we requested.

Now we ask ourselves how to set the attribute values and the text values to the result. Try to create an element with the text value "Hi World!". What value do we need to create a text node or attribute? It's not hard to guess that we'll need a string value of the value:

```
<xd:def xmlns:xd = "http://www.xdef.org/xdef/4.2">
<A xd:script="create true">
   string; create "Hello world!"
</A>
</xd:def>
```

http://xdef.syntea.cz/tutorial/en/example/C211.html

We see that we simply added the required value to the create section. For the attributes, the situation will be similar to that for text nodes. The result of the following X definition will surely be read by the reader himself:

```
<xd:def xmlns:xd = "http://www.xdef.org/xdef/4.2">
<Person xd:script="create true"
   Name="string(); create 'John'"
   Surname="string(); create 'Smith'">
   string; create "Bad guy."
</Person>
</xd:def>
```

http://xdef.syntea.cz/tutorial/en/example/C212.html

Let us add that if the object with the value is not to be created, we can set the value "null" as the data source (and the "Salary" attribute will not be created):

```
<xd:def xmlns:xd = "http://www.xdef.org/xdef/4.2">
<Person xd:script="create true"
  Name="string(); create 'John'"
  Surname="string(); create 'Smith'"
  Salary="optional int; create null"/>
</xd:def>
```

http://xdef.syntea.cz/tutorial/en/example/C213.html

Note that the data source for the value of the attribute or text node is a string. If we set a different type of data source value, the automatic conversion to string will be performed (the "toString()" method will be executed - unless the present value was null). It is therefore possible to write:

```
<xd:def xmlns:xd = "http://www.xdef.org/xdef/4.2">
<Person xd:script="create true"
  Name="string(); create 'John'"
  Surname="string(); create 'Smith'"
  Salary="optional int; create 0001234"/>
</xd:def>
```

http://xdef.syntea.cz/tutorial/en/example/C214.html

The leading zeros will be lost through conversion and the result will be:

```
<Person FirstName="Jan" Name="John" Surname="Smith" Salary="1234"/>
```

Now, try to create an element whose values will be taken from an element that we pass as a data source:

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

```
<xd:declaration>
  <![CDATA[
   /* Create XML Element to "source" variable. */
   Element source = xparse("<Person Name='John' Surname='Smith' Salary='0001234'/>");
  ]]>
</xd:declaration>

<Person    xd:script = "create source"
           Name      = "string; create xpath('@Name', source)"
           Surname   = "string; create xpath('@Surname', source)">
   <Salary xd:script = "create xpath('@Salary', source)">
          string; create xpath('@Salary', source)
   </Salary>
</Person>
</xd:def>
```

http://xdef.syntea.cz/tutorial/en/example/C215.html

The result will be:

```
<Person Name="John" Surname="Smith">
  <Salary>
    1234
  </Salary>
</Person>
```

Similarly, we can create an element based on the result of a database command. Let's have a database table containing rows with the columns "name", "surname", and "salary". We pass the table to the program as the object "ResultSet" object obtained from the database by the "query" command. The elements are created from the rows of the table with the appropriate columns (note that the parameter of the method "getItem" is case sensitive which is the property of the database commands):

```
<xd:def xmlns:xd = "http://www.xdef.org/xdef/4.2">
<xd:declaration>
   /*
    Access to a database is usually created in the Java program by the command
        DFactory.createSQLService(URL, user, password)
    and passed to the X-Definition over an external variable.
   */
   Service service = new Service("jdbc:derby://localhost:1527/sample;", "myself", "blabla");

   /* Get table from database. */
   ResultSet persons = service.query("SELECT * FROM MYTEST.PERSON");
</xd:declaration>

<List xd:script="create true">
   <Person    xd:script="occurs *; create persons"
              Name     ="string; create persons.getItem('NAME')"
              Surname  ="string; create persons.getItem('SURNAME')">
      <Salary xd:script="create persons.getItem('SALARY') != null">
              string; create persons.getItem('SALARY')
      </Salary>
   </Person>
</List>

</xd:def>
```

The "Person" elements will only be created if the lines in the ResultSet object are available. The "getItem(name)" method returns a string with the value of the appropriate column of the actual row.

You can try the database example on http://xdef.syntea.cz/tutorial/en/example/C230.html

4.1 Overview of value types of the context of the create section

As we have seen one of the following types of values can be used to create elements. Remember that the number of created elements is limited by the maximum specified in the qualifier.

Table 2 - Types of values of the context of an element

Value	Result of construction
null	creates nothing
element	creates elements
int	if positive creates elements according to the number

boolean	elements are created if the value is „true“
String	elements are created if the value is a non-empty string
Container	elements are created if the value is a nonempty Container.
ResultSet	elements are created from rows of the table from ResultSet
other types	value is converted to a String and behaves as String

For attributes or text nodes, the source value is converted to a string and the attribute or the text node is created if the value is not "null" and the string is not empty.

5 Groups

In the following text let's see how the groups "xd:sequence", "xd:choice" and "xd:mixed" behave in the construction mode. Let us remember again that the X-definition serves as the description according to which the result is constructed. We will discuss each type of group separately.

5.1 xd:sequence

Let's first show the construction of the group "xd:sequence". If we write a "create" command to the Script, the result of this command is used to create the elements of this group similar to the creation of the child nodes of the element. Let's start with a simple example:

```
<xd:def xmlns:xd = "http://www.xdef.org/xdef/4.2">
<A>
  <xd:sequence xd:script= "occurs *; create 2">
     <B xd:script= "create true"/>
      required string; create "Text";
     <C xd:script= "create true"/>
  </xd:sequence>
</A>
</xd:def>
```

http://xdef.syntea.cz/tutorial/en/example/C311.html

The result:

```
<A>
  <B/>
  Text
  <C/>
  <B/>
  Text
  <C/>
</A>
```

As the reader is sure to expect, the group elements were created twice (in the "create" command for the group a number 2 was entered).

If the data source for group creation "xd:sequence" is a list of nodes resulting from the XPath expression the situation will be similar to the generation of the child nodes of an element. We'll show it in the example that we discussed in the previous chapter. Let us, therefore, take the data source to produce the result:

```
<X>
  <A/>
  <Y/>
  <Z/>
  <A/>
  <Z/>
</X>
```

Let the element "A" be the desired result. Its descendants are described by the sequence in which the "B" elements will be created from the "A" elements from the data source and the "C" element will be created from the second occurrence of the "Z" element. The desired result will therefore be:

```
<A>
  <B/>   (created from the first "A")
  <B/>   (created from the second "A")
  <C/>   (created from the second "Z")
```

As the data source for sequence construction, we pass the element "source" in the following X-definition:

```
<xd:def xmlns:xd = "http://www.xdef.org/xdef/4.2">
<xd:declaration>
  <![CDATA[
    /* Create the element as source data. */
      Element source = xparse("<X><A/><Y/><Z/><A/><Z/></X>");
  ]]>
</xd:declaration>

<A>
  <xd:sequence>
    <B xd:script="occurs 1..*; create xpath('/X/A', source)"/>
    <C xd:script="occurs 1; create xpath('/X/Z[2]', source)"/>
  </xd:sequence>
</A>
</xd:def>
```

http://xdef.syntea.cz/tutorial/en/example/C312.html

5.2 xd:mixed

Let's look at the group "xd:mixed" and take the following X-definition:

```
<xd:def xmlns:xd = "http://www.xdef.org/xdef/4.2">
<A>
  <xd:mixed>
    <B xd:script= "create true" />
    required string; create "Text";
    <C xd:script= "create true" />
  </xd:mixed>
</A>
</xd:def>
```

http://xdef.syntea.cz/tutorial/en/example/C321.html

The result will be similar to the group "xd:sequence" (models of child nodes of the group are executed in the order they are written to the X-definition):

```
<A>
  <B/>
  Text
  <C/>
</A>
```

Let's look at a more complicated case. Here, we try to create a result using the "xd:mixed" group of the data we pass through the element:

```
<xd:def xmlns:xd = "http://www.xdef.org/xdef/4.2">
<xd:declaration>
  <![CDATA[
  /* Create the element with source data. */
  Element source = xparse("<A><C/><B/></A>");
  ]]>
</xd:declaration>

<A>
  <xd:mixed>
    <B xd:script="create xpath('B', source)"/>
    <C xd:script="create xpath('C', source)"/>
  </xd:mixed>
</A>
</xd:def>
```

http://xdef.syntea.cz/tutorial/en/example/C322.html

Note that if we process the input data in the validation mode with the input with the element which is in the variable "source" the order of the descendants of element "A" would be the same as is in the input:

```
<A>
  <C/>
  <B/>
</A>
```

But the result will be:

```
<A>
```

```
  <B/>
  <C/>
</A>
```

This is because the "xd:mixed" elements are processed in the order in which they are written in X-definition (the process is controlled by X-definition).

5.3 xd:choice

Let's finally ask what happens when we use the "xd:choice" group:

```
<xd:def xmlns:xd = "http://www.xdef.org/xdef/4.2">
<A>
  <xd:choice xd:script= "*; create 2">
    <B xd:script= "create true" />
    required string; create "Text";
    <C xd:script= "create true" />
  </xd:choice>
</A>
</xd:def>
```

http://xdef.syntea.cz/tutorial/en/example/C331.html

The processor in this case, in each iteration (we requested two), chooses the first satisfactory variant (in our case, element "B") and the result will be:

```
<A>
  <B/>
  <B/>
</A>
```

If we wanted to influence which option the processor had to choose, we would have to write somehow in the Script which option should be chosen. E.g.:

```
<xd:def xmlns:xd = "http://www.xdef.org/xdef/4.2">
<xd:declaration> int i = 0; </xd:declaration>
<A>
  <xd:choice xd:script= "*; create 3">
    <B xd:script= "create i++ == 0" />
    string; create i++ == 2 ? "Text" : null;
    <C xd:script= "create i++ == 5" />
  </xd:choice>
</A>
</xd:def>
```

http://xdef.syntea.cz/tutorial/en/example/C332.html

The result will be:

```
<A>
  <B/>
  Text
  <C/>
</A>
```

Let the reader explain why (help: after the processed variant the other variants are not processed).

Let's try to create a result using the "xd:choice" group from the data we pass through the element:

```
<xd:def xmlns:xd = "http://www.xdef.org/xdef/4.2">
<xd:declaration>
  <![CDATA[
    /* Create the element as source data. */
    Element source = xparse("<A><C/><B/></A>");
  ]]>
</xd:declaration>
<A>
  <xd:choice>
    <B  xd:script="create xpath('B', source)"/>
    <C  xd:script="create xpath('C', source)"/>
  </xd:choice>
</A>
</xd:def>
```

http://xdef.syntea.cz/tutorial/en/example/C333.html

The result will be:

```
<A>
  <B/>
</A>
```

The reason why the "C" element which is the first in the descendant sequence is not created is the same as in the previous cases: the processor handles the X-definition sequentially and performs the first "B" model.

6 The context for XML construction

In the previous explanation, we consistently specified data for the creation of each resulting XML object. In the X-Script, we specified in the "create" section for each item to be created a command that returned the value by which the processor should produce the result. We did not suppose the X-definition processor "knows" what we did in the previous step. Such writing is often not necessary. Either the "create" command can be omitted altogether (then the default operation is performed), or we can refer to the data from the previous operation of the command "create" from which the entire parts of the resulting XML object can be created. The source data used in the construction mode in this case is called the "context" for creating XML objects.

Let's start with the situation we have already mentioned, namely the possibility of creating an element from a text value. We show a variant where we explicitly do not specify a value for the text element's data source: it is automatically taken from the source element data and becomes the context for creating additional items. You can write the "Salary" element without specifying the "create" section in the Script of the node. The element is created according to the rule explained in the previous chapter. We skipped the specification of the "create" section for the text value. Data is automatically taken from the text of the source element value, which will become the context for further processing:

```
<xd:def xmlns:xd = "http://www.xdef.org/xdef/4.2">
<xd:declaration>
  <![CDATA[
    /* Create the element with source data. */
    Element source = xparse("<A>1234</A>");
  ]]>
</xd:declaration>
<Person>
  <Salary xd:script= "create source">
    required int; /* value is taken from the context */
  </Salary>
</Person>
</xd:def>
```

http://xdef.syntea.cz/tutorial/en/example/C341.html

Since we already know that an element with a text value is automatically created from a string (and this becomes a context), we can also write the X definition:

```
<xd:def xmlns:xd = "http://www.xdef.org/xdef/4.2">
<Person>
  <Salary xd:script = "create '1234'">
    required int;
  </Salary>
</Person>
</xd:def>
```

http://xdef.syntea.cz/tutorial/en/example/C342.html

We see that for the construction of the element "Salary", a string value that has become the context for creating the descendants of this element (in our case the text value of the element) was used.

Or, if we use a database (the variable "source" is ResultSet and the "getItem" method returns a string with the corresponding column):

```
<Salary xd:script= "create source.getItem('SALARY')">
   required int;
</Salary>
```

How will it be with element attributes? If you omit the "create" section in the attribute description, the appropriately named values from the context will be used for the construction of attributes in the model (e.g. element attributes, column entries in the database table rows, "NamedValue" values in the "Container" object,

etc.). Thus, if we specify an element as the data source with the "name" and "receive" attributes, the attribute values in the generated result will automatically be taken from the context. In the following example, the "Plat" element is also generated from context. The X-definition will take the form:

```
<xd:def xmlns:xd = "http://www.xdef.org/xdef/4.2">
<xd:declaration>
  <![CDATA[
  Element source = xparse("<X Name='John' Surname='Smith'><Salary>1234</Salary></X>");
  ]]>
</xd:declaration>

<Person xd:script= "create source"
       Name    = "string;"
       Surname = "string;">
  <Salary>
    required int;
  </Salary>
</Person>
</xd:def>
```

http://xdef.syntea.cz/tutorial/en/example/C343.html

The result will be:

```
<Person Name='John' Surname='Smith'><Salary>1234</Salary></Person>
```

If the child items are produced from the data source that we passed to the element, the context will automatically be used for each child descendant. Context then becomes the context for inner items. It should be remembered that as a data source for the creation of internal data elements, the elements with the same name as the context will automatically be used. When creating a text value, the corresponding text value from the context will be used. Let us show how the element "A" will be created from the element "X". The model of element A is described by the following X-definition:

```
<xd:def xmlns:xd = "http://www.xdef.org/xdef/4.2">
<xd:declaration>
  <![CDATA[
  Element source =
    xparse("<X a1='a1' a2='a2'>XXX<R a='1'><S>SSS</S></R><Q>QQQ</Q><P>PPP</P></X>");
  ]]>
</xd:declaration>
<A xd:script= "create source"
   a1 = "string;"
   a2 = "string;">
  <P>
    string;
  </P>
  <Q>
    string;
  </Q>
  <R a= "int">
    <S>
      string;
    </S>
  </R>
  string;
</A>
</xd:def>
```

http://xdef.syntea.cz/tutorial/en/example/C344.html

Note that elements were selected from the context in the order that corresponds to the model rather than to the data source. The result will be (indentation of the result is used for better readability):

```
<A a1="a1" a2="a2">
    <P>
      PPP
    </P>
    <Q>
      QQQ
    </Q>
    <R a="1">
       <S>
         SSS
       </S>
    </R>
    XXX
</A>
```

Let's look at the behavior of groups with context.

You can also use the "Container" value as a context. Named values are assigned to attributes with corresponding names. The value of "1234" from the sequence section is used to create the element "Salary" and its text value:

```
<xd:def xmlns:xd = "http://www.xdef.org/xdef/4.2">
<xd:declaration>
   /* Create a Container used as context for the creation of the model "Person". */
   Container source = [%Name='John', %Surname='Smith', '1234'];
</xd:declaration>

<Person xd:script= "create source"
       Name    = "string;"
       Surname = "string;">
  <Salary>
     string;
  </Salary>
</Person>
</xd:def>
```

http://xdef.syntea.cz/tutorial/en/example/C345.html

The result:

```
<Person Name="John" Surname="Smith">
  <Salary>
    1234
  </Salary>
</Person>
```

If the items in the sequence part of Container are again Container values, they will be used to create the descendants of the model:

```
<xd:def xmlns:xd = "http://www.xdef.org/xdef/4.2">
<xd:declaration>
   /* Create a Container used as context for the creation of the model "Person". */
   Container source = [%A=[%a='A', [%b='B', 'C'], 'D']];
</xd:declaration>

<A xd:script= "create source"
   a = "string">
  <B b = "string">
     string
  </B>
  string
</A>
</xd:def>
```

http://xdef.syntea.cz/tutorial/en/example/C346.html

The result:

```
<A a="A">
  <B b="B">
   C
  </B>
  D
</A>
```

6.1 Context used for the creation of the group xd:sequence

Let's try the following example of the X-definition:

```
<xd:def xmlns:xd = "http://www.xdef.org/xdef/4.2">
<xd:declaration>
  <![CDATA[ Element source = xparse("<X><B a='b'/><C a='c'/></X>"); ]]>
</xd:declaration>
<A>
  <xd:sequence xd:script="create source">
    <B a="string" />
    <C a="string" />
  </xd:sequence>
</A>
</xd:def>
```

http://xdef.syntea.cz/tutorial/en/example/C411.html

The result will be as we expected:

```
<A>
  <B a="b"/>
  <C a="c"/>
</A>
```

Now put the sequence of internal elements with the quantifier to allow multiple iterations of the internal elements and add the command create so that it generates several sequences:

```
<xd:def xmlns:xd = "http://www.xdef.org/xdef/4.2">
<xd:declaration>
  <![CDATA[
  Element source =
    xparse("<X><Y><B a='b'/><C a='c'/></Y><Y><C a='x'/></Y><Y><B a='y'/></Y></X>");
  ]]>
</xd:declaration>
<A>
  <xd:sequence xd:script="occurs *; create xpath('//Y', source)">
    <B xd:script="occurs ?" a="string" />
    <C xd:script="occurs ?" a="string" />
  </xd:sequence>
</A>
</xd:def>
```

http://xdef.syntea.cz/tutorial/en/example/C412.html

The reader will easily explain why the result is the following (help: the result of XPath expression is all the "Y" elements that are passed as a group creation context):

```
<A>
  <B a="b"/>
  <C a="c"/>
  <C a="x"/>
  <B a="y"/>
</A>
```

6.2 Context used for the construction of the group xd:choice

Let's again start with a simple example:

```
<xd:def xmlns:xd = "http://www.xdef.org/xdef/4.2">
<xd:declaration>
<![CDATA[
 Element source = xparse("<X><C a='c'/><B a='b'/></X>");
]]>
</xd:declaration>
<A>
  <xd:choice xd:script="create source">
    <B a="string" />
    <C a="string" />
  </xd:choice>
</A>
</xd:def>
```

http://xdef.syntea.cz/tutorial/en/example/C421.html

The result will be:

```
<A>
  <B a="b"/>
</A>
```

The reason why element "B" is created and not "C" is that the process is controlled by X-definition. Here element "B" is described earlier than element "C" which is first in the source data.

Now, modify our example for a group with a quantifier for more occurrences of the group content:

```
<xd:def xmlns:xd = "http://www.xdef.org/xdef/4.2">
<xd:declaration>
  <![CDATA[
    Element source = xparse("<X><Y><B a='b'/></Y><Y><C a='x'/></Y><Y><B a='y'/></Y></X>");
  ]]>
</xd:declaration>
<A>
   <xd:choice xd:script="occurs *; create xpath('//Y',source)">
      <B a="string" />
```

```
      <C a="string" />
   </xd:choice>
</A>
</xd:def>
```

http://xdef.syntea.cz/tutorial/en/example/C422.html

The result:

```
<A>
  <B a="b"/>
  <C a="x"/>
  <B a="y"/>
</A>
```

The reader will easily explain why.

6.3 Context used for the construction of the group xd:mixed

Let's have a simple example:

```
<xd:def xmlns:xd = "http://www.xdef.org/xdef/4.2">
<xd:declaration>
  <![CDATA[
  Element source = xparse("<X><C a='c'/><B a='b'/></X>");
  ]]>
</xd:declaration>
<A>
  <xd:mixed xd:script="create source">
    <B a="string" />
    <C a="string" />
  </xd:mixed>
</A>
</xd:def>
```

http://xdef.syntea.cz/tutorial/en/example/C431.html

Surely you know why the resulting order of the elements in the mixed group will match the X-definition entry rather than the order in the context:

```
<A>
  <B a="b"/>
  <C a="c"/>
</A>
```

Let's modify our example, as in the previous paragraph, for a mixed group:

```
<xd:def xmlns:xd = "http://www.xdef.org/xdef/4.2">
<xd:declaration>
  <![CDATA[
  Element source =
    xparse("<X><Y><B a='b'/><C a='c'/></Y><Y><C a='x'/></Y><Y><B a='y'/></Y></X>");
  ]]>
</xd:declaration>
<A>
  <xd:mixed xd:script="create xpath('//Y',source)">
    <B a="string" />
    <C a="string" />
  </xd:mixed>
</A>
</xd:def>
```

http://xdef.syntea.cz/tutorial/en/example/C432.html

The result will be:

```
<A>
  <B a="b"/>
  <C a="c"/>
  <C a="x"/>
  <B a="y"/>
</A>
```

In contrast to the choice group, the items of the result are in the same order as the input. The mixed group passes all the variants sequentially until it finds an appropriate element and, until exhausted, all the variants begin again from the first element in the model.

7 The use of the result of XPath expression (method "from")

When creating the resulting element, we often need to select a particular object from the context and possibly set a new context for the descendants of the object being created.

In the examples in the previous text, we have shown the use of XPath expression on XML data. If the XML element is a context for the element model, we can use the "from" method that executes the XPath expression above the current context. Setting a new context by using the "from" method is also valid for all descendants created by this command. Let's take a look at our example. Let's show an X-definition describing the element that we create from different parts of the context:

```
<xd:def xmlns:xd = "http://www.xdef.org/xdef/4.2">
<xd:declaration>
  <![CDATA[
    Element source = xparse("<X a='a'><Y a='x'><Z a='b'/></Y><Y a='y'><Z a='c'/></Y></X>");
  ]]>
</xd:declaration>
<A xd:script="create source"> /* context with root element X */
  <B xd:script="create from('@a')"> /* context is attribute "a" of element X */
    string; create from('text()'); /* value taken from the attribute above */
  </B>
  <C xd:script="occurs *; create from('Y/Z')"> /* sequence of elements Z */
    string; create from('@a'); /* the attribute "a" of "Z" is used to create the text node */
  </C>
</A>
</xd:def>
```

http://xdef.syntea.cz/tutorial/en/example/C501.html

In the beginning, the context for creating the "A" element at the root of the source element is set. To create element "B", the attribute "a" is used from the context set, and its value becomes the context for creating element "B". As discussed in the previous text, a textual value of an element is created from the text value passed by the "from" method (i.e., the value of attribute "a" of the current context). We also know that we could omit the "create" section here. The "from" method used for element "C" returns all nodes that correspond to the "X / X" path from the current context. The text value of these nodes is taken from the values of the attribute "a" of these nodes. The result will be:

```
<A>
  <B>a</B>
  <C>b</C>
  <C>c</C>
</A>
```

Let's show an X-definition in which values are written in the source data into the element's text instead of the attributes. The result will be the same as in the previous variant. Note that we can omit the "create" section for creating the text of elements:

```
<xd:def xmlns:xd = "http://www.xdef.org/xdef/4.2">
<xd:declaration>
  <![CDATA[
    Element source = xparse("<X>a<X><X>b</X></X><X><X>c</X></X></X>");
  ]]>
</xd:declaration>
<A xd:script="create source">
  <B xd:script="create from('text()')">
     string;
  </B>
  <C xd:script="occurs *; create from('X/X')">
     string;
  </C>
</A>
</xd:def>
```

http://xdef.syntea.cz/tutorial/en/example/C502.html

Vice versa, from the text nodes at the input we can create attributes:

```
<xd:def xmlns:xd = "http://www.xdef.org/xdef/4.2">
<xd:declaration>
  <![CDATA[
    Element source = xparse("<X>a<X><X>b</X></X><X><X>c</X></X></X>");
  ]]>
</xd:declaration>
<A xd:script="create source">
```

```
  <B xd:script="create from('text()')" a="string; create from('text()')"/>
  <C xd:script="occurs *; create from('X/X')" a="string; create from('text()')"/>
</A>
</xd:def>
```

http://xdef.syntea.cz/tutorial/en/example/C503.html

The result:

```
<A>
  <B a="a"/>
  <C a="b"/>
  <C a="c"/>
</A>
```

Let's look at a slightly more complex example from the real world. Let the input data be in the following element:

```
<Item id="0123456789">
  <Client role="1" company="SomeCompany Ltd" cId="12345678" />
  <Client role="2" name="Peter" surname="Brown" pId="311270/1234" />
  <Client role="3" name="John" surname="Smith" pId="120455/2345" cId="87654321" />
</Item>
```

From this data, we create the element "Contract", in which the attribute "number" is taken from the attribute "id" of the element "Item". Additionally, we also create the attribute "date" in which we store the current date and time taken from the operating system. The internal elements "Owner", "Client" and "Holder" are created from the "Client" elements depending on the value of the attribute "role". We create the element "Client" if the attribute "role" has the value "1" (note that we used a backslash for the apostrophe inside the string). For attributes with the same name as in the context, we can omit the "create" section. The element "Holder" is created from the element "Client" if the value of the attribute "role" is "2". The other attributes are taken automatically from the context. Finally, we create the element "Policyholder" from the element "Client" if the attribute "role" has the value "3". The attribute "name" is composed of the values of the attributes "name" and "surname" from the context. The "cId" attribute is again accepted automatically. The attribute "pId" here is not used). The X definition that performs this transformation will take this form (note that the string can be declared in multiple rows in the Script):

```
<xd:def xmlns:xd = "http://www.xdef.org/xdef/4.2">
<xd:declaration>
  <![CDATA[
    Element source = xparse("<Item id='0123456789'>
      <Client role='1' company='SomeCompany Ltd' cId='12345678' />
      <Client role='2' name='Peter' surname="Brown" pId='311270/1234' />
      <Client role='3' name='John' surname='Smith' pId='120455/2345' cId='87654321' />
    </Item>" );
  ]]>
</xd:declaration>

<Contract  xd:script="create source"
    date          ="required dateTime(); create now()"
    number        ="required num(10); create from('@id')" >
    <Ownew xd:script="create from('Client[@role=\'1\']')"
       cId        ="required num(8)"
       company    ="required string(1,30)" />
    <Holder xd:script="create from('Client[@role=\'2\']')"
       pId        ="required string(10,11) "
       name       ="required string(1,30)"
       surname    ="required string(1,30)" />
    <Policyholder xd:script="create from('Client[@role=\'3\']')"
       name       ="required string(1,30); create from('@name') + ' ' + from('@surname')"
       cId        ="required num(8)" />
</Contract>
</xd:def>
```

http://xdef.syntea.cz/tutorial/en/example/C504.html

Result is:

```
<Contract date="2012-10-7T17:05" number="0123456789">
  <Owner company="SomeCompany Ltd" cId="12345678"/>
  <Holder pId="311270/1234" name="Peter" surname="Brown"/>
  <Policholder cId="87654321" namev="John Smith"/>
</Contract>
```

Let's show an example of how to generate only elements that have some descendants:

Input XML:

```
<a>
  <b x=”1”><c y=”11”></b>
  <b x=”2” />
  <b x=”3”><c y=”33”></b >
</a
```

The XPath expression in the create section of model nodes will be "b/*/..". So the X-definition:

```
<xd:def xmlns:xd=”http://www.xdef.org/xdef/4.2' root=”a”>
  <a>
    <b xd:script="*; create from('b/*/..');" x="string()">
      <c xd:script="*” y=”string()” />
    </b>
  </a>
</xd:def>
```

Note you can also write the XPath expression as e.g: b//parent::*". You can try it on:*

http://xdef.syntea.cz/tutorial/en/example/C505.html

Result XML:

```
<a>
  <b x=”1”><c y=”11”></b>
  <b x=”3”><c y=”33”></b >
</a>
```

8 Combination of the validation mode and construction mode

Both X-definition modes, i.e. the validation mode and construction mode, can be combined. Let's use the previous example, but first, validate the data using the model "Item" (note that the "xparse" method has a second parameter with either the X definition name or "*" if the X definition has no name). When executing commands in the section xd:declaration, the input data from the variable "data" is verified according to the model "Item" (which must of course be declared as "root" in the X-definition). In the construction mode the validated element is used as a context for the model "Contract" as in the previous example:

```
<xd:def xmlns:xd = "http://www.xdef.org/xdef/4.2" root="Item">
<xd:declaration>
<![CDATA[
  String data =
   "<Item id='0123456789'>
      <Client role='1' company='SomeCompany Ltd' cId='12345678' />
      <Client role='2' name='Peter' surname='Brown' pId='311270/1234' />
      <Client role='3' name='John' surname='Smith' pId='120455/2345' cId='87654321' />
   </Item>";
  Element source = xparse(data, "*");
]]>
</xd:declaration>

<Contract xd:script="create source"
      date      ="required dateTime(); create now()"
      number    ="required num(10); create from('@id')" >
  <Owner xd:script="create from('Client[@role=\'1\']')"
      cId       ="required num(8)"
      company   ="required string(1,30)" />
  <Holder xd:script="create from('Client[@role=\'2\']')"
      pId       ="required string(10,11) "
      name      ="required string(1,30)"
      surname   ="required string(1,30)" />
  <Policyholder xd:script="create from('Client[@role=\'3\']')"
      name      ="required string(1,30); create from('@name') + ' ' + from('@surname')"
      cId       ="required num(8)" />
</Contract>

<Item id='num(10)'>
  <Client role = 'int()' company = 'string()' cId = 'num(8)' />
  <Client role = 'int()' name = 'string()' surname = 'string()' pId = 'string()' />
  <Client role = 'int()' name = 'string()' surname ='string()' pId = 'string()' cId = 'string()' />
</Item>

</xd:def>
```

http://xdef.syntea.cz/tutorial/en/example/C601.html

We can also specify the processing mode and construction mode in reverse order. First, we will process the source XML document in the validation mode, and finally (in the section "finally" of the model "Item" which describes the validated input data) we will create the result using the "xcreate" method, which will use the validated element as a context. Note that the "create" section of the model "Contract" has been omitted because the document has automatically set the validated data as a context. The result is set using the "setResultElement" method in the section "finally" of the model "Item". The example follows:

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

<Contract
    date        ="required dateTime(); create now()"
    number      ="required num(10); create from('@id')" >
  <Owner xd:script="create from('Client[@role=\'1\']')"
      cId       ="required num(8)"
      company   ="required string(1,30)" />
  <Holder xd:script="create from('Client[@role=\'2\']')"
      pId       ="required string(10,11) "
      name      ="required string(1,30)"
      surname   ="required string(1,30)" />
  <Policyholder xd:script="create from('Client[@role=\'3\']')"
      name      ="required string(1,30); create from('@name') + ' ' + from('@surname')"
      cId       ="required num(8)" />
</Contract>

<Item xd:script='finally returnElement(xcreate("Contract"))'
    id='num(10)' >
  <Client role = 'int()' company = 'string()' cId = 'num(8)' />
  <Client role = 'int()' name = 'string()' surname = 'string()' pId = 'string()' />
  <Client role = 'int()' name = 'string()' surname ='string()' pId = 'string()' cId = 'string()' />
</Item>

</xd:def>
```

http://xdef.syntea.cz/tutorial/en/example/C602.html

Finally, here is an example that creates an HTML result from the wind and temperature measurements. As a result, based on input data processing, measured daily data and the average temperature are displayed. We show that during the processing you can prepare values for the result (see the variables "sum" and "num" which are used to calculate the average temperature).

The input data:

```
<Weather date = "2005-05-11" >
   <Measurement wind = "5.3" temperature = "13.0" time = "05:00" />
   <Measurement wind = "7.2" temperature = "15.2" time = "11:00" />
   <Measurement wind = "8.7" temperature = "18.1" time = "15:00" />
   <Measurement wind = "3.9" temperature = "16.5" time = "20:00" />
</Weather>
```

The X-definition:

```
<xd:def xmlns:xd = "http://www.xdef.org/xdef/4.2" root = "Weather">
<xd:declaration>
  float sum = 0; int num = 0;
</xd:declaration>

<Weather xd:script="finally returnElement(xcreate('html'))"
   date = "optional date()">
   <Measurement xd:script = "occurs +"
       wind = "required float(0, 150)"
       temperature = "required float(-40,60) /*limit sof temperature*/
                     onTrue {num++; sum += getParsedFloat();}"
       time = "required time()" />
</Weather>

<html>
  <h3>string; create "Weather on " + from("@date")</h3>
  <li xd:script = "occurs +; create from('Measurement')">
    string; create "Time: " + from("@time") + ", wind: " + from("@wind") +
                   ", temperature: " + from("@temperature")
  </li>
  <h3 style="fixed 'background: yellow'">
    string; create num==0 ? "No data" : "Average temperature: " + sum/num
  </h3>
</html>
</xd:def>
```

http://xdef.syntea.cz/tutorial/en/example/C603.html

The result:

```
<html>
  <h3>Weather on 2005-05-11</h3>
  <li>Time: 05:00:00, wind: 5.3, temperature: 13.0</li>
  <li>Time: 11:00:00, wind: 7.2, temperature: 15.2</li>
  <li>Time: 15:00:00, wind: 8.7, temperature: 18.1</li>
  <li>Time: 20:00:00, wind: 3.9, temperature: 16.5</li>
  <h3 style="background: yellow">Average temperature: 15.7</h2>
</html>
```

9 Template model

To create XML data in which the values are constant we can use the "template" model. It will be created by entering the keyword "template" into the model script. The "template" command causes the text nodes and attribute values of the model to be converted to a script in which these values are declared as constants. If we need to use a variable in the model at some point, we have to use the Script. We'll do this by writing down the keyword "**$$$script:**" in an attribute or a text node. The text behind this entry is then processed in the usual way as a Script: For example:

```
<xd:def xmlns:xd='http://www.xdef.org/xdef/4.2' root='html'
        xmlns='http://www.w3.org/1999/xhtml'>
<html xd:script="template">
  <head>
    <title>Today</title>
      <meta http-equiv="Content-Type" content="text/html; charset=windows-1250"/>
  </head>
  <body bgcolor="yellow">
    <h1>Today is:</h1>
    <h2>$$$script: string(); create now().toString('{L(en,US)}EEEE, d. MMMM yyyy GG, hh:mm a')</h2>
    <h3>This is an example of a template model in construction mode.</h3>
  </body>
</html>
</xd:def>
```

http://xdef.syntea.cz/tutorial/en/example/C701.html

The content of the text values in the model is automatically converted to a script where the text values are converted as fixed values. The internally generated form of the model would look like this:

```
<html>
  <head>
    <title>
      fixed "Actual date";
    </title>
    <meta http-equiv="fixed 'Content-Type'"
          content="fixed 'text/html;'"
          charset="fixed 'windows-1250'"/>
  </head>
  <body bgcolor="fixed 'yellow'">
    <h1>
      fixed "Actual date:";
    </h1>
    <h2>
      string(); create now().toString('{L(en,US)}EEEE, d. MMMM yyyy GG, hh:mm a')
    </h2>
    <h3>
      fixed "This is an example of a template model in construction mode.";
    </h3>
  </body>
</html>
```

When executing the construction mode after writing "$$$ script:", the value is created by the "create" section as already explained. The result will be:

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Actual date</title>
  </head>
  <body bgcolor="yellow">
    <h1>Actual date:</h1>
    <h2> Friday, 5. August 2017 AD, 11:31 AM</h2>
```

```
    <h3> This is an example of a template model in construction mode.</h3>
  </body>
</html>
```

9.1 Example of HTML creation using the template model

Let's finally give an example where we enter input data as a context and generate HTML data from them. We have input data with the unsorted list of historical regents in the Czech lands:

```
<regents>
  <regent>
    <name>Tomáš Garrigue Masaryk</name>
   <reigned title="prezident">
      <from>1918</from>
      <to>1935</to>
    </reigned>
  </regent>
  <regent>
    <name>Václav Havel</name>
   <reigned title="prezident">
      <from>1989</from>
      <to>2003</to>
    </reigned>
  </regent>
  <regent>
    <name>Václav I.</name>
    <reigned titul="král český">
      <from>1230</from>
      <to>1253</to>
    </reigned>
  </regent>
  ...
</regents>
```

Some parts of the X-definition are described as a template model - see the "head" element and the first occurrence of the "tr" element (table heading). Other data is generated from input data. Note that the data about kings from the context - i.e. the result of the method from ("//regent") - is sorted by the beginning of the government (if the information at the end of the government is not filled, then the entry is empty):

```
<xd:def xmlns:xd='http://www.xdef.org/xdef/4.2' xmlns='http://www.w3.org/1999/xhtml'>
<html xd:script='template'>
   <head>
      <title>Regents in the Czech lands</title>
   </head>
   <body>
     <table border="border">
       <tr>
         <td>Regent</td>
         <td>Ruled from</td>
         <td>Ruled to</td>
         <td>Years</td>
       </tr>
       <tr xd:script='$$$script: string(); occurs *; create from("//regent").sort("reigned/from/text()")'>
         <td>create from('name/text()')</td>
         <td>create from('reigned/from/text()')</td>
         <td>optional string(); create from('reigned/to/text()')</td>
         <td>optional string(); create from('reigned/to/text()').isEmpty() ? null /*blank*/
           : from('number(reigned/to/text()) - number(reigned/from/text())');
         </td>
       </tr>
     </table>
   </body>
</html>
</xd:def>
```

http://xdef.syntea.cz/tutorial/en/example/C711.html

The result:

```
<html xmlns='http://www.w3.org/1999/xhtml'>
  <head><title>Regents in the Czech lands</title></head>
  <body>
    <table border="border">
      <tr>
        <td>Regent</td>
        <td>Ruled from</td>
        <td>Ruled to</td>
```

```
        <td>Years</td>
      </tr>
      <tr>
        <td>Václav I.</td>
        <td>1230</td>
        <td>1253</td>
        <td>23</td>
      </tr>
         ...
      <tr>
        <td>Miloš Zeman</td>
        <td>2013</td>
        <td/>
        <td/>
      </tr>
    </table>
  </body>
</html>
```