User Manual of BiXJ

Dongxi Liu
PSD Research Meeting
May 04, 2007

Outline

- Commands of using BiXJ
- BiXJ Constructs
- Library Definition
- Type Definition

Commands

- The command with the source and view specified in the *code*.bix
  - java -jar *BiXJ*.jar *code*.bix [0|1]
- The command with the source and view specified in the command
  - java -jar *BiXJ*.jar *code*.bix [0|1] *src*.xml *view*.xml
- The options:
  - 0 for forward transformation
  - 1 for backward transformation

Notations

- An XML value can be an element, string (or texts), attribute or a namespace.
  - *s* denote one value
  - *S* or *V* denote a list of values

BiXJ Constructs

- Generic
  - xid, xconst, xseq, xpar, xmap, xif, xlet, xvar, xfunapp
- XML
  - Element or Text
    - xchild, xgettag, xchcont, xrename, xfndata
  - Attribute
    - xattribute, xattname, xattval,xmakeatt
  - Name Space
    - xnamespace, xnsprefix, xnsuri, xsetns, xnsdec, xnsadddec

BiXJ Constrcuts - *cont'*

- List
  - xorder, xsub, xdistinct
- Arithmetic
  - xsum, xcount
- Predicate
  - xeq, xgt, xlt, xiselement, xistext, xwithtag, xwithname, xexist
- I/O
  - input, output

Generic Constructs

- xid
- xconst
- xseq
- xpar
- xmap
- xif
- xlet
- xvar
- xfunapp

xid

<xid />

Source:

A list of values, say *S*.

View:

The list *S*.

xconst

<xconst>*S*</xconst>

Argument:

A list of values *S*.

Source:

A list of values.

View:

The constant *S* for any source data

xseq

<xseq>$X_1\ X_2 \ldots X_n$</xseq> :

Arguments:

A list of transformations: $X_1,\ldots, X_n$

Source:

A list of values, say *S*.

View:

The result of applying X_i in sequence.

X_1 is applied to the source *S*, and X_i $(2 \le i \le n)$ is applied to the result of applying X_{i-1}.

xpar

<xpar>$X_1 \ldots X_n$</xpar>:

Arguments:

A list of transformations: $X_1,\ldots, X_n$

Source:

A list of values.

View:

The list obtained by concatenating $V_1,\ldots, V_{n-1}$ and V_n, where V_i is the result of applying X_i to the empty value ().

xmap

<xmap>*X*</xmap> :

Argument:

A transformation *X*.

Source:

A list consisting of values, for instance, $s_1, s_2, \ldots, s_n$.

View:

The concatenation of $V_1,..,V_n$, where V_i is the result of applying *X* to s_i.

xif

<xif>P X_1 X_2</xif>:
- Arguments:
  - Three transformations: P, X_1 and X_2.
- Source:
  - A list of values, say S.
- View:
  - The result of applying X_1 to S, if the result of applying P to S is not the empty list.
  - The result of applying X_2 to S, otherwise.

xlet

<xlet><var>*Var*</var> X</xlet>
- Arguments:
  - A variable *Var*, wrapped in the var element, for the variable to be bound
  - A transformation X for the valid scope of *Var*.
- Source:
  - A list of values, say S.
- View:
  - The result of applying X to the empty value (), with the variable *Var* bound to S.

xvar

<xvar>*Var*</xvar>:
- Argument:
  - A variable *Var*.
- Source:
  - A list of values.
- View:
  - The most recent bound value of variable *Var*.

Function declaration

<function name="*fn*"
arg_1="Var_1" … arg_n="Var_n">
X
</function>

fn is the function name
$Var_1 \ldots Var_n$ are the formal parameters of this function
X is the function body

Functions are defined in a library, and used by xfunapp in the next page. The library definition is introduced in page 45.

xfunapp

<xfunapp>
<name>*fn*</name> <args>$X_1 \ldots X_n$</*args*>
</xfunapp>:
- Arguments:
  - The function name *fn*, which can be a string or a transformation.
  - A list of transformations for generating the concrete parameters.

xfunapp - *cont'*

- Source:
  - A list of values.
- View:
  - The result of applying the function body X to the empty value (), under the context with the formal parameter Var_i bound to the result of applying X_i to ().

Constructs for Element or Text

- xchild, xchcont, xrename, xfndata

xchild

<xchild /> :
Source:
A singleton list containing an element, say *<tag>S</tag>*.
View:
The list *S*.

xgettag

<xgettag /> :
Source:
A singleton list containing an element, say *<tag>S</tag>*.
View:
A singleton list containing string *tag* .

xchcont

<xchcont>X_1 X_2 … X_n</xchcont> :
Arguments:
A list of transformations: X_1,…, X_n
Source:
A singleton list containing an element, say *e*.
View:
A singleton list containing the element *e* with its contents and attributes replaced by:
Removing the existing contents and attributes of *e*;
Applying each X_i, from i=1 *to n*, to the empty value (). The result is added into the attributes of *e* if it is an attribute, and added into *e*'s content list otherwise.

xrename

<xrename>*X*</xrename>:
Argument:
A transformation *X*.
Source:
A singleton list containing an element, say *e*.
View:
A singleton list containing the element *e* with its tag changed into the string computed by *X* on the source data.

xfndata

<xfndata />:
Source:
A list of elements or strings, say s_1,…,s_n.
View:
A list of strings. The *i*-th string is the result of concatenating all text contents in s_i if it is an element, or just s_i *itself* if it is a string.

Constructs for Attribute

- xattribute
- xattname
- xattval
- xmakeatt

xattribute

<xattribute /> :
Source:
A singleton list containing an element, say <*tag* an_1= "av_1"… an_n="av_n" >*S*</*tag*>.
View:
The list, containing an_1="av_1",…, an_n="av_n" .

xattname

<xattname />:
Source:
A singleton list containing an attribute, say *an* = "*av*".
View:
A singleton list containing the attribute name "*an*".

xattval

<xattval />:
Source:
A singleton list containing an attribute, say *an* = "*av*".
View:
A singleton list containing the attribute value "*av*".

xmakeatt

<xmakeatt>X_1 X_2</xmakeatt>:
Arguments:
Two transformations X_1 and X_2 for computing the attribute name and value, respectively.
Source:
A list of values, say *S*.
View:
An attribute with the result of applying X_1 to *S* as its name, and the result of applying X_2 to S as its value.

Constructs for Name Space

- xnamespace
- xnsprefix
- xnsuri
- xsetns
- xnsdec
- xnsadddec

xnamespace

<xnamespace />:

Source:

A singleton list containing an attribute or element, say *s*.

View:

A singleton list containing the namespace of *s*.

xnsprefix

<xnsprefix />:

Source:

A singleton list containing a namespace, say *s*.

View:

A singleton list containing the prefix of *s*.

xnsuri

<xnsuri />:

Source:

A singleton list containing a namespace, say *n*.

View:

A singleton list containing the URI of namespace *n*.

xsetns

<xsetns>X_1 X_2</xsetns>:

Arguments:

Two transformations X_1 and X_2

Source:

A singleton list containing an element, say *e*.

View:

A singleton list containing *e* with a new namespace, whose prefix and URI are obtained by applying X_1 and X_2 to the empty value (), respectively.

xnsdec

<xnsdec />:

Source:

A singleton list containing an element, say *e*.

View:

A list of namespaces declared in the element *e*.

xnsadddec

<xnsadddec>X_1 X_2</xnsadddec>:

Arguments:

Two transformations X_1 and X_2

Source:

A singleton list containing an element, say *e*.

View:

A singleton list containing *e* with its namespace declaration extended by a new namespace. Its prefix and URI are obtained by applying X_1 and X_2 to the empty value (), respectively.

Constructs for List

- xorder
- xsub
- xdistinct

xorder

<xorder>*X* <dir>*Dir*</dir></xorder>

Arguments:
Dir can be ascending or descending, and the default is ascending when <dir> is omitted.
X is a transformation

Source:
A list of values, say $s_1,\ldots,s_n$.

View:
The list of sorting $s_1,\ldots, s_n$ in the *Dir* (ascending or descending) order based on the corresponding result of applying *X* to s_i.

xsub

<xsub>X_1 X_2</xsub>:

Arguments:
Two transformations X_1 and X_2

Source:
A list of values, say *S*.

View:
A sub list of *S*, with the starting position and length computed by applying X_1 and X_2 to *S*, respectively.
Note: the starting position begins with 1.

xdistinct

<xdistinct />:

Source:
A list of values, say *S*.

View:
A sub list of *S*, containing all distinct values from *S*.

Arithmetic Constructs

- xsum
- xcount

xsum

<xsum>X_1 X_2</xsum>:

Arguments:
Two transformations X_1 and X_2.

Source:
A list of values, say *S*.

View:
A singleton list containing a digit string, obtained by summing the integer results of X_1 and X_2 on *S*.

xcount

<xcount />:
- Source:
  - a list of values, say *S*.
- View:
  - A singleton list containing the string "*n*", where *n* is the number of values in *S*.

Predicates

- xeq
- xgt
- xlt
- xiselement
- xistext
- xwithtag
- xwithname
- xexist

xeq

<xeq>X_1 X_2</xeq>:
- Arguments:
  - Two transformations X_1 and X_2.
- Source:
  - A list of values, say *S*.
- View:
  - A singleton list containing the string "true" if the results of X_1 and X_2 on *S* are equal, or an empty list otherwise.

xgt

<xgt>X_1 X_2</xgt>:
- Arguments:
  - Two transformations X_1 and X_2.
- Source:
  - A list of values, say *S*.
- View:
  - A singleton list containing the string "true" if the result of X_1 on *S*, after converted as an integer, is greater than the integer result of X_2 on *S*, or an empty list otherwise.

xlt

<xlt>X_1 X_2</xlt>:
- Arguments:
  - Two transformations X_1 and X_2.
- Source:
  - A list of values, say *S*.
- View:
  - A singleton list containing the string "true" if the result of X_1 on *S*, after being converted as an integer, is less than the integer result of X_2 on *S*, or an empty list otherwise.

xiselement

<xiselement />:
- Source:
  - A list of values, say *S*.
- View:
  - A singleton list containing the string "true" if *S* contains only an element, or an empty list otherwise.

xistext

<xistext>:
- Source:
  - A list of values, say *S*.
- View:
  - A singleton list containing the string "true" if *S* contains only a string (or a text), or an empty list otherwise.

xwithtag

<xwithtag>*str*</xwithtag>:
- Argument:
  - The string *str*
- Source:
  - A list of values, say *S*.
- View:
  - A singleton list containing the string "true" if *S* contains an element with the tag *str*, or an empty list otherwise.

xwithname

<xwithname>*str*</xwithname>:
- Argument:
  - The string *str*
- Source:
  - A list of values, say *S*.
- View:
  - A singleton list containing the string "true" if *S* contains an attribute with the name *str*, or an empty list otherwise.

xexist

<xexist>*X*</xexist>:
- Argument:
  - One transformation *X*.
- Source:
  - A list of values, say $s_1,\ldots,s_n$.
- View:
  - A singleton list containing the string "true" if there exists s_i, such that applying *X* to s_i returns a nonempty list, or an empty list otherwise.

I/O Constructs

- input
- output

input

<input>*str*</input>:
- Argument:
  - The file name *str*.
- Source:
  - A list of values, say *S*.
- View:
  - The root element in the file *str*.

output

<output>*str*</output>:

Argument:

The file name *str*.

Source:

A list of values, say *S*.

View:

The list *S*. In addition, *S* will be used to replace the content of file *str*. If *S* contains only one element, it will be written into *str* directly, otherwise the values in *S* will be wrapped up *by* a special tag *view*, and then written into *str*.

BiXJ Library

- A library is defined in a single XML file
  - The root element must be: <library>
  - Under the root element is a list of function declarations
    - The format of function declarations has been shown with the construct of xfunapp.

Library Import

- In the code file, a library is imported by the following instruction:
  - <?import library="*lib-name*.bix"?>

Type Definition

- The type for a source document is defined in an XML file
- The type file should be included in the corresponding source XML file with the following instruction:
  - <?import srctype=" *ty*.xml"?>

Two notations: *Ty* and *Att*

- *Ty* represents one of the following types:
  - Element type, unit type, string type, sequence type, choice type, recursive type
- *Att* represents either an attribute type or a sequence of attribute types

Element type

<TyElement [occurrence= "*"]>
<label>*tag*</label>
Ty
Att
</TyElement>

Unit Type

<TyUnit> </TyUnit>

String (or Text) Type

<TyString [occurrence= "*"] />

Sequence Type

<TySeq [occurrence= "*"] >
$Ty_1 \cdots Ty_n$
</TySeq>

Choice Type

<TyChoice [occurrence= "*"] >
$Ty_1 \cdots Ty_n$
</TyChoice>

Recursive Type

<TyRec [occurrence= "*"] >
<var>*tyvar*</var>
Ty
</TyRec>

Attibute

<TyAtt>
<name>id</name>
<requirement>
[implied | required]
</requirement>
</TyAtt>

Attribute Sequence

<TySeq>
$Att_1 \cdots Att_n$
</TySeq>

End