

XML-based Variant Configuration Language (XVCL)

Specification Version 2.10

June 5, 2006

Table of Contents

1	Learning XVCL - how you should read this document.....	5
2	XVCL overview	5
2.1	Defining the processing flow	6
2.2	An ancestor-descendent x-frame hierarchy	7
3	Preliminary definitions.....	7
4	Rules for writing well-formed x-frames	8
4.1	The x-frame structure.....	8
4.2	Well-formed XVCL commands.....	8
4.3	Validity Checking	10
5	Notations for defining XVCL commands.....	10
5.1	Meta symbols	11
5.2	Non-terminal symbols.....	11
5.3	Terminal symbols.....	11
6	Comments in XVCL	11
7	<x-frame> command.....	11
7.1	Syntax	11
7.2	Command description	12
7.3	Attributes definition	12
8	Definition of <adapt>, <insert>, <insert-before> and <insert-after> commands .	15
8.1	Syntax	15
8.2	Command description:	16
8.3	Attribute definition for <insert-before>, <insert> and <insert-after> commands:	21
8.4	Attribute definition for <break> command:.....	22
8.5	Attribute definition for <adapt> command:.....	22
9	Variables and expressions.....	24
9.1	Syntax	24
9.2	References to variables	25
9.3	Name Expressions.....	26
9.4	String Expressions	27
9.5	Evaluation of Arithmetic Expressions	29
9.6	How are expressions used?	29
9.7	Variable scoping rules.....	30
10	<value-of> command	32

10.1	Syntax	32
10.2	Command definition:	32
10.3	Attribute definition:	32
11	<set> command	33
11.1	Syntax	33
11.2	Command definition:	33
11.3	Attribute definition:	33
12	<set-multi command>	34
12.1	Syntax	34
12.2	Command description:	34
12.3	Attribute definition:	35
13	Additional rules for <set> and <set-multi> commands	36
14	<select> command	37
14.1	Syntax	37
14.2	Command description:	37
14.3	Attribute definition for <select> command:	38
14.4	The attributes of the <option> command:.....	38
15	<while> command	39
15.1	Syntax	39
15.2	Command description:	39
15.3	Attribute definition:	43
16	Definition of <ifdef> command	43
16.1	Syntax:	43
16.2	Command description:	43
16.3	Attribute definition:	43
17	Definition of <ifndef> command	43
17.1	Syntax:	43
17.2	Command description:	44
17.3	Attribute definition:	44
18	Definition of <remove> command.....	44
18.1	Syntax:	44
18.2	Command description:	44
18.3	Attribute definition:	45
19	Definition of <message> command.....	45
19.1	Syntax	45

19.2	Command description:	45
19.3	Attributes definition:	45
20	Processor options and configuration file.....	45
	Document Type Definition (DTD) for XVCL.....	47

1 Learning XVCL - how you should read this document

From this document you will learn how to create valid XVCL documents, called x-frames. This document is essentially a Reference Manual and should be read once you have already grasped basic XVCL concepts. Therefore, view the Demo on the XVCL Web Site and read the XVCL Tutorial paper first. The XVCL Tutorial describes a simple subset of XVCL and illustrates the usage of basic XVCL features. Browsing through the White Papers at the XVCL Web Site will also help you to start with XVCL - you will find there the rationale for using XVCL (and its predecessor frame technology) and examples of solutions. At this point, you should browse through this XVCL specifications document to roughly get an idea of advanced XVCL features that may be useful to you.

Having done that, you will know enough about XVCL to start thinking about the structure of the XVCL solution for your problem domain. As you develop your solution, you can refer to various sections of this XVCL specifications document. Build your solution incrementally. Start working with small sub-problems and simple variants. Once you have a solution for that, gradually extend the scope of experimentation.

Note that while principles of XVCL remain the same in all the application domains, the structure of the XVCL solution and the process that leads to arriving at the XVCL solution depends on the nature of application domain and types of variants to be addressed.

Our XVCL team at NUS will be glad to hear from you and work with you on problems of mutual interest. Contact us whenever you hit the problems and tell us about your successes and failures.

2 XVCL overview

When developing an XVCL solution, we partition a problem description (e.g., a software program) into generic, adaptable meta-components called x-frames. Each x-frame contains a fragment of problem description, called Textual Content. The Textual Content is written in a base language, which can be a programming language such as Java, a natural language such as English or any other language. While in examples presented in this manual the base language is Java, we stress that the basic principle of applying XVCL to manage variants in problems described in other base languages - remains the same.

Textual Content in x-frames is instrumented with XVCL commands for change. XVCL commands mark the anticipated variation points (also called “hot spots”) in x-frames, injecting flexibility into their Textual Contents. XVCL can be seen as a meta-language whose commands direct adaptation of x-frames. The x-frame adaptation process includes x-frame composition and customization.

X-frames related by <adapt> commands form an x-framework. If you use XVCL to support software product lines, then an x-framework will form your product line architecture. The specification x-frame, SPC for short, specifies what variant requirements you need in a specific system, a product line member. The SPC specifies how to adapt the x-framework in order to accommodate required variants.

The SPC becomes a root of an x-framework. During x-frame processing, the XVCL processor interprets the XVCL commands contained in the SPC, traverses an x-framework, performs adaptation by executing XVCL commands embedded in x-frames, and emits code components for a specific system, a member of the product line.

2.1 Defining the processing flow

XVCL processor starts processing with an SPC. XVCL commands in the SPC, and in each subsequently <adapt>ed x-frame, are processed in the sequence they appear in the x-frame. Whenever the processor encounters an <adapt> command in the currently processed x-frame, processing of the current x-frame is suspended and the processor will start processing the <adapt>ed x-frame. Once processing of the <adapt>ed x-frame is completed, the processor resumes processing of the current x-frame. When the XVCL processor reaches the end of the SPC - the processing is completed. Figures 1a and 1b illustrate an x-framework and the processing flow.

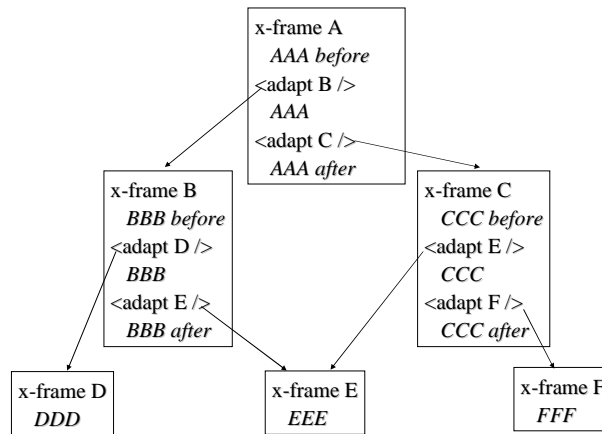


Figure 1a. An x-framework (B, C, D, E and F) and SPC (A)

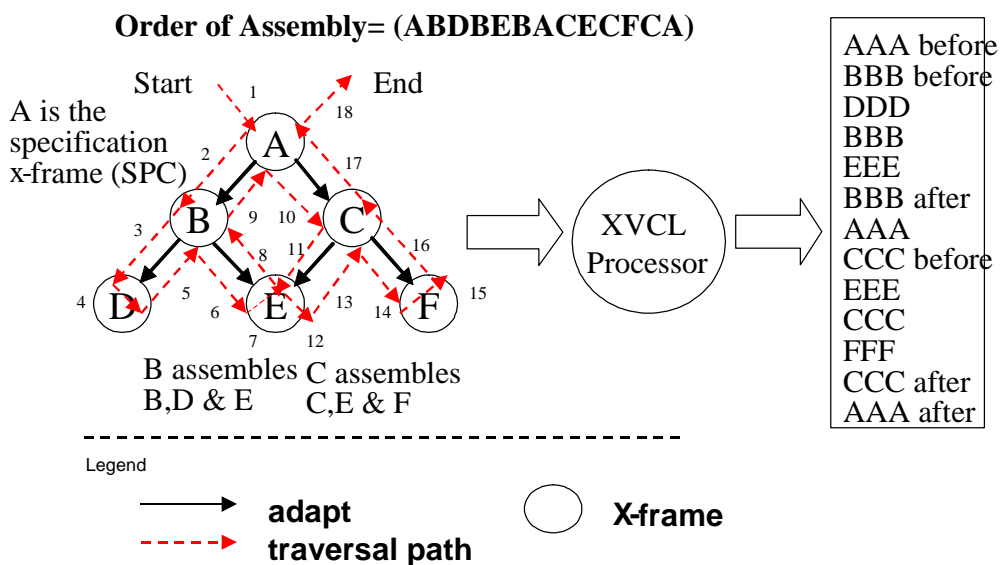


Figure 1b. Processing an x-framework

As the XVCL processor reads through x-frames and interprets XVCL commands, it also emits the output, according to the semantics of the XVCL commands, as described in this document.

The only exception to the above processing flow is when the processor interprets insert commands in <adapt> bodies, as explained in Section 8 .

For a given SPC, the processing flow is a trace that goes through the Content of visited x-frames (starting with SPC). Suppose p1 and p2 are two, not necessarily distinct, points in the same or different x-frames. We say that:

p1 **precedes** p2 (and p2 **follows** p1) if p1 appears before p2 in the processing flow. We use notation: $p1 \rightarrow p2$ to indicate processing flow from p1 to p2.

2.2 An ancestor-descendent x-frame hierarchy

Given an x-framework and SPC, for any two x-frames X and Y, we say that:

- X is an **ancestor** of Y, if X <adapt>s (directly or indirectly) Y in the processing flow defined by the SPC; we call Y a **descendant** of X
- X is an **immediate ancestor** (or **parent**) of Y, if X directly <adapt>s Y in the processing flow defined by this SPC; we call Y an **immediate descendant** (or **child**) of X. Note that for X to be a parent of Y, X must contain an <adapt> command whose name attribute yields the name of x-frame Y.

In Figure 1, x-frame A is an ancestor of all other x-frames. X-frame E has two immediate ancestors namely, x-frames B and C.

3 Preliminary definitions

Textual Content – can be any text that is intended to be customized by XVCL commands. This text may include anything including symbols that XML reserves for its own use, namely .<., .>, .&., double quote (“) and single quote (‘). These symbols are not interpreted as XVCL commands by the processor 2.10 or later. For compatibility with earlier versions, you can still keep those symbols in CDATA sections. The processor ignores XVCL commands if they are written inside CDATA section.

XVCL commands – are XVCL commands that are not written inside CDATA sections and are meant to be interpreted by the processor.

XVCL reserved characters or symbols – are characters that are reserved for use in the XVCL language such as “[”, “?” and “@”. XVCL uses “?” and “@” characters in Expressions and “[” character in <select> command. The effect of using them in the attributes of XVCL commands will be explained in relevant sections throughout the document.

Variable name - is a mixture of any characters but “?”, “@” and “,”.

4 Rules for writing well-formed x-frames

In the rest of this document, we assume that the reader has a basic understanding of XML¹. XVCL is a dialect of XML, so the usual XML rules apply to XVCL. In addition, well-formed x-frames must obey rules that are specific to XVCL.

XVCL commands are written as XML tags. While the syntax structure and semantics of the Textual Content does not matter, XVCL commands must follow the rules of the XVCL language.

4.1 The x-frame structure

Each x-frame is stored in a separate file. All the x-frames (including SPC) have the code skeleton structure shown in Figure 2.

```
<x-frame name="abc">
<!--x-frame body -- >
</x-frame>
```

Figure 2. The structure of an x-frame

Each file contains exactly one x-frame. The processor does not check or use the x-frame name, only the name of the file that contains an x-frame. But it is a good practice to make the x-frame name the same as the name of a file that contains a given x-frame. It is also a good practice to add “xvcl” extensions to a file that contains an x-frame, but the processor does not check that, either.

Each x-frame has exactly one x-frame body enclosed within x-frame start and closing tags, as indicated in Figure 1. The x-frame body is the Textual Content of an x-frame instrumented with XVCL commands for flexibility. Each x-frame contains one root tag called `<x-frame>`, enclosing all other XVCL command tags, except `<x-frame>`.

The above also implies that x-frames can not be nested.

The line `<!-- the body of x-frame -- >` is a comment.

4.2 Well-formed XVCL commands

As in XML, each XVCL command start tag must have its corresponding closing tag such as:

```
<x-frame name="..">
...
</x-frame>
```

Tagged structures must be properly nested, i.e., they are not allowed to overlap with each other.

XVCL tags are case sensitive. That is, the case of the start tag and its corresponding end tag must be the same. So, if we write, for example:

¹ For more information on XML please refer to “www.w3c.org/TR/REC-xml”

`<break> </BREAK>`

the XVCL processor will report an error.

XVCL commands may include attributes to provide the additional processing information. Attributes are written inside the angle brackets of the command tag. The attributes may appear in any order. In the example below, we define a break point named “break_a”: the `<break>` command tag has the attribute “name” which is assigned a value “break_a”.

```
<break name="break_a">
```

...

```
</break>
```

An empty tag (i.e., one with no contents between the begin tag and end tag) can be written as `<tag />`. For example, the empty tag for `<break>` can be written as:

```
<break name="break_a"/>
```

Here are examples illustrating the above rules. Example (a) shows a well-formed x-frame, while examples (b), (c), (d), (e), (f) and (g) contain errors.

```
<x-frame name="A">
  <adapt x-frame = "B">
  </adapt>
</x-frame>
```

Example (a) is well-formed because it has only one root element `<x-frame>` and the tags do not overlap.

```
<x-frame name="A">
  <adapt x-frame= "B">
</x-frame>
  </adapt>
```

Example (b) is not well-formed because the tags overlap.

```
<x-frame name="A">
  <adapt x-frame="B">
  </adapt>
```

Example (c) is not well-formed because the end tag of `<x-frame>` is missing.

```
<x-frame name="A">
  <adapt x-frame = "B">
  </adapt>
</x-frame name= "A">
```

Example (d) is not well-formed because the `</x-frame>` tag must not contain an attribute.

```
<x-frame name="A">
```

```
<value-of var />
</x-frame >
```

Example (e) is not well-formed because the attribute **var** in the `<value-of>` tag does not have a value assigned to it.

```
<x-frame name="A">
  <value-of <value-of var="x"/> />
</x-frame >
```

Example (f) is not well-formed because the `<value-of>` tag contains another angle brackets in it.

```
<x-frame name="A">
  <adapt x-frame="B"/>
</X-FRAME >
```

Example (g) is not well-formed because the start and end tags of `<x-frame>` do not match each other.

4.3 Validity Checking

XVCL is supported by a processor that traverses an x-framework according to specifications contained in the SPC and interprets XVCL commands embedded in visited x-frames. The main task of the XVCL processor is to customize and assemble the customized result of x-frames. But before that, the XVCL processor checks the validity of the x-frame file to ensure it obeys XVCL grammar rules. An XVCL file is valid if it conforms to the rules mentioned in its corresponding DTD. The DTD defines XVCL command syntax, command attributes, and valid command nesting rules. The DTD is case sensitive, which means that the letter case of the commands in XVCL files must appear exactly as in the DTD).

By default, all XVCL commands are in small letters. That is the XVCL processor will only understand commands written in small letters. However, the developer could change the cases of the command declarations in DTD to any desired case. When changing the case sensitivity, the developer should not change the format (grammar rules) of the DTD because this may change the processing structure of XVCL processor and may lead to undesired results.

5 Notations for defining XVCL commands

In the following sections, we define XVCL commands. For each command, we first specify command's syntax structure and then describe the meaning of the command and the role of its attributes.

We use the following Extended BNF conventions to specify the syntax structure of XVCL commands:

5.1 *Meta symbols*

Definition symbol is :=, e.g., A := B.

Short form may be used if all the left-hand-side symbols are defined in the same way, e.g., A, B, C := D.

Alternative symbol is |, e.g., A:= B | C | D

Repetition (0 or more times) symbol is * ,e.g., A := B*

Repetition (1 or more times) symbol is + ,e.g., A := B+

Grouping is symbolized by round brackets (and), e.g., A := (B | C)*

An optional part of command definition is symbolized by square brackets []

5.2 *Non-terminal symbols*

Non-terminals are written using a mixture of lower and uppercase letters, digits and symbol -. For readability, non-terminals representing command attribute values are written in *italics*, e.g., *dir-name*.

All other non-terminals are written in regular font, e.g., x-frame.

5.3 *Terminal symbols*

Terminals represent keywords (command names and command attribute names), identifiers, special symbols, etc. For readability, keywords are written in **bold**. Special symbols such as “, <, >, etc. are written as are, without quotation symbols - as long as this does not lead to ambiguities. Other terminals are written in capital letters such as IDENT, STRING, etc. Terminals representing command attribute values are written in *italics*, e.g., *X-FRAME-NAME*.

6 **Comments in XVCL**

XVCL comments are written between <!-- and -->. A comment can occupy a single line:

```
<!-- This is a single line comment in XVCL -->
```

or may spread over many lines:

```
<!-- This is a  
multi-line comment in XVCL -->
```

7 **<x-frame> command**

7.1 *Syntax*

```
x-frame := <x-frame name = “X-FRAME-NAME” [ outdir = “dir-name” ]  
[ outfile = “file-name” ] [ language = “language-name” ]>
```

x-frame-body

</x-frame>

<i>X-FRAME-NAME</i>	:= STRING – character string is a mixture of any characters but “?”, “@” and “,”.
<i>dir-name, file-name, language-name</i>	:= Expression
Expression	:= defined later in this document
x-frame-body	:= (textual-content break adapt set set-multi select ifdef ifndef value-of message while remove)*

7.2 Command description

The **<x-frame>** command defines the start and end of an x-frame. **<x-frame>** is the root tag of an x-frame. An x-frame body contains all other XVCL commands (except **<x-frame>**) inter-mixed with Textual Content.

7.3 Attributes definition

name = “*X-FRAME-NAME*”

This attribute specifies the name of the x-frame. The *X-FRAME-NAME* must be a STRING.

In **<adapt>** command, we specify the name of a file containing the **<adapt>**ed x-frame rather than the *X-FRAME-NAME*. Therefore, the processor does not check or use the *X-FRAME-NAME*. But it is a good practice to make *X-FRAME-NAME* the same as the name of a file that contains a given x-frame.

outdir = “*dir-name*”

This attribute specifies a directory where the XVCL processor will store the file with the output emitted from the current x-frame. The *dir-name* must be an Expression.

If the specified directory does not exist, the XVCL processor will create one. The *dir-name* can be either an absolute path like “c:\mydir\test” or a partial path, including the null path, for example the name “test” or “xvcl\test”.

If *dir-name* is a partial path, the XVCL processor will append it to the output directory path of the current x-frame’s parent. When the processor traverses back up the x-framework, it removes the directory paths that are appended at each level. This way, the directory structure may grow as the processor traverses down and shrink accordingly when the processor traverses back up the x-framework, placing outputs from various x-frames in different directories. For example, suppose the parent’s output directory is “c:\mydir\”. Defining “test” in **outdir** attribute of the current x-frame will make the XVCL processor emit the output from the current x-frame into the file that will be placed in “c:\mydir\test” directory. This feature is useful, for example, when the XVCL processor needs emit Java code from different x-frames into different directories.

If *dir-name* is an absolute path, the XVCL processor emits output of the current x-frame into this directory. The current output directory is set to be this path and

descendent x-frames can append their own partial paths to it. In this way, subsequently processed x-frames can create new output directories, as desired if these do not exist. Figure 3 shows the output directory structure created by the XVCL processor, assuming that SPC's **outdir** attribute is defined as "c:\abc" and the descendent x-frames A, B, C, D and E define their **outdir** attribute as "a", "b", "c", "d" and "e", respectively.

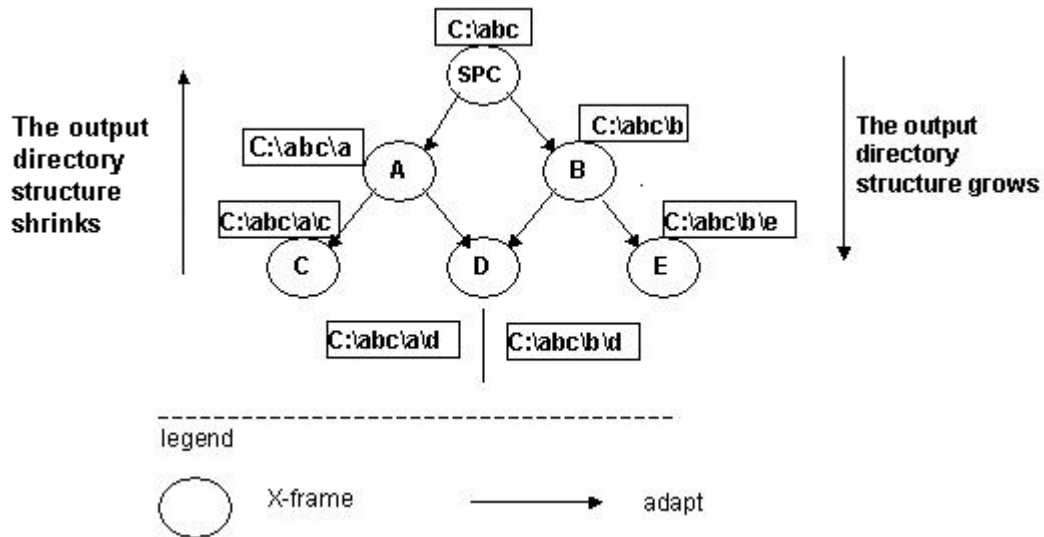


Figure 3. An example of the output directory structure

If the **outdir** attribute is not defined in the current <x-frame>, then the parent's output directory path will be used.

If no ancestor x-frame, including SPC, has defined an output path in the **outdir** attribute, the processor will emit output from the current x-frame (as well as from all its ancestors) to a directory where SPC is stored.

If the *dir-name* of SPC's **outdir** attribute is a partial path, for example "test", then the processor will append "test" to the directory path where SPC is stored and emit output from the SPC to that directory.

When both the <adapt> command in the parent x-frame and the adapted <x-frame> command define the output directory path in their **outdir** attributes, the <adapt> command's output directory path will be used.

For example, if x-frame X <adapt>s Y as follows:

```
<adapt x-frame="Y.xvcl" outdir="c:\ancestor\" outfile="b.java"/>
```

and x-frame Y is defined as:

```
<x-frame name="Y" outdir="c:\descendent\"/>
```

then the output from x-frame Y will be emitted to the directory "c:\ancestor\b.java".

Value *dir-name* is often defined by an XVCL Expression that involves variables. In such a case, an ancestor x-frame can override *dir-names* in the descendant x-frames via variables that are <set> in ancestor x-frames and referenced in *dir-name* Expressions in descendent x-frames.

outfile = “file-name”

This attribute specifies a file to which the XVCL processor will emit the output from the current x-frame. The *file-name* must be an Expression that yields a legal file name or absolute directory path including the file name. Partial (relative) paths are not allowed in the **outfile** attribute.

An extension (if any), is treated as an integral part of the file name.

When the processor emits output to file *file-name* for the first time in processing flow, and file *file-name* exists, the existing file is deleted before the output is emitted. If any subsequently processed x-frame emits output to the same file during the same run of the processor, the output is appended to the contents of that file.

If **outfile** is omitted, the processor checks the **outfile** attribute of the parent x-frame that <adapt>s the current one as follows: If **outfile** attribute is defined in the parent’s command that <adapt>s the current x-frame, that file name will be used. Otherwise, the parent’s <x-frame> command’s output file will be used, if defined. If an output file name is not defined in any of the ancestor x-frames including SPC, the SPC’s name will be used as the *file-name*.

When both the parent’s x-frame <adapt> command and the current <x-frame> command define the output file name in their **outfile** attributes, the parent’s <adapt> command’s output file name will be used. This is consistent with XVCL’s variable scoping rules (see Section 9.7).

For example, if x-frame X <adapt>s Y as follows:

```
<adapt x-frame=“Y.xvcl” outfile=“a.java”/>
```

and x-frame Y is defined as:

```
<x-frame name=“Y” outfile=“b.java”/>
```

then the output from x-frame Y will be emitted to the file “a.java”.

language = “language-name”

This attribute specifies the base language in which the Textual Content of the x-frame is written (e.g., text, java, xmi, vb, etc).

Currently, this attribute is ignored by the XVCL processor. It merely gives a hint to the developer how to interpret the Textual Content. In the future, we envision that the processing of x-frames may be tailored to the needs of different base languages via this attribute.

8 Definition of <adapt>, <insert>, <insert-before> and <insert-after> commands

Commands <adapt>, <insert>, <insert-before> and <insert-after> are always used together, that is why we define them in one section.

8.1 Syntax

adapt := **<adapt** **x-frame="file-name"** [**outdir="dir-name"**]
[**outfile="file-name"**] [**samelevel="yes-no"**] [**once="yes-no"**] **src="yes-no"**[>

adapt-body

</adapt>

file-name := Expression

yes-no := Expression

adapt-body := (insert | insert-before | insert-after)*

insert := **<insert break="break-name">**

Insert-Content

</insert>

break-name := Expression

insert-before := **<insert-before break="break-name">**

Insert-Content

</insert-before>

insert-after := **<insert-after break="break-name">**

Insert-Content

</insert-after>

Insert-Content := (textual-content | break | adapt | set | set-multi | select |
ifdef | ifndef | value-of | message | while | remove)*

break := **<break name="break-name">**

Break-Content

</break>

Break-Content := (textual-content | adapt | set | set-multi | select | ifdef | ifndef | value-of | message | while | remove)*

8.2 Command description:

<adapt>

The <adapt> command instructs the processor to:

- process the x-frame specified as the value of the **x-frame** attribute, *file-name*,
- process all the descendent x-frames of the above x-frame,
- perform customizations of visited x-frames as specified in the body of the <adapt> command (subject to the rules described further in this section), and
- emit the output to the specified output file.

Note that all the x-frames are read-only. That is, when the x-frame is adapted, the processor does not make changes on the original x-frame. Rather, it reads the content of the x-frame and performs necessary operations in memory before writing it to the output file.

An x-frame is not allowed to <adapt> itself or any of its ancestor x-frames, that is recursive adaptations are not allowed.

The *file-name* of the <adapt>ed x-frame given in attribute **x-frame** must be an Expression.

<insert>, <insert-before> and <insert-after>

The body of an <adapt> command may contain zero or more of <insert>, <insert-before> and <insert-after> commands. As many rules are the same for all three commands <insert>, <insert-before> and <insert-after>, for brevity of the description, we shall use term **insert-xxx** to refer to any of the three commands.

The **insert-xxx** commands modify <adapt>ed x-frames at break points identified by matching <break> commands. This matching between **insert-xxx** and <break> commands is established by values of attribute **break** of **insert-xxx** and **name** of <break> commands. The exact matching rules will be described later in the section.

The <insert> command replaces the Break-Content inside its matching <break> commands with its Insert-Content.

The <insert-before> command inserts its Insert-Content before the matching <break> commands.

The <insert-after> command inserts its Insert-Content after the matching <break> commands.

Notice that the <insert-before> and <insert-after> commands do not replace the Break-Content inside its matching <break> commands.

The **insert-xxx** commands can only appear inside the body of <adapt> commands. There is no limit to how many **insert-xxx** commands can be contained in one <adapt> command.

The body of an `<adapt>` command may contain multiple **insert-xxx** commands referring to the same *break-name*. Similarly, **insert-xxx** commands referring to the same *break-name* may exist in many different `<adapt>` commands, placed in many x-frames.

<break>

The `<break>` command marks a place in the x-frame (called a break point) at which the x-frame can be customized by an `<insert>`, `<insert-before>` or `<insert-after>` command declared in the ancestor x-frames.

Details of how these commands work together

The **insert-xxx** commands are matched with subsequently processed `<break>` commands in the descendent x-frames via the *break-name* defined in the **break** attribute of these commands. In addition, to the *break-name*, scoping rules are used to determine which **insert-xxx** commands match which `<break>` commands. We describe the matching process later in this section.

The following are the details of how insertions are processed:

- If an `<insert-before>` command matches the `<break>` command, then the Insert-Content of the `<insert-before>` command will be processed as if it were inserted just before the matching `<break>` in the x-frame.
- If an `<insert>` command matches a `<break>` command, then the Insert-Content of the `<insert>` command will be processed as if it replaced the Break-Content of the `<break>` command. (If the Insert-Content of the `<insert>` command is empty, it is equivalent to deleting the Break-Content of the `<break>` command.)
- If no `<insert>` command matches a particular `<break>` command, the processor will process its Break-Content.
- The `<insert-after>` works analogous to `<insert-before>`.

For all the **insert-xxx** commands, the inserted Insert-Content will be processed as if the Insert-Content had originally appeared at the insertion point in the target x-frame.

If multiple `<insert>` commands within the body of the same `<adapt>` command refer the same `<break>`, then the Insert-Content of all those commands is concatenated before the insertion is made at the matching `<break>` command. The same applies to `<insert-before>` and `<insert-after>` commands. This is illustrated in the following example.

x-frame A:

```

<adapt x-frame="B.xvcl">
  <insert-before break="x">
    xAB before
  </insert-before>
  <insert break="x">
    xAB
  </insert>
  <insert-after break="x">
    xAB after
  </insert-after>
  <insert break="x">
    xAB again
  </insert>
</adapt>

```

xAB is concatenated with xAB again

x-frame B:

```

before break in B
<break name="x">
  break-content
</break>
after break in B

```

result:

```

before break in B
xAB before
xAB } concatenated Insert-Content from <insert>
xAB again } commands into <break> x
xAB after
after break in B

```

Comments:

Declaring multiple **insert-xxx** commands to the same break point within the same <adapt> command will result in concatenation of the contents of all the <insert>s, before any insertion is conducted. That is to say, these <insert> declarations will be regarded as one <insert> if the target break point is the same.

Referring to above example, x-frame A has two <insert>s with the body "xAB" and "xAB again" that are defined to insert into break x in x-frame B. The first <insert>'s

“xAB” with the second <insert>’s “xAB again” are concatenated as if there was only one <insert> containing:

xAB

xAB again

Scoping rule for matching insert-xxx commands with <break>s

If <insert> commands refer to the same <break> by *break-name* from within multiple x-frames, only the <insert> command that is executed first in the processing flow will match this <break>. The remaining <insert> commands referring to the same <break> in other x-frames are ignored.

The following examples further demonstrate this matching rule:

The following example illustrates matching between <insert> commands and <break>s. The same rule applies to <insert-before> and <insert-after> commands.

x-frame A:

```
<adapt x-frame="B.xvcl">
  <insert break="x">
    xAB
  </insert>
</adapt>
<break name="x"/> // this <break> will be ignored.
```

x-frame B:

```
<adapt x-frame="D.xvcl">
  <insert break="x"> // this <insert> will be overridden by insert in x-frame A
    xBD
  </insert>
</adapt>
```

x-frame D:

```
<break name="x">
  break-content
</break>
```

result:

xAB

Comments:

In the above example, x-frames A and B declare <insert>s for <break> x. However, x-frame B’s <insert> command is overridden by x-frame A’s <insert> command because x-frame B is the descendent of x-frame A.

It is also important to note that the <break> command in x-frame A is not affected by <insert> command. This is because the <insert> commands only have affect to the adapted x-frame and its descendent x-frames.

Multiple insertions

If there is more than one <break> command with the same name, all the **insert-xxx** commands matching that <break> will make insertions to all those <break>s. This is called multiple insertions. This rule is illustrated in the following example.

x-frame A:

```
<adapt x-frame="B.xvcl">
  <insert-before break="x">
    xAB before
  </insert-before>
  <insert break="x">
    xAB
  </insert>
  <insert-after break="x">
    xAB after
  </insert-after>
</adapt>
```

x-frames B:

```
<adapt x-frame="D.xvcl">
</adapt>
  before break in B
  <break name="x">
    break-content
  </break>
  after break in B
```

x-frame D:

```
before break in D
  <break name="x">
    content.
  </break>
```

after break in D

result:

before break in D

xAB before

xAB

xAB after

after break in D

before break in B

xAB before

xAB

xAB after

after break in B

Comments:

If there are many `<break>`s with the same name in the descendent x-frames, the **insert-xxx** commands make insertions to all the matching `<break>`s.

A note on nested `<break>`s

Nested `<break>`s are not allowed. Though nested `<break>`s make sense at the first glance, they lead to more problems than they solve. For example, we might attempt to `<insert>` into an inner `<break>` that had been deleted by some other command. The following example demonstrates this:

Suppose x-frame A adapts x-frame B and x-frame B adapts x-frame C. Further, X-frame A contains `<insert>` command for `<break>` y and x-frame B contains `<insert>` command for `<break>` x and in x-frame C, we have the following nested `<break>`s:

```
<break name="x">
  <break name="y">
    </break>
  </break>
```

In the above situation, x-frame A's `<insert>` command will not be able to insert to `<break>` y, since it has been already superseded by the `<insert>` command in x-frame B.

8.3 Attribute definition for `<insert-before>`, `<insert>` and `<insert-after>` commands:

break = "break-name"

This attribute specifies the name of the `<break>` where the Insert-Content of **insert-xxx** should be inserted. The *break-name* must be an Expression.

8.4 Attribute definition for <break> command:

name = "*break-name*"

This attribute specifies the name of a break point. The *break-name* must be an Expression.

8.5 Attribute definition for <adapt> command:

x-frame = "*file-name*"

This attribute specifies the name of the file containing the x-frame to be <adapt>ed. The *file-name* should be Expression that yields either an absolute path including file name, a relative path or just a file name.

The processor looks for a specified file in the “designated directory”. The processor uses the following rules to determine the “designated directory”:

1. the designated directory for SPC is the directory where SPC x-frame is located.
2. suppose x-frame A, with designated directory C:\dir_A\, contains command **<adapt x-frame="B"/>**
 - a) if B is just a file name, then the processor looks for file B in C:\dir_A\; the designated directory of B becomes also C:\dir_A\;
 - b) if B is an absolute path, say C:\dir_B\b.xvcl, then the processor looks for file C:\dir_B\b.xvcl; the designated directory of B becomes C:\dir_B\
 - c) if B is a relative path, say dir_B\b.xvcl, then the processor looks for file C:\dir_A\dir_B\b.xvcl, where C:\dir_A\ is a designated directory of x-frame A; the designated directory of B becomes C:\dir_A\dir_B\

If the file extension is omitted, the processor will first look for the exact match and, if not found, for file named *file-name* with “.xvcl” extension. Otherwise, the processor will use the specified file name and the extension.

outdir = "*dir-name*"

This attribute specifies a directory where the XVCL processor will store the file with the output emitted from the <adapt>ed x-frame. The definition for this attribute is the same as the definition for the **outdir** attribute of the <x-frame> command (see Section 7.3). The *dir-name* must be an Expression.

outfile = "*file-name*"

This attribute specifies a file into which the XVCL processor will emit the output from the <adapt>ed x-frame. The definition for this attribute is the same as the **outfile** attribute of the <x-frame> command (see Section 7.3). The *file-name* must be an Expression.

samelevel = "*yes-no*"

This attribute specifies whether or not to raise the variables declared in the adapted <x-frame> to the current x-frame (that is one that contains the <adapt> command). The *yes-no* must be an Expression that yields value “yes” or “no”. If omitted, the default value is “no”.

If the value is “yes”, variables whose values are <set> (or <set-multi>) in the adapted x-frame are raised to the current x-frame. In other words, those variables behave as if they were <set> (or <set-multi>) at the place of <adapt> command in the current x-frame. Having raised the variables, the usual variable scoping rules will be applied to determine values of referenced variables (Section 9.7).

The following example illustrates this concept:

Suppose in **x-frame A** we have:

```
<set var="x" value="XA"/>
<adapt x-frame="B" samelevel="yes"/> // the variable x will be reset to "XB" here
<value-of expr="?@x?"/> // the value of x is "XB"
```

and in **x-frame B**:

```
<set var="x" value="XB"/>
```

The result of evaluation of <value-of expr="?@x?"/> is “XB”. For details about variables setting and their associated rules please refer to Section 11.

The example below further illustrates the effects of attribute **samelevel**.

x-frame A:

```
<set var="x" value="XA1"/>
  value of variable x is <value-of expr="?@x?"/> // value of x is XA1
  <adapt x-frame="B.xvcl" samelevel="yes"/>
  value of variable x is reset to <value-of expr="?@x?"/> // value of x is XB1
  value of variable y is <value-of expr="?@y?"/> // value of y is YB1
  <adapt x-frame="C.xvcl"/>
```

x-frame B:

```
<set var="y" value="YB1"/>
<set var="x" value="XB1"/>
```

x-frame C :

```
Because of same level, in x-frame C we have:
  value of variable x is <value-of expr="?@x?"/>
  value of variable y is <value-of expr="?@y?"/>
```

result:

value of variable x is XA1

value of variable x is reset to XB1

value of variable y is YB1

Because of same level, in x-frame C we have::

value of variable x is XB1

value of variable y YB1

Comments:

The net effect of raising variables is the same as if all the <set> commands from x-frame B appeared in x-frame A at the B's adaptation point. Thus, the value of x is reset to XB1. When adapting x-frame C in x-frame A, variable x with the new value XB1 together with variable y with value XY1 will be passed down to x-frame C.

once = "yes-no"

This attribute specifies whether or not the subsequent <adapt>s of the <adapt>ed x-frame should be ignored (value "yes") or not (value "no"). The *yes-no* must be an Expression that yields value "yes" or "no". The default value is "no".

Notice that if the value of attribute **once** is "no" (which is the usual case), the named x-frame can be adapted many times during processing of an x-framework.

src="yes-no"

This attribute tells whether the file to be adapted is a proper x-frame (no") or just any source file ("yes"). For "yes", the processor outputs file's contents without processing it. The default value is "no", meaning a proper x-frame.

9 Variables and expressions

Generic names increase flexibility and adaptability of programs and play an important role in building generic, reusable programs. XVCL variables and Expressions provide powerful means for creating generic names and controlling the x-framework customization process.

XVCL Expressions include STRING constants, Name Expressions (direct and indirect references to variables and chains of variable references), String Expressions (concatenations of Name Expressions and STRINGS) and Arithmetical Expressions. We use the term XVCL Expression (or Expression for short) to mean STRING, Name Expression, String Expression or Arithmetic Expression.

9.1 Syntax

Expression := Name-Expression | String-Expression | Arithmetical-Expression

Name-Expression := ?@ (STRING | @) * VAR-NAME ?

String-Expression := (Name-Expression | STRING) +

VAR-NAME := a mixture of any characters but "?", "@" and ","

STRING := a mixture of any characters but “?”, “@” and “,”

9.2 References to variables

A direct reference to variable C is written as: @C:

Each extra symbol ‘@’ in the front of a variable name indicates a level of indirection. So:

@@C means value-of (value-of (C))

@@@C means value-of (value-of (value-of (C))), and so on.

XVCL processor replaces references to variables by variables’ respective values. Here, we should mention, that XVCL processor stores all the variables defined so far in the Symbol Table along with their current values, as assigned to variables in <set> and <set-multi> commands (see Sections 11 and 12). Table 1 gives an example of Symbol Table we shall use in the examples.

<i>Name</i>	<i>Value</i>
A	X
X	Y
Y	Z
C	U
U	BU
BU	V
AV	W
AT	S
V	T
R	B?@@A?B?@C?
BG	H
BYBU	G
E	?@F?
F	?@G?
G	L

Table 1. The Symbol Table with variables

For example, the value of @@C is **BU** and the value of @@@C is **V**.

A reference to a non-existing variable in Symbol Table is considered an error in all situations.

9.3 Name Expressions

We shall introduce Name Expressions first and then explain String Expressions. A simple Name Expression may contain just a variable reference, such as: `?@C?` or `?@@C?`. (A Name Expression must be enclosed between question mark symbols '?'.)

More complex (but more useful) Name Expressions can be written as: `?@A@B@C?`. In that case, the value of such a Name Expression is computed from right to left as follows:

value-of (A | value-of (B | value-of (C)))

where symbol '|' means string concatenation. Notice that only C is treated as a variable name, while A and B are treated as strings.

For example, referring to the Table 1, the evaluation of Name Expression, `?@A@B@C?` is done as follows:

1. get the value of variable C
 - the intermediate result is U
2. concatenate B and U and get the value of variable BU
 - the intermediate result is V
3. concatenate A and V and get the value of variable AV
 - the final result is W.

After each evaluation step, the intermediate value computed is concatenated with the character string on the left to form a new variable name that is looked up in the Symbol Table. Evaluation of a Name Expression continues until the whole Name Expression is evaluated.

Examples below illustrate evaluation of Name Expressions. For detailed steps please refer back to previous example.

Example 1.

The evaluation of Name Expression `?@@A?` is as follows:

1. get the value of A which is X
2. get the value of X which is Y

So, the final result of the evaluation is Y.

Example 2.

The evaluation of Name Expression `?@A@@B@C?` is as follows:

1. get the value of C which is U
2. get the value of BU which is V
3. get the value of V which is T
4. get the value of AT which is S

So, the final the final result of the evaluation is S.

Deferred evaluation

A Name-Expression may include references to variables whose evaluation is deferred. Such variables are <set> (or <set-multi>) with the value of defer-evaluation attribute “yes”, for example:

```
<set var="E" value=" ?@F?" defer-evaluation="yes"/>
```

```
<set var="F" value=" ?@G?" defer-evaluation="yes"/>
```

An Expression defining the value of a deferred variable is evaluated at the variable reference point rather than at the point where the variable is <set>. For the above <set> commands, entries for variables E and F are indicated in Table 1.

Examples below illustrate how deferred evaluation affects evaluation of Name Expressions.

Example 3.

Suppose we have:

```
<set var="E" value=" ?@F?" defer-evaluation="yes"/>
```

```
<set var="F" value=" ?@G?" defer-evaluation="yes"/>
```

and the Symbol Table as shown in Table 1.

Evaluation of the Name Expression ?@E? is as follows:

1. get the value of variable E
 - the result is ?@F? whose evaluation is deferred
2. get the value of variable F
 - the result is ?@G? whose evaluation is also deferred
3. get the value of G
 - the final result is L.

In the above example, we have two deferred evaluations chained together. The chain of deferred evaluations may be of any length.

Referenced variable names created at each intermediate evaluation step must represent variables that exist in the Symbol Table, otherwise the XVCL processor reports an error. All such variables must have been defined in <set> or <set-multi> commands before a given Name Expression is evaluated.

Deferred evaluation is further explained in Sections 11.3 and 0.

9.4 String Expressions

String Expressions may contain any number of Name Expressions intermixed with character strings.

To evaluate a String Expression, we evaluate the Name Expressions from the left to the right of the String Expression, replace Name Expressions with their respective values and concatenate with character strings at the point of replacement.

Below, we illustrate evaluation of String Expressions with examples, assuming variable values indicated in Table 1.

Example 4.

We evaluate String Expression `?@A@B@C?P?@X?` as follows:

1. evaluate Name Expression `?@A@B@C?`
 - the result is W
2. replace `?@A@B@C?` with W
 - partially evaluated String Expression becomes `WP?@X?`
3. evaluate Name Expression `?@X?`
 - the result is Y
4. replace `?@X?` with Y
 - the final result is WPY.

Example 5.

The evaluation of a String Expression `"B?@@A?B?@C?"` is as follows:

1. evaluate the value of `?@@A?`
 - the result is Y
2. replace `?@@A?` with Y
 - partially evaluated String Expression becomes now `BYB?@C?`
3. evaluate the value of `?@C?`
 - the result is U
4. replace `?@C?` with U
 - the final result is BYBU.

Example 6 (with deferred evaluation)

Suppose we have:

```
<set var="R" value=" B?@@A?B?@C?" defer-evaluation="yes"/>
```

The evaluation of the Name Expression `?@B@@R?` is as follows:

1. get the value of R
 - the result is `B?@@A?B?@C?`
2. evaluate the value of `B?@@A?B?@C?` as shown in Example 5
 - the result is BYBU
3. replace `@R` with BYBU
 - partially evaluated Name Expression becomes now `?@B@ BYBU?`
4. get the value of variable BYBU
 - the result is G
5. replace `@BYBU` with G
 - partially evaluated Name Expression becomes now `?@BG?`
6. get the value of BG

- the final result is H.

9.5 Evaluation of Arithmetic Expressions

If an Expression is a well-formed Arithmetic Expression - it will be accepted as such and the XVCL processor will evaluate its value.

A well-formed arithmetical expression is formed with the following five operators +, -, /, * and ^ (power). The operands can be either integers or decimals. Strings should represent integer or decimal numbers. References of variables and expressions must yield an integer or decimal numbers.

All decimal answers are rounded down to the nearest whole number.

The XVCL processor will report an error if a division by zero occurs.

Nested parenthesis can be used in expressions to indicated the grouping of operators and operands as in the examples below:

Suppose we have the following set commands:

```
<set var="x" value="6"/>
```

```
<set var="y" value="4"/>
```

Example 1: $(?@x? + ?@y?)/5$

Example 2: $(3 * (1.2 + 3.4))/4$

In example 1, the values of variable x and y are added which yields 10 and the result is divided by 5, yielding 2.

In example 2, the calculation is done on the innermost parenthesis, yielding the value 4.6. Next 4.6 is multiplied with 3 and 15 is then divided by 4, yielding 3.

Note that the attributes all XVCL commands can accept Arithmetic Expressions. However, Arithmetic Expressions should not be used in the <while> and <select> commands' attributes. Otherwise, they will be calculated and the resulting values will be passed to attributes of those commands.

If the Arithmetic Expression is not valid, (e.g., Arithmetic Expressions that contain strings) the processor will regard the whole Arithmetic Expression as a string and emit it to the output. For example, if the processor encounters the Arithmetic Expression $(a + 3) * c$, the processor cannot evaluate it since it is not a valid Arithmetic Expression. So $(a + 3) * c$ is regarded it as a string and emit to its output.

All scientific calculations such as **mod, tan, cos, etc are not supported** since XVCL is not intended to be a programming language.

9.6 How are expressions used?

Expressions (Name Expression, String Expression and Arithmetic Expression), rather than character strings, are commonly used in attributes of XVCL commands (except the name attribute of <x-frame> command) to denote x-frame names to be adapted, pathnames, variable values, variable names, break names etc. Below is an example of using Name Expression in <adapt> command:

```
<adapt x-frame="?"@x-frame-name?" outdir="?"@dir-name?" outfile="?"@fine-name?" >
```

9.7 Variable scoping rules

Variable scoping rules are the same for both single-value and multi-value variables. The <set> command in the ancestor x-frame takes precedence over <set> commands in its descendent x-frames. That is, once an x-frame A sets the value of variable x, <set> commands that <set> the same variable x in descendent x-frames (if any) will not take effect. However, the subsequent <set> commands in x-frame A can reset the value of variable x. This is illustrated in the following example: (Note the we use “//” java style comment to explain the codes inside x-frames. They are there to help the reader to understand the code)

x-frame A:

```
<set var="x" value="XA1"/> // value of x is set to XA1
<adapt x-frame="B.xvcl"/> // value of x is XA1 in x-frame B and its descendents
<set var="x" value="XA2"/> // values of x is reset to XA2
<adapt x-frame="C.xvcl"/> // value of x is XA in x-frame C and its descendents
value of variable x in x-frame A is <value-of expr="?"@x?"/>
```

x-frame B:

```
value of variable x is <value-of expr="?"@x?"/> in x-frame B
<adapt x-frame="E.xvcl"/>
```

x-frame C:

```
value of variable x is <value-of expr="?"@x?"/> in x-frame C
<adapt x-frame="E.xvcl"/>
```

x-frame E:

```
value of variable x is <value-of expr="?"@x?"/> in x-frame E
```

result:

```
value of variable x is XA1 in x-frame B
value of variable x is XA1 in x-frame E
value of variable x is XA2 in x-frame C
value of variable x is XA2 in x-frame E
final value of variable x in x-frame A is XA2
```

Comments:

Notice that x-frame E is adapted by x-frame B and x-frame C. When x-frame E is adapted by x-frame B, the value of variable x in x-frame E is XA1, which is passed

down by x-frame B. When x-frame E is adapted from x-frame C, the value of variable x in x-frame E becomes XA2. This is because the previous value XA1 of variable x in x-frame A is reset again just before x-frame B is adapted again.

Variables become undefined as soon as the processing returns to the parent of the x-frame that effectively set those variables.

Note that variables that are set within <insert> commands become undefined when the x-frame containing the <break> returns the processing to its parent x-frame. This is illustrated in the example below.

x-frame A:

```
<adapt x-frame="B.xvcl">
  <insert break="x">
    <set var="VA" value="XA" />
  </insert >
</adapt>
<!-- <value-of expr="?@VA?"/> -- >// variable VA is out of scope here
```

x-frame B:

```
<set var="VA" value="XB" />
value of VA before break x is <value-of expr="?@VA?"/> in x-frame B // here VA is
                                                                    //XB
<break name="x"/> // the insert content, <set> command in x-frame A is inserted
value of VA after break x is <value-of expr="?@VA?"/> in x-frame B // here VA is
                                                                    //XA
<adapt x-frame="D.xvcl"/>
```

x-frame D:

```
value of aa is <value-of expr="?@VA?"/> in x-frame D
```

result:

value of VA before break x is XB in x-frame B
value of VA after break x is XA in x-frame B
value of VA is XA in x-frame D

Comments:

The <set> command in x-frame A is inserted to the break x in x-frame B. X-frame B also has the <set> command that sets the variable VA to XB. The value of VA is XB before the <break> x in x-frame B. After the <break> x in x-frame B, the value of VA becomes XA because the <insert> command in x-frame A has inserted the <set> command that sets the variable VA's value to XA.

When `<set>` commands are inserted into a `<break>`, these commands behave as if they are written in the place of `<break>` command. It is very important to note that ANY XVCL COMMANDS that are inserted to break `x` will be processed as if they are originally written at the insertion point in the target `x`-frame.

In the case of `<insert-before>` command, the `<set>` will behave as if it is originally written just before the `<break>` `x` in `x`-frame `B`, where as, in the case of `<insert-after>` the `<set>` command will behave as if it is originally written just after the `<break>` `x` in `x`-frame `B`.

The variable scoping rules are important for reuse. To be reused in many contexts, `x`-frames are heavily parameterized by variables and Expressions. Typically, names of adapted `x`-frames, break points, etc. are represented by Expressions rather than STRINGS. `X`-frames use `<set>` (and `<set-multi>`) commands to define default values for variables. Ancestor `x`-frames can override the defaults, if necessary, adapting an `x`-frame to the specific reuse context.

10 `<value-of>` command

10.1 Syntax

`value-of` := `<value-of expr= "Expression" />`

10.2 Command definition:

The value of the Expression is evaluated and emitted to the output. The evaluated result replaces the command. Expressions are described in Section 9.

For example,

```
<set var="name" value="John"/>
```

```
My name is <value-of expr="?@name?"/>.
```

The result after processing will be:

```
My name is John.
```

The same result can be produced by including "My name is" to the **expr** attribute which forms a String Expression as in the following example:

```
<value-of expr="My name is ?@name?."/>
```

The `<value-of>` command is an empty tag and cannot contain any content between its start tag and end tag.

10.3 Attribute definition:

expr = "Expression"

This attribute specifies an Expression to be evaluated (see Section 9).

11 <set> command

11.1 Syntax

```

set      := <set var="single-var-name" value="value" [defer-evaluation="yes-no"]/>
single-var-name := Expression
value      := Expression
yes-no     := Expression

```

11.2 Command definition:

The <set> command assigns a “value” defined in **value** attribute to single-value variable “single-var-name” defined in **var** attribute.

The <set> command is an empty tag and cannot contain any content between its start tag and end tag.

When a variable is <set> for the first time, say in x-frame A, the variable is entered into the Symbol Table along with its value. For any subsequent command in the same x-frame that <set>s value of the same variable, the processor updates the value of the variable in the Symbol Table.

Suppose the processor executes the following <set> command in x-frame A:

```
<set var="x" value =“X1”/>
```

If variable x is not in the Symbol Table, variable x is entered along with its value X1.

Suppose another <set> command in the same x-frame is subsequently executed:

```
<set var="x" value=“X2”/>
```

This time the processor will just updates the value of x to X2.

The <set> commands in A’s descendent x-frames will normally be ignored and the variable is removed from the Symbol Table once the processing of an x-frame that inserted the variable to the Symbol Table is completed. But there are exceptions from this rule. We refer the reader to the Section 9.7 on variable scoping, Section 8.5 on attribute **samelevel** and Section 16 on <remove> command for further details.

11.3 Attribute definition:

var =“single-var-name”

This attribute specifies the name of a single-value variable. The *single-var-name* must be an Expression that yields a legal variable name. A legal variable name is a mixture of any characters but “?”, “@” and “,”.

value =“value”

This attribute specifies the value of the variable. The “value” must be an Expression.

The evaluation of the variable’s “value”, can be deferred as explained in defer-evaluation attribute below.

defer-evaluation = "yes-no"

This attribute specifies if the evaluation of the variable's value should be deferred (indicated by value "yes") or not (indicated by value "no"). The *yes-no* must be an Expression that yields value "yes" or "no". The XVCL processor reports an error otherwise. If omitted, the default value is "no".

The value of that variable is evaluated at the reference point rather than at the point where the variable is <set>. The processor stores the Expression in the Symbol Table and the evaluation takes place each time the variable is referenced.

For example, as the result of the following <set> command:

```
<set var="x" value="?"@y?" defer-evaluation="yes"/>
```

the Expression ?@y? is entered into the Symbol Table as is, without evaluating its value. At each point of reference to variable x, the processor evaluates Expression ?@y? to come up with the value for variable x at a given reference point.

If **defer-evaluation** is "no", the current value of variable y is assigned to x.

We refer the reader to Sections 9.3 and 9.4 for more examples of deferred evaluation.

12 <set-multi command>

12.1 Syntax

```
set-multi      :=      <set-multi var="multi-var-name" value="value (, value)*"
                        [defer-evaluation="yes-no"]/>
multi-var-name :=      Expression
value          :=      Expression
```

12.2 Command description:

The <set-multi> command defines the multi-value variable. The multi-value variables are mainly used in <while> commands for iteration (see Section 15)

The <set-multi> command assigns a list of values specified in the **value** attribute to *multi-var-name*. Values must be separated by comma ','. The values are Expressions.

If a specific *value* is a reference to a multi-value variable, e.g., ?@MULTI?, then the whole list of values of variable MULTI is included into the list of values of the *multi-var-name* (unless variable MULTI happens to be a control variable in the <while> loop enclosing a give <set-multi> command, see Section 15).

The *values* are processed from the left to the right creating a list of values. This list of values is assigned to the "*multi-var-name*" described in the **var** attribute.

This is illustrated in the example below:

```
<set-multi var="A" value="1,2"/>
<set var="B" value="3"/>
<set-multi var="C" value="?"@A?, ?@B?"/>
```

```
<set-multi var="D" value="A, B"/>
```

For `<set-multi>` variable C, the multi-value variable A is first evaluated, which yields a partial list of values: 1,2. Next, the value of the single-value variable B is evaluated and appended to the above partial list of values, yielding the final list of values 1, 2 and 3. Multi-value variable C will receive values 1,2,3.

Multi-value variable D will receive values A,B.

If there is only one variable *value* specified in the **value** attribute, the value(s) of that variable will be assigned to the variable *multi-var-name*. In the example below, variable C will receive values 1, 2.

```
<set-multi var="C" value="?@A?"/>
```

If the same variable appears more than once in the **value** attribute, each occurrence of that variable will contribute the resulting list of values. For example,

```
<set-multi var="C" value="??@A?, ?@A?, ?@A?"/>
```

variable "C" will receive values 1,2,1,2,1,2.

The scope and overriding rules of multi-value variables is the same as for single-value variables.

The `<set-multi>` command is an empty tag and cannot contain any content between its start tag and end tag.

Escape character

In the **value** attribute, comma plays the role of a separator. To avoid interpreting comma as a separator, an escape character `\` should be used before comma: `\"`. For example, after command:

```
<set-multi var="A" value="1\",2"/>
```

the value of A is one element "1,2".

12.3 Attribute definition:

var = "*multi-var-name*"

This attribute specifies the name of multi-value variable. The *multi-var-name* must be an Expression that yields a legal variable name. A legal name of a variable is a mixture of any characters but "?", "@" and ",".

defer-evaluation = "*yes-no*"

Deferred evaluation for multi-value variables works in the same way as deferred evaluation of single-valued variables (see Section 0).

In the following example, Expressions `?@a?`, `?@b?`, `(?@c?+2)` will be entered in to the Symbol Table without evaluating them and evaluation will takes place each time variable A is referenced. (The details of evaluation of `?@` variable? is explained in Section 9)

```
<set-multi var="A" value="?@b?,?@b?, (?@c? + 2)" defer-evaluation= "yes"/>
```

Note: The “defer-evaluation” has effect only on the **value** attribute. If variables appear in any other attributes of <set-multi> command, the evaluation will take place immediately.

13 Additional rules for <set> and <set-multi> commands

The single-value variable defined by <set> and multi-value variable defined by <set-multi> commands must not be the same. The processor will report an error otherwise.

When an x-frame is <adapt>ed with **samelevel** = “yes”, all the <set> and <set-multi> commands defined in that x-frame behave as if they were originally written in their parent x-frame in place of <adapt> command. This is illustrated in the following example. This example uses <set> command for illustration but the same rule applies for <set-multi> command as well.

x-frame A:

```

<set var="x" value="XA1"/>
    value of variable x is <value-of expr="?@x?"/> // variable x value is XA1
<adapt x-frame="B.xvcl" samelevel="yes"/> // variables x and y in x-frame B is
                                         // raised here and behave as if they
                                         // are written here. So the value of
                                         //x is reset to XB1.
value of variable x is reset to <value-of expr="?@x?"/> // x's value is reset to XB1
value of variable y is <value-of expr="?@y?"/> // variable y's value comes from
                                         //x-frame B

<adapt x-frame="C.xvcl"/>
```

x-frame B:

```

<set var="y" value="YB1"/>
<set var="x" value="XB1"/>
```

x-frame C :

```

    Because of same level, Now x-frame C has:
    value of variable x is <value-of expr="?@x?"/>
    value of variable y is <value-of expr="?@y?"/>
```

result:

```

    value of variable x is XA1
value of variable x is reset to XB1
value of variable y is YB1
```

Because of same level, Now x-frame C has:

value of variable x is XB1

value of variable y YB1

variable x and y are got from the samelevel effects from B.xvcl

Comments:

Notice that x-frame B is adapted with samelevel defined yes in x-frame A. This causes the variables x and y in x-frame B to be raised to the adapting x-frame A. These variables behave as if they are originally written in x-frame A. Thus, the original x value defined in A is reset to XB1. Note that before adapting x-frame B the value of variable x is XA1. After adapting x-frame B the value of x becomes XB1. When adapting x-frame C from x-frame A, variable x with the new value XB1 together with variable y with value XY1 will be passed down to x-frame C.

14 <select> command

14.1 Syntax

```

select ::= <select option= "control-variable">
    ( <option value= "value ( / value)*" [comp-operator="comp-operator ( ,
    comp-operator)*"]>
        option-body
    </option> )*
    [<otherwise>
        option-body
    </otherwise>]
</select>

control-variable ::= single-var-name

option-body ::= ( textual-content | break | adapt | set | set-multi | select |
                ifdef | ifndef | value-of | message | while | remove)*

value ::= Expression

comp-operator ::= Expression
    
```

14.2 Command description:

The <select> command selects zero or more from the listed option clauses. The XVCL processor processes each of the selected option clauses immediately upon selection.

Options are selected based on the value of control-variable specified in the **option** attribute. If control-variable is undefined (that is it does not exist in the Symbol Table), the processor issues an error message and terminates processing.

As indicated in the *Syntax* section, the `<select>` command can contain zero or more `<option>` commands followed by an optional `<otherwise>` command. We use term **option clause** to denote either `<option>` or `<otherwise>` commands.

The order of the appearance of the `<option>` and `<otherwise>` commands is strictly controlled. The XVCL processor will report an error if the order is not obeyed. The option clauses can only be written inside a `<select>` command.

The XVCL processor checks `<option>`s in sequential order and selects for processing `<option>`s as follows: the value of the control-variable of `<select>` is compared against the *values* specified in the **value** attribute of each `<option>` using the *comp-operators* specified in the **comp-operator** attribute. If the **comp-operator** attribute is omitted, the processor uses the equality operator (=) by default for matching all the *values* specified in the value attribute of `<option>` against the value of the control-variable.

If none of `<option>`s is selected, then `<otherwise>` command is selected, if present.

14.3 Attribute definition for `<select>` command:

`<option= "control-variable">`

This attribute specifies a single-value variable name. The *control-variable* must be an Expression that yields a valid single-value variable name whose value will be used by `<option>`s in selecting `<option>`s for processing.

14.4 The attributes of the `<option>` command:

`<option value= "value (/ value)*" [comp-operator="comp-operator (, comp-operator)*"]>`

Each *value* and *comp-operator* must be an Expression. The *comp-operator* Expression must yield one of the following comparison operators: `<`, `>`, `!=`, `=`, `<=` and `>=`.

The *values* in **value** attribute are separated by the OR symbol “|” and their corresponding *comp-operators* in attribute **comp-operator** are separated by commas. If the **comp-operator** is omitted the processor uses the equality operator (=) by default for all the *values*. If the **comp-operator** is specified, the number of *values* in the value attribute and the number of *comp-operators* in the **comp-operator** attribute must be the same.

The following is the general format of the `<option>` command:

`<option value="value1|value2|...|valuen" comp-operator="comp-operator1,comp-operator2,... ,comp-operatorn">`

To determine whether a given `<option>` should be selected or not, the value of the control-variable is compared against *i*'th *value* using the *i*'th *comp-operator*. If any of these comparisons is TRUE, the `<option>` is selected. If all the comparisons are FALSE, the `<option>` is not selected.

The following example illustrates selection of options:

```
<set var = "x" value = "a" />
<select option = "x">
```

<option value="a | b" comp-operator="=,="> // this option is selected if variable x has the

// value "a" OR "b"

</option>

....

</select>

The comparison is made based on the type of the *value*. String comparison is made for alphanumeric and alphabetic values, and numeric comparison is made for numeric values. In order to make a numeric comparison, the value of both the control-variable and *value* must be numeric. Otherwise, string comparison will be made.

For example, if the option value is "1" and control-variable value is "2" then the numeric comparison is made. If the option value is "A1" and control-variable value is "2" then the string comparison is made.

15 <while> command

15.1 Syntax

<while using-items-in = "multi-var-name (, multi-var-name)* ">

while-body

</while>

while-body := (textual-content | break | adapt | set | set-multi | select |
ifdef | ifndef | value-of | message | while | remove)*

15.2 Command description:

The <while> command iterates over its body. All the multi-value variables listed in **using-items-in** attribute must have the same number of values. Each iteration uses the *i*th value of each of the multi-value variables listed in the **using-items-in** attribute. Starting with 1, <while> implicitly increments the value of the index *i* by 1 in each iteration and terminates after processing the last values of variables. It follows that the number of iterations is equal to the number of values in each of the multi-value variables listed in the attribute **using-items-in**.

Values of the multi-value variables listed in the attribute **using-items-in** can be referenced in the body of a <while> command. If used inside <while> command, these multi-value variables behave like a single-value variables and can be referenced just like a single-value variable.

This is illustrated in the following example.

X-frame A:

```
<set-multi var="xxx" value="1,2,3"/>
```

```
<adapt x-frame="B.xvcl"/>
```

X-frame B:

```
<while using-items-in="xxx">
  value of xxx is <value-of expr="?@xxx?"/>
</while>
```

results:

```
value of xxx is 1
value of xxx is 2
value of xxx is 3
```

Comments:

The multi-value variable xxx has the values 1,2,3. The <while> command in x-frame B uses the multi-value variable xxx. The <while> command iterates over its body for each value in the multi-value variable xxx. In the first iteration the value of xxx is 1. The value of xxx becomes 2 and 3 in the subsequent iterations. It follows that the number of iterations is equal to the number of values in a multi-value variable.

The following example illustrates the use of one or more multi-value variables in while loop.

```
<set-multi var="x" value="1,3"/>
<set-multi var="y" value="2,3"/>
<while using-items-in="x, y">
  <select option="x">
    <option value="?@y?">
      The value of x is <value-of var="?@x?"/>
      and the value of y is "<value-of var="?@y?"/>
    </option>...
  </select>...
</while>
```

In the above example, during the first iteration the value of x is 1 and the value of y is 2. In the first iteration, the <option> command checks if value of variable x equals the value of variable y, therefore this condition is not satisfied. In the second iteration, both variables have the same value of 3 and the condition is met. As a result, the text:

```
The value of x is 3
and the value of y is 3
```

is emitted.

<while> command is often used for code generation. For example, generating code that creates database tables, creating user interface buttons and menus, etc.

Dynamic invocation of multi-value variables is also possible in <while> commands, as illustrated in example below:

```

<set-multi var="index" value="1,2"/>
<set-multi var="multi1" value="a,b,c"/>
<set-multi var="multi2" value="d,e"/>...
<while using-items-in="index">
  <while using-items-in="multi?@index?">
    ...
  </while>
</while>

```

The variable `index` is used to create the names of multi-value variables: `multi1` and `multi2`. For each iteration of the outer `<while>`, Expression `multi?@index?` will generate the new multi-value variable name in the inner `<while>` loop.

This is further illustrated in the following complete example.

X-frame A:

```

<set-multi var="multi1" value="A,B"/>
<set-multi var="multi2" value="C,D"/>
<set-multi var="index" value="1,2"/>
<adapt x-frame="B.xvcl"/>

```

X-frame B:

```

<while using-items-in="index">
  <while using-items-in="multi?@index?"> // in the first iteration VA1 is used
                                     // in the second iteration VA2 is used
    <value-of expr="multi?@index?" />
  </while>
</while>

```

result:

```

A }
B }   from multi1
C }
D }

```

Comments:

In the first iteration the value of `"index"` is 1 and `multi1` is used in the inner loop as a result of evaluating `multi?@index?`. As a result the A and B, is emitted in the inner loop iterations.

Similarly in the second loop `multi2` is used in the inner loop and as a result C and D are emitted in the inner loop iteration. In that way the name of the multi-value variable in inner `<while>` loop keeps changing in each iteration of the outer loop.

If there is an `<adapt>` command inside `<while>`, these multi-value variables (currently act as single-value variable) with corresponding *i*'th values are passed down to that x-frame and all its descendent x-frames. The variables that are passed down in this way have the same privilege as the single-value variables, i.e., they can be referred to in other XVCL commands. The following example illustrates this.

```
<set-multi var="x" value="1,2,3"/>
<while using-items-in="x">
    <adapt x-frame="B.xvcl"/> // for each iteration x with new value is passed
                               //down to x-frame B
</while>
<!-- <value-of var="?@x?"> --> // if this is not commented the processor will
                               //produce error. x cannot be referenced here
```

X-frame B:

```
Value of multi-value variable x in x-frame B is <value-of expr="?@x?">
```

result:

Value of multi-value variable x in x-frame B is 1

Value of multi-value variable x in x-frame B is 2

Value of multi-value variable x in x-frame B is 3

Comments:

Notice the `<adapt>` command inside `<while>` in x-frame A.

The `<while>` in x-frame A passes the single-value behavior of variable *x* with different values in each iteration. Referring to the example, in the first iteration, the single-value variable with value 1 is passed down to x-frame B. Similarly, single-value variable with value 2 and 3 are passed down to x-frame B in the second and third iteration of the `<while>` loop.

It is important to note the commented `<value-of>` command in the x-frame A outside of `<while>` loop. If `<value-of>` command is not commented, the processor will produce the variable undefined error. This is because multi-value variables cannot be referenced like single-value variable, outside of the `<while>` and the scope explained above.

When `<while>` commands are nested, as shown in the following example, both multi-value variables *x* and *y* can be referenced as single-value variables inside the inner `<while>` loop.

```
<while using-items-in="x">
    <while using-items-in="y">
        // both "x" and "y" can be referenced as single-value variables here
    </while>
</while>
```

15.3 Attribute definition:

using-items-in = "*multi-var-name* (, *multi-var-name*)*"

This attribute specifies one or more multi-value variables to be used by <while> command. The *multi-var-name* must be an Expression that yields the name of a multi-value variable existing in the Symbol Table. Otherwise, a warning will be produced and the <while> will be skipped.

16 Definition of <ifdef> command

16.1 Syntax:

```
ifdef  :=  <ifdef var="var-name">
          if-body
        </ifdef>
var-name      :=  Expression
if-body       :=  ( textual-content | break | adapt | set | set-multi | select |
                  ifdef | ifndef | value-of | message | while | remove )*
```

16.2 Command description:

If variable *var-name* is defined (that is, it exists in the Symbol Table), the XVCL processor processes the if-body. Otherwise, the if-body is not processed.

16.3 Attribute definition:

var="var-name"

Expression must yield a legal name of a variable (either single-value or multi-value) that may or may not exist in the Symbol Table.

17 Definition of <ifndef> command

17.1 Syntax:

```
ifndef :=  <ifndef var="var-name">
          if-body
        </ifndef>
```

17.2 Command description:

If variable *var-name* is undefined (that is, it does not exist in the Symbol Table), the XVCL processor processes the if-body. Otherwise, the if-body is not processed.

17.3 Attribute definition:

var="var-name"

Expression must yield a legal name of a variable (either single-value or multi-value) that may or may not exist in the Symbol Table.

18 Definition of <remove> command

18.1 Syntax:

```
<remove var="var-name"/>
```

18.2 Command description:

This command un-defines a variable "*var-name*" described in **var** attribute by removing it from the Symbol Table. The "*var-name*" should have been previously defined by <set> or <set-multi> command. Otherwise, the command is ignored and a warning message is produced. The variable is removed if and only if the <remove> command appears in the x-frame that originally defines that variable (i.e., the x-frame that entered that variable into the Symbol Table). Otherwise, the command is ignored and a warning message is produced.

The following example illustrates this.

X-frame A:

```
<set var= "x" value="XA"/>
<adapt x-frame="B.xvcl"/>
```

X-frame B:

```
<remove var="x"/> // warning! variable x cannot be remove
```

result:

warning will be produced when processing <remove> command in x-frame B.

comments:

The warning is generated because the descendent x-frames are not allowed to <remove> any variables defined by ancestor x-frames.

18.3 Attribute definition:

var = "var-name"

This attribute specifies a variable name to be removed from the Symbol Table. It can be the name of a single-value or a multi-value variable. The *var-name* must be an Expression that yields the name of the variable to be removed.

19 Definition of <message> command

19.1 Syntax

<message text=" message " [continue = " yes-no "] />

19.2 Command description:

When the XVCL processor encounters the <message> command, it will display a *message* on the screen. This command does not affect the output emitted by the XVCL processor. The <message> command can be used for debugging purpose like checking variable values and trapping error situations. The typical usage of this command is in the <select> command. However, it is also useful for tracking variable values by displaying them to the screen during processing.

19.3 Attributes definition:

message = "text"

This attribute specifies a message to be displayed. The *message* must be an Expression.

continue = "yes-no"

This attribute specifies if the processing should continue (value "yes") or not (value "no"). The *yes-no* must be an Expression that yields value "yes" or "no". If this attribute is omitted, the default value is "yes".

20 Processor options and configuration file

The following extra features can be activated via processor options and configuration file:

-T option and variable optionT

When the processor is invoked with -T option, for example:

```
java -jar xvcl.jar -T SPC
```

the processor will include the comment line with the name of an x-frame as the first line in the output emitted from that x-frame.

The user should specify comment symbols in 'begin-comment' and 'end-comment' variables. The default for begin-comment is //. The default for end-comment is nil (nothing).

Currently, we do not provide trace information for <insert> commands.

-B option and variable optionB

When the processor is invoked with -B option, all the white spaces (blank, space, etc.) in the Textual Content around an XVCL command (any command with the exception of <value-of>) are trimmed. XVCL white space characters are the same as in Java. A character is an XVCL white space character if and only if it satisfies one of the following criteria:

- It is a Unicode space character (SPACE_SEPARATOR, LINE_SEPARATOR, or PARAGRAPH_SEPARATOR) but is not also a non-breaking space ('\u00A0', '\u2007', '\u202F').
- It is '\u0009', HORIZONTAL TABULATION.
- It is '\u000A', LINE FEED.
- It is '\u000B', VERTICAL TABULATION.
- It is '\u000C', FORM FEED.
- It is '\u000D', CARRIAGE RETURN.
- It is '\u001C', FILE SEPARATOR.
- It is '\u001D', GROUP SEPARATOR.
- It is '\u001E', RECORD SEPARATOR.
- It is '\u001F', UNIT SEPARATOR.

Option -B does not remove white spaces (and any Textual Contents) inside CDATA tags. Therefore, if you want to keep blank lines after or before a command, you can put them into CDATA tags.

-N option and variable optionN (version 2.07 only)

Option -N was used to instruct the processor to recognize a namespace "xvcl". The processor 2.10 recognizes regular XVCL commands (e.g., <adapt> as well as in namespace notation (e.g., <xvcl:adapt>) without specifying -N. Therefore, option -N has been removed.

-V option

Option -V runs the processor in validation mode, i.e., to process an x-framework without emitting the output.

-L option

Option -L instructs the processor to log the names of the files generated during processing into a file named [SPC].log placed in the same directory as SPC.

Document Type Definition (DTD) for XVCL

```

<?xml version="1.0" encoding="us-ascii"?>

<!-- this entity is used for passing comas (,) as value in multi-valued -->
<!ENTITY comma "\", ">
<!ENTITY newline "\n">
<!ENTITY return "\r">
<!ENTITY tab "\t">

<!ELEMENT x-frame (#PCDATA | break | adapt | set | set-multi | select | value-of |
message | while | remove | ifdef | ifndef | print)*>
<!ELEMENT break (#PCDATA | adapt | set | set-multi | select | value-of | message |
while | remove | ifdef | ifndef | print)*>
<!ELEMENT adapt (insert | insert-after | insert-before)*>
<!ELEMENT insert (#PCDATA | break | adapt | set | set-multi | select | value-of |
message | while | remove | ifdef | ifndef | print)*>
<!ELEMENT insert-before (#PCDATA | break | adapt | set | set-multi | select | value-of
| message | while | remove | ifdef | ifndef | print)*>
<!ELEMENT insert-after (#PCDATA | break | adapt | set | set-multi | select | value-of |
message | while | remove | ifdef | ifndef | print)*>
<!ELEMENT set-multi EMPTY>
<!ELEMENT set EMPTY>
<!ELEMENT select (option*, otherwise?)>
<!ELEMENT ifndef (#PCDATA | break | adapt | set | set-multi | select | value-of |
message | while | remove | ifdef | ifndef | print)*>
<!ELEMENT ifdef (#PCDATA | break | adapt | set | set-multi | select | value-of |
message | while | remove | ifdef | ifndef | print)*>
<!ELEMENT option (#PCDATA | break | adapt | set | set-multi | select | value-of |
message | while | remove | ifdef | ifndef | print)*>
<!ELEMENT otherwise (#PCDATA | break | adapt | set | set-multi | select | value-of |
message | while | remove | ifdef | ifndef | print)*>
<!ELEMENT while (#PCDATA | break | adapt | set | set-multi | select | value-of |
message | while | remove | ifdef | ifndef | print)*>
<!ELEMENT message EMPTY>
<!ELEMENT value-of EMPTY>
<!ELEMENT remove EMPTY>
<!ELEMENT print EMPTY>

<!ATTLIST x-frame
    name CDATA #REQUIRED
    outdir CDATA #IMPLIED
    outfile CDATA #IMPLIED
    language CDATA #IMPLIED

<!ATTLIST break
    name CDATA #REQUIRED>
<!ATTLIST adapt
    x-frame CDATA #REQUIRED

```

```
    outdir CDATA #IMPLIED
    outfile CDATA #IMPLIED
    samelevel CDATA #IMPLIED
    once CDATA #IMPLIED>
<!ATTLIST set
    var CDATA #REQUIRED
    value CDATA #REQUIRED
    defer-evaluation CDATA #IMPLIED>
<!ATTLIST set-multi
    var CDATA #REQUIRED
    value CDATA #REQUIRED
    defer-evaluation CDATA #IMPLIED>
<!ATTLIST insert
    break CDATA #REQUIRED>
<!ATTLIST insert-after
    break CDATA #REQUIRED>
<!ATTLIST insert-before
    break CDATA #REQUIRED>
<!ATTLIST select
    option CDATA #REQUIRED>
<!ATTLIST option
    value CDATA #REQUIRED
    comp-operator CDATA #IMPLIED>
<!ATTLIST while
    using-items-in CDATA #REQUIRED>
<!ATTLIST value-of
    expr CDATA #REQUIRED>
<!ATTLIST message
    text CDATA #REQUIRED
    continue CDATA #IMPLIED>
<!ATTLIST remove
    var CDATA #REQUIRED>
<!ATTLIST ifdef
    var CDATA #REQUIRED>
<!ATTLIST ifndef
    var CDATA #REQUIRED>
<!ATTLIST print
    text CDATA #REQUIRED>
```