*2022-10-09 / CLFEY - Copied from https://epeios.q37.info/tools/xppq/*

# *XPPq*, the XML preprocessor

## Quick access

- *GitHub* version of the command-line tool: https://q37.info/github/xppq-cli,
- online demonstration:
  - of the *Java* package: http://q37.info/s/rppxgdf4,
  - of the *Node.js* module: http://q37.info/s/7zwtt3h4.

## Description

*XPPq* is a command-line tool which transforms an *XML* file to another *XML* file, following directives inserted directly in the source *XML* file. This directives allow to handle macros, to affect value to variables and to test their values, to include files… In a glance, XPPq aims to be to *XML* what *CPP* is to C/C++.

To achieve this, following directives are available:

- `<xpp:expand href="file"/>` which allows the inclusion of a file,
- `<xpp:define name="name">…</xpp:define>`, which, in combination with `<xpp:expand select="name"/>` allows the definition and expansion of a macro,
- `<xpp:set name="name" value="value"/>`, which, in combination with `<xpp:ifeq select="name" value="value">…</xpp:ifeq>` allows the conditional inclusion of an XML tree,
- …

More about this directives and the others can be found in the directives dedicated section.

The preprocessor is also embedded in all the *Epeios* software which deals with *XML* files. So the *xpp* directives can also be used in those files.

The *XML* namespace is: `http://epeios.q37.info/ns/xpp` (`xmlns:xpp="http://epeios.q37.info/ns/xpp"`).

*XPPq* handles 8-bits encoded files, and also *UTF-8* encoded files, with or without *BOM*. All included files (using the *expand* directive) have to use the same encoding as the file which includes them.

*XPPq* is available as:

- command-line tool,

- *Java* package,
- *Node.js* module.

# Example

File `Common.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<xpp:bloc xmlns:xpp="http://epeios.q37.info/ns/xpp">
  <xpp:define name="LinuxURL">
    <xpp:bloc>linux.org</xpp:bloc>
  </xpp:define>
  <xpp:define name="WindowsURL">
    <xpp:bloc>windows.com</xpp:bloc>
  </xpp:define>
  <xpp:define name="Linux">
    <xpp:bloc>
      <xpp:define name="Directory">
        <xpp:bloc>/home/dupond/</xpp:bloc>
      </xpp:define>
      <xpp:define name="RootURL">
        <xpp:expand select="LinuxURL"/>
      </xpp:define>
    </xpp:bloc>
  </xpp:define>
  <xpp:define name="Windows">
    <xpp:bloc>
      <xpp:define name="Directory">
        <xpp:bloc>c:\Documents\Dupond\</xpp:bloc>
      </xpp:define>
      <xpp:define name="RootURL">
        <xpp:expand select="WindowsURL"/>
      </xpp:define>
    </xpp:bloc>
  </xpp:define>
  <xpp:define name="File">
    <xpp:bloc>
      <xpp:expand select="Directory"/>
      <xpp:expand select="FileName"/>
    </xpp:bloc>
  </xpp:define>
  <xpp:ifeq select="OS" value="Linux">
    <xpp:expand select="Linux"/>
  </xpp:ifeq>
  <xpp:ifeq select="OS" value="Windows">
    <xpp:expand select="Windows"/>
  </xpp:ifeq>
  <SomeFile>
    <xpp:define name="FileName">
      <xpp:bloc>SomeFile</xpp:bloc>
    </xpp:define>
    <xpp:expand select="File"/>
  </SomeFile>
  <OtherFile>
    <xpp:define name="FileName">
      <xpp:bloc>OtherFile</xpp:bloc>
```

```
    </xpp:define>
    <xpp:expand select="File"/>
  </OtherFile>
  <SomeURL>
    <xpp:bloc>http://</xpp:bloc>
    <xpp:expand select="RootURL"/>
    <xpp:bloc>/something</xpp:bloc>
  </SomeURL>
</xpp:bloc>
```

File `Linux.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration xmlns:xpp="http://epeios.q37.info/ns/xpp">
  <xpp:set name="OS" value="Linux"/>
  <xpp:expand href="common.xml"/>
</Configuration>
```

File `Windows.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration xmlns:xpp="http://epeios.q37.info/ns/xpp">
  <xpp:set name="OS" value="Windows"/>
  <xpp:expand href="common.xml"/>
</Configuration>
```

`xppq Linux.xml` outputs following:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration>
       <SomeFile>/home/dupond/SomeFile</SomeFile>
       <OtherFile>/home/dupond/OtherFile</OtherFile>
       <SomeURL>http://linux.org/something</SomeURL>
</Configuration>
```

`xppq Windows.xml` outputs following:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration>
       <SomeFile>c:\Documents\Dupond\SomeFile</SomeFile>
       <OtherFile>c:\Documents\Dupond\OtherFile</OtherFile>
       <SomeURL>http://windows.com/something</SomeURL>
</Configuration>
```

# The directives

`xpp` is used as the namespace by default, but you can change it (see `xppq --help`).

## Summary

- `<xpp:define name="NAME">...</xpp:define>`: definition of a macro named **NAME**.

- `<xpp:expand select="NAME">...</xpp:expand>`: expansion of the macro named **NAME**.
- `<UserTag xpp:attribute="ATTRIBUTE_NAME,MACRO_NAME">...</UserTag>`: expansion of the macro named **MACRO_NAME** as value of an attribute named **ATTRIBUTE_NAME**.
- `<xpp:expand href="FILENAME"/>`: inclusion of the content of the file named **FILENAME**.
- `<xpp:set name="NAME" value="VALUE"/>`: set the variable named **NAME** to the value **VALUE**.
- `<xpp:ifeq select="NAME" value="VALUE">...</xpp:ifeq>`: the content of this directive is skipped unless variable named **NAME** has **VALUE** as value.
- `<xpp:cdata>...</xpp:cdata>`: the content of this directive is put in a *CDATA* section.
- `<xpp:bloc>...</xpp:bloc>`: without attribute, this directive has no effect by itself, but is required under some circumstances by above directives.

## The `define` directive

### Description

This directive allows to define a macro. The body of a macro must contain one and only one root tag (use `bloc` directive if needed) ; it can not be empty (nor `<xpp:define name="…"></xpp:define>` nor `<xpp:define name="…"/>`). The body of a macro, when enclosed in a `bloc` directive, can contain several child tags.

The content of a macro is expanded using the `expand` directive with `select` attribute.

### Syntax

#### Definition

```
<xpp:define name="NAME">
Body
</xpp:define>
```

Defines a macro named **NAME** with content **Body**.

#### Expansion

```
<xpp:expand select="NAME"/>
```

Expands the content of the macro named **NAME** (the entire directive is replaced by **Body** as defined above).

## The `expand` directive (with `select` attribute)

### Description

The `expand` directive with `select` attribute allows to expand a macro previously defined by a `define` directive.

### Syntax

#### Definition

```
<xpp:define name="NAME">
Body
</xpp:define>
```

Defines a macro named **NAME** with content **Body**.

#### Expansion

```
<xpp:expand select="NAME"/>
```

Expands the content of the macro named **NAME** (the entire directive is remplaced by **Body** as defined above).

## The `expand` directive (with `href` attribute)

### Description

The `expand` directive with `href` attribute allows to include a file. The included file must contain one and only one root tag (use `bloc` directive if needed). If the file location is relative, the base directory is the one of the file which contains the given `expand` directive.

### Syntax

```
<xpp:expand href="FILENAME"/>
```

The whole `expand` directive is replaced by the content of the file named **FILENAME**.

## The `attribute` directive

### Description

The `attribute` directive allows to create an attribute with the content of a macro previously defined by a `define` directive as value. Unlike the other directives, which are all used as a tag, the `attribute` directive must be used as an attribute.

The body of the macro must expand to a value, as it will be used as an attribute value. It can contain other `xpp` directives, but once there are all handled, the result must be a value, with no tag.

NOTA : this feature is experimental, but seems so long reliable.

### Syntax

#### Definition

```
<xpp:define name="MACRO_NAME">VALUE</xpp:define>
```

Defines a macro named **MACRO_NAME** with content **VALUE**.

#### Expansion

```
<UserTag xpp:attribute="ATTRIBUTE_NAME,MACRO_NAME"/>
```

Creates an attribute named **ATTRIBUTE_NAME** with the content of the macro named **MACRO_NAME** as value.

## The `set` directive

### Description

The `set` directive allows to affect a value to a variable. The value of a variable can be tested with the `ifeq` directive.

### Syntax

```
<xpp:set name="VARIABLE" value="VALUE"/>
```

Sets the variables named **NAME** with the value **VALUE**.

## The `ifeq` directive

### Description

The `ifeq` directive allows to test the value of a variables defined by the `set` directive, and, depending of the result of this test, to skip or not the content of the given `ifeq` directive.

### Syntax

```
<xpp:ifeq select="VARIABLE" value="VALUE">
BODY
</xpp:ifeq>
```

The variable named **VARIABLE** is tested. If **VARIABLE** was set to **VALUE** by a `set` directive, then **BODY** is handled, otherwise **BODY** is skipped. Even if **BODY** is skipped, it must be a valid `xml` tree (i.e. must contain one and only one root tag ; use `bloc` directive if needed).

## The `cdata` directive

### Description

The content of this directive is put into a *CDATA* section, i.e the content is put between the `<![CDATA[…]]>` delimiters. But, unlike what happens with the usual *CDATA* section, an error is issued if this content is not a well-formed XML tree.

### Syntax

```
<xpp:cdata>content</xpp:cdata>
```

## The `bloc` directive

### Global

#### Description

The content of the `define` and `ifeq` directives, and also the content of a file included by the `expand` directive (with `href` attribute), must be a valid XML tree, *i.e.* it must contain one, and only one, root tag (with possibly child tags).

In the resulting file, opening and closing `bloc` directives do not appear.

#### Content with only a value

Case where a content is only a value (without enclosing tag) :

```
<xpp:bloc>/home/dupont/</xpp:bloc>
```

#### Content with more then one root tags

Case where the content is constituted by more then one root tags :

```
<xpp:bloc>
    <Name>
        <Last>Doe</Last>
        <First>John</First>
    </Name>
    <EMail>doe@site.com</EMail>
</xpp:bloc>
```

### The `preserve` attribute

The `preserve` attribute can only take 2 values : `yes` or `no`.

By default, the entire XML tree is considered as surrounded by a `bloc` with a `preserve` attribute of value `no`.

This attribute is ignored when `xppq` is launched without the `--preserve` flag.

When the preprocessor encounters a `bloc` directive with a `preserve` attribute of value `yes`, then all the enclosed preprocessor directives are ignored, and the enclosed XML tree is outputted as is, with the preprocessor directives. Even an enclosed `bloc` directive, with or without a `preserve` attribute to `no`, and the enclosed XML tree, is outputted as is (i.e. doesn't cancel a surrounding `bloc` directive with a `preserve` value of `yes`).

### The `marker` attribute

A value of a `marker` attribute can only be empty, or containing one character.

When the preprocessor encounters a `bloc` directive with a non-empty `marker` attribute, variable name surrounded by the character defined in the `marker` attribute are replaced by the value of this variable in all the attribute values of all the enclosed tags, excluding attributes of preprocessor directives. The variable must exist. As an exception, this occurs also for the `href` and `select` attribute of the `expand` directive, and the `value` attribute of the `set` directive. A double instance of the character defined in the `marker` attribute is replaced by a single instance of this character.

To cancel such substitutions, enclose the concerned XML with a `bloc` directive with an empty `marker` attribute. By default, the root XML tree is considered enclosed by a `bloc` tag with an empty `marker` attribute, i.e., by default, no substitutions are made. A `bloc` directive with or without a `marker` attribute, empty or not, can contain other `bloc` directive with or without a `marker` attribute, empty or not…