M.Dobrovnik
H.Groiss

XWDL

An extensible formulation of WDL in XML

2.Dezember 2002

Dokumentversion 1.0

Groiss Informatics GmbH

Contents

1 Overview **3**

2 Usage **4**
2.1 HTML-Client 4
2.2 API 4

3 The basic DTD **6**
3.1 An Example 9
3.1.1 WDL 10
3.1.2 XDWL 11

4 The extension model **15**
4.1 The extension DTD 15
4.2 An Example 17

5 Extension API **21**

1 Overview

This document presents the XWDL, an extensible XML based dialect of WDL.
The classic approach to define process types in @enterprise was to use the Workflow Description Language (WDL) or to draw the process with the process editor applet.
WDL is designed as a kind of structured, human-readable process programming language. It is not mainly targeted for the exchange of process type information with other systems. In order to semantically analyze the WDL-scripts, those third-party systems would have to make use of conventional parsing techniques.
The export/import format of @enterprise allows one to transfer application definitions (which contain process definitions) between @enterprise systems. While this format is XML based, the process information is still sent along as a WDL-Script.
The formulation of WDL in a structure-rich XML has the following aims / benefits:

- third-party applications can generate XWDL-Scripts on the grounds of a well understood formalism
- use a plain DTD-driven XML editor to write XWDL-Scripts with automatic syntactical correctness
- verification of the syntax using solely an out of the box XMl-parser.
- third party extensions could be accomodated using an extension approach for the DTD
- a new version of the process editor could use this structure to communicate with the server [1]

[1] appropriate descriptions of tasks, users, roles, departments, ... would be needed

2 Usage

2.1 HTML-Client

XWDL-Processes can be loaded into the system exactly like WDL Processes. There are two new links on the Process / Script page for viewing (IE6 needed) or downloading the XWDL-Code of a process.

2.2 API

A simple API is provided to insert XWDL-Processes into the system.

```
package com.groiss.wf.xwdl;
 public class ProcessParser implements IProcessParser{

  public ProcessDefinition loadProcess(InputStream is, boolean genRoles, boolean genTasks)
   throws Exception;

  public ProcessDefinition loadProcess(String fileName, boolean genRoles, boolean genTasks)
   throws Exception;

   public String getErrors();
```

A XWDL-Process can be loaded from an InputStream or from a File which is specified via its filename. The booleans genRoles and genTasks state whether roles and tasks should be generated. When the process could be loaded without errors, no Exception is thrown and the getErrors method will return the empty string.
A typical usage would be like this:

```
ProcessParser pp = new com.groiss.wf.xwdl.ProcessParser;
try {
ProcessDefinition pd  = pp.loadProcess(fileName, true, true);
} catch (Exception ex) {
//rollback;
}
```

```
if (pp.getErrors().length() != 0) {
// error occured;
// rollback;
} else {
// commit;
}
```

3 The basic DTD

The dtd uses ENTITY definitions for the content of each element. This allows for extensions of the DTD in a modular manner. The extension mechanism is described in the next section.

```
<!ENTITY % xwdl.process.xmlns.extra "">
<!ENTITY % xwdl.stmtlist.content
      "(label?, (if | while | loop | repeat | exit | goto
                | andpar | orpar | activity | system
                | call | choice | branch ))*">
<!ENTITY % xwdl.process.content      "(params?, forms?, timing?, %xwdl.stmtlist.content;)">
<!ENTITY % xwdl.params.content        "(formdecl*)">
<!ENTITY % xwdl.forms.content         "(formdecl*)">
<!ENTITY % xwdl.form.content          "EMPTY">
<!ENTITY % xwdl.timing.content        "(activity?)">
<!ENTITY % xwdl.formdecl.content      "EMPTY">
<!ENTITY % xwdl.label.content         "EMPTY">
<!ENTITY % xwdl.if.content           "(then, elsif*,  else?)">
<!ENTITY % xwdl.then.content          "%xwdl.stmtlist.content;">
<!ENTITY % xwdl.else.content          "%xwdl.stmtlist.content;">
<!ENTITY % xwdl.elsif.content         "(then)">
<!ENTITY % xwdl.while.content         "%xwdl.stmtlist.content;">
<!ENTITY % xwdl.loop.content          "%xwdl.stmtlist.content;">
<!ENTITY % xwdl.repeat.content        "%xwdl.stmtlist.content;">
<!ENTITY % xwdl.exit.content          "EMPTY">
<!ENTITY % xwdl.goto.content          "EMPTY">
<!ENTITY % xwdl.andpar.content        "(parallel+)">
<!ENTITY % xwdl.orpar.content         "(parallel+)">
<!ENTITY % xwdl.parallel.content      "%xwdl.stmtlist.content;">
<!ENTITY % xwdl.activity.content      "(agent*,form*)">
<!ENTITY % xwdl.system.content        "EMPTY">
<!ENTITY % xwdl.call.content          "(form*)">
<!ENTITY % xwdl.choice.content        "(case+)">
<!ENTITY % xwdl.case.content          "%xwdl.stmtlist.content;">
<!ENTITY % xwdl.branch.content        "%xwdl.stmtlist.content;">
<!ENTITY % xwdl.agent.content         "EMPTY">
```

```
<!ENTITY % xwdl.geo.atts "x CDATA #IMPLIED y CDATA #IMPLIED">
<!ENTITY % xwdl.geoend.atts "xend CDATA #IMPLIED yend CDATA #IMPLIED">
```

Each element is then defined on the basis of the previously declared entities.

```
<!ELEMENT process    %xwdl.process.content;>
<!ELEMENT params     %xwdl.params.content;>
<!ELEMENT forms      %xwdl.forms.content;>
<!ELEMENT timing     %xwdl.timing.content;>
<!ELEMENT form       %xwdl.forms.content;>
<!ELEMENT formdecl   %xwdl.forms.content;>
<!ELEMENT label      %xwdl.label.content;>
<!ELEMENT if         %xwdl.if.content;>
<!ELEMENT then       %xwdl.then.content;>
<!ELEMENT else       %xwdl.else.content;>
<!ELEMENT elsif      %xwdl.elsif.content;>
<!ELEMENT while      %xwdl.while.content;>
<!ELEMENT loop       %xwdl.loop.content;>
<!ELEMENT repeat     %xwdl.repeat.content;>
<!ELEMENT exit       %xwdl.exit.content;>
<!ELEMENT goto       %xwdl.goto.content;>
<!ELEMENT andpar     %xwdl.andpar.content;>
<!ELEMENT orpar      %xwdl.orpar.content;>
<!ELEMENT parallel   %xwdl.parallel.content;>
<!ELEMENT activity   %xwdl.activity.content;>
<!ELEMENT system     %xwdl.system.content;>
<!ELEMENT call       %xwdl.call.content;>
<!ELEMENT choice     %xwdl.choice.content;>
<!ELEMENT case       %xwdl.case.content;>
<!ELEMENT branch     %xwdl.branch.content;>
<!ELEMENT agent      %xwdl.agent.content;>
```

Then the rather straightforward attribute declarations follow.

```
<!ATTLIST process
   xmlns:xwdl CDATA #FIXED "http://www.groiss.com"
   %xwdl.process.xmlns.extra;
id CDATA #REQUIRED
name CDATA #IMPLIED
version CDATA #IMPLIED
subject CDATA #IMPLIED
application CDATA #IMPLIED
startfunction CDATA #IMPLIED
owner CDATA #IMPLIED
description CDATA #IMPLIED
%xwdl.geo.atts;
%xwdl.geoend.atts;
```

```
>
<!ATTLIST formdecl
   id CDATA #REQUIRED
typ CDATA #REQUIRED
baseform CDATA #IMPLIED
>
<!ATTLIST timing
maxtime CDATA #IMPLIED
maxtimeunit CDATA #IMPLIED
timeoutaction CDATA #IMPLIED
>

<!ATTLIST form
name CDATA #REQUIRED
>
<!ATTLIST label
id CDATA #REQUIRED
>
<!ATTLIST if
condition CDATA #REQUIRED
%xwdl.geo.atts;
%xwdl.geoend.atts;
>
<!ATTLIST elsif
condition CDATA #REQUIRED
%xwdl.geo.atts;
>
<!ATTLIST while
condition CDATA #REQUIRED
%xwdl.geo.atts;
>
<!ATTLIST loop
%xwdl.geo.atts;
>
<!ATTLIST repeat
condition CDATA #REQUIRED
%xwdl.geo.atts;
%xwdl.geoend.atts;
>
<!ATTLIST exit
condition CDATA #REQUIRED
%xwdl.geo.atts;
>
<!ATTLIST goto
label CDATA #REQUIRED
%xwdl.geo.atts;
>
```

```
<!ATTLIST andpar
%xwdl.geo.atts;
%xwdl.geoend.atts;
>
<!ATTLIST orpar
%xwdl.geo.atts;
%xwdl.geoend.atts;
>
<!ATTLIST branch
%xwdl.geo.atts;
%xwdl.geoend.atts;
>
<!ATTLIST activity
id CDATA #REQUIRED
name CDATA #IMPLIED
version CDATA #IMPLIED
skipable CDATA #IMPLIED
%xwdl.geo.atts;
>
<!ATTLIST system
methodcall CDATA #REQUIRED
%xwdl.geo.atts;
>
<!ATTLIST call
id CDATA #REQUIRED
name CDATA #IMPLIED
%xwdl.geo.atts;
>
<!ATTLIST choice
%xwdl.geo.atts;
%xwdl.geoend.atts;
>
<!ATTLIST case
name CDATA #REQUIRED
condition CDATA #IMPLIED
%xwdl.geo.atts;
>
<!ATTLIST agent
string CDATA #REQUIRED
>
```

3.1 An Example

We will now present a rendering of WDL in XWDL by means of an example.

3.1.1 WDL

The example in WDL is:

```
process all_things_x()
version 1;
name "all control structures";
description "Test the control structures";
maxtime 10 minutes;
forms form Jobform;
timeoutaction none;
timeouttask
all,r adm_task(form);
application default;
startfunction ;
begin
  <first>
  all start_task(form);
  loop
    choice
    "first choice: an if":
      if (form.type = "hw") then
        all hw_task(form);
      elsif (form.type = "sw") then
        form.recipient swx_task(form);
      elsif (form.type = "adm") then
        first:user adm_task(form);
      else
        first:user none_task(form);
      end;
    "second choice: a while":
      while (form.type = "hw") do
        form.recipient while_task1(form);
        <in_while>
        form.recipient while_task2(form);
        form.recipient while_task3(form);
      end;
    "third choice: a loop":
      loop
        form.recipient loop_task(form);
        exit when (form.type = "hw");
      end;
    "fourth choice: system steps":
      system com.groiss.demo.SystemSteps.emptyMethod();
      form.recipient between_task(form);
      system com.groiss.demo.SystemSteps.emptyMethod();
      system com.groiss.demo.SystemSteps.emptyMethod();
      form.recipient aftersys_task(form);
```

```
    "fifth choice: andpar":
      andpar
        form.recipient andpar1_task(form);
        |
        all andpar2_task(form);
        |
        form.recipient andpar3_task(form);
      end;
    "sixth choice: orpar":
      orpar
        form.recipient orpar1_task(form);
        |
        form.recipient orpar2_task(form);
        |
        form.recipient orpar3_task(form);
      end;
    "eight choice: subprocesses":
      call subflow1(form);
    "nineth choice: goto (into the while)":
      goto in_while;
    end;
    exit when (form.finished = 1);
  end;
end
```

3.1.2 XDWL

The corresponding formulation in XWDL would look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE process SYSTEM "./conf/xwdl.dtd">
<process id="all_things_x" version="1"
name="all control structures" description="Test the control structures"
application="default">
  <forms>
    <formdecl id="form" typ="Jobform" />
  </forms>
  <timing timeoutaction="none" maxtime="10" maxtimeunit="minutes">
    <activity id="adm_task">
      <agent string="all" />
      <agent string="r" />
      <form name="form" />
    </activity>
  </timing>
  <label id="first" />
```

```
<activity id="start_task">
  <agent string="all" />
  <form name="form" />
</activity>
<loop>
  <choice>
    <case name="first choice: an if">
      <if condition="(form.type = &quot;hw&quot;)">
        <then>
          <activity id="hw_task">
            <agent string="all" />
            <form name="form" />
          </activity>
        </then>
        <elsif condition="(form.type = &quot;sw&quot;)">
          <then>
            <activity id="swx_task">
              <agent string="form.recipient" />
              <form name="form" />
            </activity>
          </then>
        </elsif>
        <elsif condition="(form.type = &quot;adm&quot;)">
          <then>
            <activity id="adm_task">
              <agent string="first:user" />
              <form name="form" />
            </activity>
          </then>
        </elsif>
        <else>
          <activity id="none_task">
            <agent string="first:user" />
            <form name="form" />
          </activity>
        </else>
      </if>
    </case>
    <case name="second choice: a while">
      <while condition="(form.type = &quot;hw&quot;)">
        <activity id="while_task1">
          <agent string="form.recipient" />
          <form name="form" />
        </activity>
        <label id="in_while" />
        <activity id="while_task2">
          <agent string="form.recipient" />
```

```
      <form name="form" />
    </activity>
    <activity id="while_task3">
      <agent string="form.recipient" />
      <form name="form" />
    </activity>
  </while>
</case>
<case name="third choice: a loop">
  <loop>
    <activity id="loop_task">
      <agent string="form.recipient" />
      <form name="form" />
    </activity>
    <exit condition="(form.type = &amp;quot;hw&amp;quot;)" />
  </loop>
</case>
<case name="fourth choice: system steps">
  <system methodcall="com.groiss.demo.SystemSteps.emptyMethod()" />
  <activity id="between_task">
    <agent string="form.recipient" />
    <form name="form" />
  </activity>
  <system methodcall="com.groiss.demo.SystemSteps.emptyMethod()" />
  <system methodcall="com.groiss.demo.SystemSteps.emptyMethod()" />
  <activity id="aftersys_task">
    <agent string="form.recipient" />
    <form name="form" />
  </activity>
</case>
<case name="fifth choice: andpar">
  <andpar>
    <parallel>
      <activity id="andpar1_task">
        <agent string="form.recipient" />
        <form name="form" />
      </activity>
    </parallel>
    <parallel>
      <activity id="andpar2_task">
        <agent string="all" />
        <form name="form" />
      </activity>
    </parallel>
    <parallel>
      <activity id="andpar3_task">
        <agent string="form.recipient" />
```

```
            <form name="form" />
          </activity>
        </parallel>
      </andpar>
    </case>
    <case name="sixth choice: orpar">
      <orpar>
        <parallel>
          <activity id="orpar1_task">
            <agent string="form.recipient" />
            <form name="form" />
          </activity>
        </parallel>
        <parallel>
          <activity id="orpar2_task">
            <agent string="form.recipient" />
            <form name="form" />
          </activity>
        </parallel>
        <parallel>
          <activity id="orpar3_task">
            <agent string="form.recipient" />
            <form name="form" />
          </activity>
        </parallel>
      </orpar>
    </case>
    <case name="eight choice: subprocesses">
      <call id="subflow1">
        <form name="form" />
      </call>
    </case>
    <case name="nineth choice: goto (into the while)">
      <goto label="in_while" />
    </case>
  </choice>
  <exit condition="(form.finished = 1)" />
 </loop>
</process>
```

4 The extension model

4.1 The extension DTD

The extension mechanism follows the spirit of the formulation of Modular XHTML [1] without introducing any unneeded complexity.
The main idea is to leave the basic XWDL DTD untouched and to define a specific extension DTD which would include the original DTD like this:

```
<![ INCLUDE [
<!ENTITY % xwdl.mod SYSTEM "./xwdl.dtd">
%xwdl.mod;]]>
```

Before the inclusion, one would define a name for the extension like this:

```
<!ENTITY % adonis.name "adonis">
<!ENTITY % adonis.pfx "%adonis.name;:">
```

Further a namespace for the extension is to be defined:

```
<!ENTITY  % xwdl.process.xmlns.extra 'xmlns:%adonis.name;
CDATA #FIXED "http://www.woanders.com"'>
```

The xwdl.process.xmlns.extra entity was included in the attributes for the process element in the main xwdl.dtd file. By defining the namespace here, we can annotate the specific elements with the name prefix (adonis in this case).
Additional attributes would be declared via stand alone attribute lists like in the following example. We add an extra attribute to the element if with an attribute name which is prefiexed by the namespace in the extension DTD. It is defined as implied, so it is not mandatory

```
<!ENTITY % adonis.if.condition.qname "%adonis.pfx;condition">
<!ATTLIST if
   %adonis.if.condition.qname;       CDATA       #IMPLIED
>
```

Changes in the element structure are implemented by defining the new elements in the extension DTD and then by defining the corresponding . . . content entity from the xwdl.dtd file. The example declares a new element adonis:followingProcess with four attributes and states the new content model for the activity. Thereby we can use the new element within activity elements after the original content (agents and forms).
It is a requirement, that the original content of the elements like described in the xwdl.dtd file is not altered but merely augmented.

```
<!ENTITY % adonis.followingProcess.qname "%adonis.pfx;followingProcess">
<!ELEMENT %adonis.followingProcess.qname; EMPTY>
<!ATTLIST %adonis.followingProcess.qname;
   id CDATA #REQUIRED
   name CDATA #IMPLIED
   version CDATA #IMPLIED
   gs CDATA #IMPLIED
 >

<!ENTITY % xwdl.activity.content "(agent*,form*,%adonis.followingProcess.qname;*)" >
```

System steps can be extended as follows:

```
<!ENTITY % adonis.varout.qname "%adonis.pfx;varout">
<!ELEMENT %adonis.varout.qname; EMPTY>
<!ATTLIST %adonis.varout.qname;
task CDATA #REQUIRED
>
<!ENTITY % xwdl.system.content "(%adonis.varout.qname;)?">
```

The whole extension dtd looks like this:

```
<!ENTITY % adonis.name "adonis">
<!ENTITY % adonis.pfx "%adonis.name;:">
<!ENTITY  % xwdl.process.xmlns.extra 'xmlns:%adonis.name; CDATA #FIXED "http://www.woa

<!ENTITY % adonis.if.condition.qname "%adonis.pfx;condition">
<!ATTLIST if
   %adonis.if.condition.qname;       CDATA       #IMPLIED
 >

<!ENTITY % adonis.followingProcess.qname "%adonis.pfx;followingProcess">
<!ELEMENT %adonis.followingProcess.qname; EMPTY>
<!ATTLIST %adonis.followingProcess.qname;
   id CDATA #REQUIRED
   name CDATA #IMPLIED
   version CDATA #IMPLIED
   gs CDATA #IMPLIED
 >
```

```
<!ENTITY % adonis.varout.qname "%adonis.pfx;varout">
<!ELEMENT %adonis.varout.qname; EMPTY>
<!ATTLIST %adonis.varout.qname;
task CDATA #REQUIRED
>

<!ENTITY % xwdl.activity.content "(agent*,form*,%adonis.followingProcess.qname;*)" >

<!ENTITY % xwdl.system.content "(%adonis.varout.qname;)?">

<![ INCLUDE [
<!ENTITY % xwdl.mod SYSTEM "./xwdl.dtd">
%xwdl.mod;]]>
```

4.2 An Example

An extended XDWL file using the above extension dtd could look like this:
<?xml version="1.0" encoding="UTF-8"?> <?xwdl extensionHandler="com.groiss.wf.xwdl.NullExtensi
<!DOCTYPE process SYSTEM "./conf/adonis.dtd"> <process xmlns:xdwl='http://www.groiss.com'
xmlns:adonis="http://www.woanders.com"

```
<?xml version="1.0" encoding="UTF-8"?>
<?xwdl extensionHandler="com.groiss.wf.xwdl.NullExtensionHandler"?>
<!DOCTYPE process SYSTEM "./conf/adonis.dtd">
<process xmlns:xdwl='http://www.groiss.com' xmlns:adonis="http://www.woanders.com"
 id="all_things_x" version="1"
name="all control structures" description="Test the control structures"
application="default">
  <forms>
    <formdecl id="form" typ="Jobform" />
  </forms>
  <timing timeoutaction="none" maxtime="10" maxtimeunit="minutes">
    <activity id="adm_task">
      <agent string="all" />
      <agent string="r" />
      <form name="form" />
    </activity>
  </timing>
  <label id="first" />
  <activity id="start_task">
    <agent string="all" />
    <form name="form" />
  </activity>
  <loop>
    <choice>
      <case name="first choice: an if">
```

```
 <if condition="(form.type = &quot;hw&quot;)" adonis:condition="ccccccccccc">
   <then>
    <activity id="hw_task">
     <agent string="all" />
     <form name="form" />
     <adonis:followingProcess id="idid id" gs="gsgsgs"/>
     <adonis:followingProcess id="idid id2" gs="gsgsgs2"/>
    </activity>
   </then>
   <elsif condition="(form.type = &quot;sw&quot;)">
    <then>
     <activity id="swx_task" name="the name of this task">
      <agent string="form.recipient" />
      <form name="form" />
     </activity>
    </then>
   </elsif>
   <elsif condition="(form.type = &quot;adm&quot;)">
    <then>
     <activity id="adm_task">
      <agent string="first:user" />
      <form name="form" />
     </activity>
    </then>
   </elsif>
   <else>
    <activity id="none_task">
     <agent string="first:user" />
     <form name="form" />
    </activity>
   </else>
  </if>
</case>
<case name="second choice: a while">
  <while condition="(form.type = &quot;hw&quot;)">
   <activity id="while_task1">
    <agent string="form.recipient" />
    <form name="form" />
   </activity>
   <label id="in_while" />
   <activity id="while_task2">
    <agent string="form.recipient" />
    <form name="form" />
   </activity>
   <activity id="while_task3">
    <agent string="form.recipient" />
    <form name="form" />
```

```
    </activity>
  </while>
</case>
<case name="third choice: a loop">
  <loop>
    <activity id="loop_task">
      <agent string="form.recipient" />
      <form name="form" />
    </activity>
    <exit condition="(form.type = &amp;quot;hw&amp;quot;)" />
  </loop>
</case>
<case name="fourth choice: system steps">
  <system methodcall="com.groiss.demo.SystemSteps.emptyMethod()">
     <adonis:varout task="something"/>
  </system>
  <activity id="between_task">
    <agent string="form.recipient" />
    <form name="form" />
  </activity>
  <system methodcall="com.groiss.demo.SystemSteps.emptyMethod()" />
  <system methodcall="com.groiss.demo.SystemSteps.emptyMethod()" />
  <activity id="aftersys_task">
    <agent string="form.recipient" />
    <form name="form" />
  </activity>
</case>
<case name="fifth choice: andpar">
  <andpar>
    <parallel>
      <activity id="andpar1_task">
        <agent string="form.recipient" />
        <form name="form" />
      </activity>
    </parallel>
    <parallel>
      <activity id="andpar2_task">
        <agent string="all" />
        <form name="form" />
      </activity>
    </parallel>
    <parallel>
      <activity id="andpar3_task">
        <agent string="form.recipient" />
        <form name="form" />
      </activity>
    </parallel>
```

```
      </andpar>
    </case>
    <case name="sixth choice: orpar">
      <orpar>
        <parallel>
          <activity id="orpar1_task">
            <agent string="form.recipient" />
            <form name="form" />
          </activity>
        </parallel>
        <parallel>
          <activity id="orpar2_task">
            <agent string="form.recipient" />
            <form name="form" />
          </activity>
        </parallel>
        <parallel>
          <activity id="orpar3_task">
            <agent string="form.recipient" />
            <form name="form" />
          </activity>
        </parallel>
      </orpar>
    </case>
    <case name="eight choice: subprocesses">
      <call id="subflow1">
        <form name="form" />
      </call>
    </case>
    <case name="nineth choice: goto (into the while)">
      <goto label="in_while" />
    </case>
  </choice>
  <exit condition="(form.finished = 1)" />
 </loop>
</process>
```

5 Extension API

Parsing a standard XWDL-file without extensions is done by @enterprise itself.
For the proper treatment of extension attributes and extension elements, we define a callback-interface. We will use the JDOM-API [2] for processing.

```
package com.groiss.wf.xwdl;
import org.jdom.Element;
import com.dec.avw.core.ProcessDefinition;
import com.dec.avw.core.Step;

public interface IExtensionHandler {
  public void init();
  public void handle(Element e, Step s, ProcessDefinition pd);
}
```

Call details:

- for extended elements: when the element is recognized, processing of the JDOM-tree of the element is done by the handler. The tree walker in @enterprise will never step "into" such a subtree.
- for extended attributes: when the containing element is recognized. The handler is expected to process the extended attributes and nothing else.

The extensionHandler is specified via a processing-instruction in the XWDL-file:

```
<?xwdl extensionHandler="at.adonis.xwdl.ExtensionHandler"?>
```

The processing instruction must be included at the outermost document level (before the root XML element).
For debugging purposes, a NullExtensionHandler can be specified. This handler logs its calls to the system log at log level 0.

```
<?xwdl extensionHandler="com.groiss.wf.xwdl.NullExtensionHandler"?>
```

Bibliography

[1] Modularization of XHTML; http://www.w3.org/TR/xhtml-modularization/

[2] http://www.jdom.org/