Nomad - Module Definition DTD

by Christian Schneider

Table of contents

1. Module Specification v1.2 (Draft)

1.1. Purpose

The Module Specification is a markup language for defining the modules of a modular synthesizer. The specification is designed to support different use cases:

- **documentation** for developers - along with the specification we provide a XSL-stylesheet that transforms the XML data into a html document.
- **module reference** - in the same manner as the developer documentation the XML data can be used to generate a user's manual
- as **blueprint** for generating module instances

1.2. Comparison

The intention of the Module Specification v1.0 is to automate generating module instances of modules supported by the Nord Modular synthesizer. The format was designed only for this purpose and it was impossible to specify other properties than available in modules of this synthesizer model. Version v1.2 tries to be independent of the synthesizer model.

1.2.1. Differences

In this version

- is the definition of substitutions embeded rather than specified in another file
- are substitutions defined in ECMAscript (Javascript)
- polymorphism: modules can inherit properties from other modules
- it is possible to describe the document and it's content

1.2.2. Planned improvements

Future versions of the specification will support

- adding and managing custom modules built from existing ones
- nested modules (requires properties that describe internal cabling)
- templates

2. Describing the Document

The root element always contains a `version` attribute used to determine the version of this specification.

```
<module-spec version="1.2">
...
</module-spec>
```

The first child of the `module-spec`-element is the `header` element which contains all meta data describing the document as a whole.

```
<module-spec version="1.2">
  <header>
    <language code="en" />
    <license><!-- the license of this document --></license>
    <vendor href="http://nmedit.sf.net">The nmedit open source
project</vendor>
    <last-modified>2006-07-14</last-modified>
    <device>
      <name>Nord Modular</name>
      <version>3.03</version>
      <vendor href="http://www.clavia.se">Clavia</vendor>
    </device>
  </header>
  ...
</module-spec>
```

- `language` identifies the english language to be used in this document
- `license` specifies the license assigned to this document
- `vendor` specifies the vendor (and it's url) of this document
- `last-modified` specifies the date of the last modification made to this document
- `device` specifies the synthesizer model's `name`, os-`version`, `vendor` and optionally the vendors url

3. Declarations

To differentiate parameters or interfaces it is possible to associate key-value pairs with them.

```
<module-spec version="1.2">
  <header> ... </header>
  <declarations>
    <class name="type" component="interface" values="input,output"
default="input">
      differs between input and output connectors
    </class>
    <class name="signal" component="interface"
values="audio,master-slave,control,logic" default="audio">
      differs between input and output connectors
    </class>
    <class name="type" component="parameter" values="custom,parameter"
default="parameter">
      differs between usual parameters and custom parameters controlling
display modes
    </class>
  </declarations>
  ...
</module-spec>
```

- In the example above a class named `type` for the `interface`-component is declared with the possible values `input` and `output`. If no class is specified within an interface element the default value `input` is used.
- In the example above a class named `type` for the `parameter`-component is declared with the possible values `custom` and `parameter`. If no class is specified within an parameter element the default value `parameter` is used.
- In the example above there are no classes declared for modules

4. Scripting

The section allows scripting with ECMAScript (javascript). Functions can be declared that transform one or more parameter values into the value they represent.

Example:

```
<module-spec version="1.2">
  <header> ... </header>
  <declarations> ... </declarations>
  <script>
  <![CDATA[
    function beatsPerMinute(var a)
    {
       if (0<=a && a<=127)
       {
          return (24+a)+" bpm";
       }
       else
       {
          return null;
       }
    }
    ... /* more function declarations */
  ]]>
  </script>
  ...
</module-spec>
```

5. Modules

```
<module-spec version="1.2">
  <header> ... </header>
  <declarations> ... </declarations>
  <script> ... </script>
  <modules>
    <module>
      <const name="load" type="float">
        Constant describing the dsp-load.
      </const>

      <param class="indirect" name="x">x-coordinate of module</parameter>
      <param class="indirect" name="y">y-coordinate of module</parameter>

      <param class="label" name="name" type="string" string-limit="16">
        Module name containing up to 16 characters.
      </param>
    </module>

    <module id="5" type="instancible"
extends="default-parameters,constants">
      <name>BPM</name>
      <const name="load" value="0.002" />

      <param class="parameter" id="1" name="BPM" min="0" max="127"
display-value="beatsPerMinute(this)">
        Beats Per Minute
      </param>

      <param name="display-type" class="custom" min="0" max="1" />

      <interface class="type(input),signal(audio)" />
    </module>

    ...
  </modules>
  ...
</module-spec>
```

6. Module Hierarchy

To the definition of a module also belongs their relation to each other.

```
<module-spec version="1.2">
  <header> ... </header>
  <declarations> ... </declarations>
  <script> ... </script>
  <modules> ... </modules>
  <hierarchy>
    <group>
      <label>In/Out</label>
      <description>In- and Outputs</description>
      <group level="xyz">
        <module id="1" />
        <module id="2" />
      </group>
      <group>
        <module id="3" />
        <module id="4" />
      </group>
    </group>
    <group>
      <label>Oscillator</label>
      <description>Oscillators</description>
      <group>
        <module id="5" />
        <module id="6" />
      </group>
      <group>
        <module id="7" />
      </group>
    </group>
    ...
  </hierarchy>
  ...
</module-spec>
```