Configuration Model and Settings

sipXconfig Model files

Model files are used to describe a feature's configuration model, such as their properties, default values and constraints. In addition, in order to add support for a phone or a gateway, you have to create an information model for the device you want to manage.

Information model consist of settings, that are assembled in groups. Each group should consists of related settings. Groups can contain settings or other groups.

sipXconfig models are defined in XML files conforming to the setting.dtd file. The file can be found in the source directory: "main/sipXconfig/neoconf/etc/ ".

In addition to internal properties of groups and settings, model definition contains also user visible elements such as labels and descriptions. In future it will be possible to localize all user visible element by providing standard I18n properties files.

Settings group

Members of the same groups are presented to the end user on one screen. *In future sipXconfig may provide independent screen defintion to allow for other layouts and presenting settings independent of their groups.*

Property	Type	Description	Required
name	attribute	internal group name - cannot be empty	yes
advanced	attribute	if set to true, marks the entire group to be non-essential and should not be shown to user by default.	no (default: no)
hidden	attribute	if set to true group should not be displayed by UI	no (default: no)
if	attribute	property that must be set to use this group (see Conditional settings)	no
unless	attribute	property that must not be set to use this group (see Conditional settings)	no
copy-of	attribute	The name of the group the current group is based on	no
label	element	user visible group label	yes
profileName	element	information used by module generating device configuration (profile)	no
description	element	user visible group description	no
setting	element	definition of the setting that belong to this group	no
group	element	definition of the group that belongs to this group	no
collection	element	definition of the collection of settings that belongs to this group	no

Settings

Example:

```
<setting name="SipUri">
  <label>SIP URI</label>
  <value></value>
  <description>Identifier in SIP address</description>
</setting>
```

Property	Type	Description	Required
name	attribute	internal setting name - cannot be empty	yes
advanced	attribute	if set to true UI will hint that the setting is advanced, for example by displaying it on a separate panel, or only after user requests to see advanced settings, non-essential settings should be set to advanced to improve usability	no (default: no)
hidden	attribute	if set to true it will not be displayed by UI	no (default: no)
if	attribute	property that must be set to use this group (see Conditional settings)	no
unless	attribute	property that must not be set to use this group (see Conditional settings)	no
copy-of	attribute	The internal name of the setting the current setting is copied from	no
label	element	user visible setting label	no
profileName	element	information used by module generating device configuration (profile)	no
type	element	setting type that determines how setting is presented and validated	no - string is default type

value	element	default value of this setting	no - can be empty
description	element	user visible setting description	no

Sharing Settings and Setting Groups

Two separate setting models can share a settings or setting groups. A deep copy is made of a specified setting and variations can be defined.

```
<group name="human">
  <group name="eat">
    <setting name="hamburger"/>
      <group name="fruit">
         <setting name="apple"/>
       </group>
  </group>
</group>
<group name="man" copy-of="/human">
  <setting name="shave">
    <value>face</value>
  </setting>
</group>
<group name="woman" copy-of="/human">
  <setting name="shave">
    <value>legs</value>
  </setting>
  <setting name="giveBirth"/>
 </group>
```

Settings types

Each setting describes a configuration property. A setting is of a particular setting type. The setting type controls how the property is displayed. For example, a string or integer setting type is displayed as a textfield and a boolean type is displayed as a checkbox. Setting types have a "required" flag to indicate if the setting is optional or not (except for boolean, enum and multi-enum types). The required flag is defaulted to "no". Below are descriptions of each supported setting types:

- string
  rendered as text field with maximum and minimum length constraints, optionally can be also constrained by regular expression pattern, required is an optional attribute if it is not set or if it is set to false text box can be left empty

```
<string required="yes" minLen="5" maxLen="30"/>
```

```
<string required="yes"/>
  <pattern>\d(1,4)[a-Z]*</pattern>
</string>
```

If string represents a password sipXconfig will render it using masked edit widget.

```
<string password="yes"/>
```

- integer
  rendered as text field accepting numeric values only, the default range is from 0 to Integer.MAX_INT; the range can be changed by manipulating min and max attributes, if required is not set or if it is set to false, empty values are accepted.

```
<integer min="-17" max="42">
```

- real
  similar to integer but accepting floating point values - the default range is from 0 to Float.MAX_VALUE ()

```
<real min="-17.15" max="42.23">
```

- ipaddr
  rendered as text field accepting IP addresses only, required is an optional attribute if it is not set or if it is set to false text box can be left empty

```
<ipaddr />
```

- hostname
  rendered as text field accepting IP addresses and DNS hostnames, required is an optional attribute if it is not set or if it is set to false text box can be left empty

```
<hostname />
```

- enumeration
  rendered as drop-down list of options, labels are optional, if they are not provided values are displayed as labels

```
<enum>
  <option><label>Red</label><value>0xff0000</value></option>
  <option><label>Green</label><value>0x00ff00</value></option>
  <option><label>Blue</label><value>0x0000ff</value></option>
</enum>
```

- multi enumeration
  similar to enumeration but allows for more than one option to be selected, the value is represented as a bar '|' separated list of option values, the order matters

```
<multi-enum>
  <option><label>Red</label><value>0xff0000</value></option>
  <option><label>Green</label><value>0x00ff00</value></option>
  <option><label>Blue</label><value>0x0000ff</value></option>
</multi-enum>
```

for this setting the value representing Red,Blue list will look like this:

```
<value>0xff0000|0x0000ff</value>
```

- boolean
  rendered as checkboxes, what value is stored for true and false is configurable. By default, true values are 1 and false values are 0.

```
<boolean>
  <true><value>Yes</value></true>
  <false><value>No</value></false>
</boolean>
```

- file
  if the *variable* attribute is set to yes it is rendered as an drop-down list of files in a specified directory - allows for uploading and downloading such files and for selecting out of the set of previously uploaded files. If the variable is set to false you can only upload the file and delete it. There is no dropdown list rendered. The value of the setting is base name of the file (directory is not part).

```
<file required="yes" variable="yes">
  <directory>/tmp</directory>
  <contentType>audio/x-wav</contentType>
</file>
```

- sip-uri
  represents full SIP URI, can be restricted to user part only

  ```
  <sip-uri userPartOnly="yes" />
  ```

Shared Setting Types

You can define a setting type once and share it throughout your model. Declare your types you'd like to share with an "id" attribute, and reference it from where you'd normally use a setting type

Example, if you want "True" and "False" to be define for select values.

```
  <type id="yes-no">
    <boolean>
      <true><value>Yes</value></true>
      <false><value>No</value></false>
    </boolean>
  </type>
  <group name="candyPreferences">
    <setting name="chocolate">
      <type refid="yes-no"/>
    </setting>
    <setting name="taffy">
      <type refid="yes-no"/>
    </setting>
  </group>
```

Conditional settings

Often times, a single settings XML file is shared by several device models. Because of this, some settings may not be applicable to particular models. To handle this situation, the

```
<group>
```

and

```
<setting>
```

elements provide two other attributes:

```
if
```

and

```
unless
```

, which take the name of a property as the condition. If the property specified in the

```
if
```

attribute is defined, then the setting or group will be used. If the property specified in the

```
unless
```

attribute is defined, then the setting or group will not be used.

Currently, only one property each may be specified for

```
if
```

and

```
unless
```

.

Model conditions

Some settings may only be applicable to a particular device model, or may not apply to a particular device model.

The model ID is automatically defined as a property of each model. This is usually the

```
id
```

attribute value of a corresponding Spring bean. For example, the Polycom SoundStation IP 7000 is defined as:

```
<bean id="polycom7000" parent="polycomModel">
```

In this example case, the Polycom 7000 has a built-in property called "polycom7000" that we can use for conditional settings:

```
<!-- This setting will only be used in Polycom 7000s -->
<setting name="setting1" if="polycom7000"> ... </setting>

<!-- This setting will NOT be used for Polycom 7000s -->
<setting name="setting2" unless="polycom7000"> ... </setting>
```

Supported feature conditions

Sometimes, a particular model may be too narrow of a restriction. On product lines with many different models, such as Polycom phones or AudioCodes gateways, many settings are tied to particular features which are supported on some subset of models.

All device models extending from

```
DeviceDescriptor
```

have a

```
supportedFeatures
```

property, which is a set of feature names. These feature names may then be used as properties in the

```
if
```

and

```
unless
```

attributes for settings and groups. The set of supported features are defined in the Spring bean corresponding to the model.

For example, the Polycom SoundStation IP 4000 is defined as follows in

```
polycom-models.beans.xml
```

:

```
<bean id="polycom4000" parent="polycomModel">
  <property name="label" value="Polycom SoundStation IP 4000" />
  <property name="maxLineCount" value="1" />
      <property name="supportedFeatures">
        <set>
          <value>disableCallList</value>
          <value>voiceQualityMonitoring</value>
        </set>
  </property>
</bean>
```

The Polycom 4000 supports the "disableCallList" and "voiceQualityMonitoring" features. These features are checked in the conditional settings:

```
<group name="qualityMonitoring" if="voiceQualityMonitoring">
  ...
</group>
...
<group name="feature">
  <setting name="calllist" if="disableCallList">
    ...
  </setting>
</group>
```

As the above example shows, the group

```
qualityMonitoring
```

and the setting

```
calllist
```

will be used for the Polycom 4000 and other models supporting these features, and not in others.