

Contents

- [Sample Rules Definition File](#)
 - [The Complete XML File](#)
 - [JSON Format File](#)
 - [The conditions Element](#)
 - [The contexts Element](#)
 - [The objectProperties Element](#)

Sample Rules Definition File

The following is an example of an XML rules definition file that demonstrates all of the elements and attributes that are available in the [xml schema](#). It also illustrates a number of validation types. A version of the file in JSON format which describes the exact same rules follows the XML file. The complete files are listed, followed by excerpts from the files with a more detailed description of each section.

The Complete XML File

```
<?xml version="1.0" encoding="UTF-8"?>
<validateThis xsi:noNamespaceSchemaLocation="validateThis.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <conditions>
    <condition name="MustLikeSomething"
      serverTest="getLikeCheese() EQ 0 AND getLikeChocolate() EQ 0"
      clientTest="&{"[name='LikeCheese']}".getValue() == 0 && {"[name='LikeChocolate']}".getValue() == 0;" />
  </conditions>
  <contexts>
    <context name="Register" formName="frmRegister" />
    <context name="Profile" formName="frmProfile" />
  </contexts>
  <objectProperties>
    <property name="UserName" desc="Email Address">
      <rule type="required" contexts="*" />
      <rule type="email" failureMessage="Hey, buddy, you call that an Email Address?" />
    </property>
    <property name="Nickname">
      <rule type="custom" failureMessage="That Nickname is already taken. Please try a different Nickname.">
        <param name="methodName" value="CheckDupNickname" />
        <param name="remoteURL" value="CheckDupNickname.cfm" />
      </rule>
    </property>
    <property name="UserPass" desc="Password">
      <rule type="required" />
      <rule type="rangelength">
        <param name="minlength" value="5" />
        <param name="maxlength" value="10" />
      </rule>
    </property>
    <property name="VerifyPassword">
      <rule type="required" />
      <rule type="equalTo">
        <param name="ComparePropertyName" value="UserPass" />
      </rule>
    </property>
    <property name="UserGroup" clientfieldname="UserGroupId">
      <rule type="required" />
    </property>
    <property name="Salutation">
      <rule type="required" contexts="Profile" />
      <rule type="regex"
        failureMessage="Only Dr, Prof, Mr, Mrs, Ms, or Miss (with or without a period) are allowed.">
        <param name="Regex" value="(Dr|Prof|Mr|Mrs|Ms|Miss)(\.)?$" />
      </rule>
    </property>
    <property name="FirstName">
      <rule type="required" contexts="Profile" />
    </property>
    <property name="LastName">
      <rule type="required" contexts="Profile" />
      <rule type="required" contexts="Register">
        <param name="DependentPropertyName" value="FirstName" />
      </rule>
    </property>
    <property name="LikeOther" desc="What do you like?">
      <rule type="required" condition="MustLikeSomething"
        failureMessage="If you don't like Cheese and you don't like Chocolate, you must like something!">
      </rule>
    </property>
    <property name="HowMuch" desc="How much money would you like?">
      <rule type="numeric" />
    </property>
    <property name="AllowCommunication" desc="May we send you information?"/>
    <property name="CommunicationMethod">
      <rule type="required"
        failureMessage="If you are allowing communication, you must choose a communication method.">
        <param name="DependentPropertyName" value="AllowCommunication" />
        <param name="DependentPropertyValue" value="1" />
      </rule>
    </property>
  </objectProperties>
</validateThis>
```

JSON Format File

```
{
  "validateThis" : {
    "conditions" : [
      {
        "name": "MustLikeSomething",
        "serverTest": "getLikeCheese() EQ 0 AND getLikeChocolate() EQ 0",
        "clientTest": "&{"[name='LikeCheese']}".getValue() == 0 && {"[name='LikeChocolate']}".getValue() == 0;"
      }
    ],
    "contexts" : [
      {
        "name": "Register", "formName": "frmRegister"
      },
      {
        "name": "Profile", "formName": "frmProfile"
      }
    ],
    "objectProperties" : [
      {
        "name": "UserName", "desc": "Email Address",
        "rules": [
          {
            "type": "required", "contexts": "*"
          },
          {
            "type": "email", "failureMessage": "Hey, buddy, you call that an Email Address?"
          }
        ]
      },
      {
        "name": "Nickname",
        "rules": [
          {
            "type": "custom", "failureMessage": "That Nickname is already taken. Please try another",
            "params": [
              {
                "name": "methodName", "value": "CheckDupNickname"
              },
              {
                "name": "remoteURL", "value": "CheckDupNickname.cfm"
              }
            ]
          }
        ]
      },
      {
        "name": "UserPass", "desc": "Password",
        "rules": [
          {
            "type": "required"
          },
          {
            "type": "rangelength",
            "params": [
              {
                "name": "minlength", "value": "5"
              },
              {
                "name": "maxlength", "value": "10"
              }
            ]
          }
        ]
      },
      {
        "name": "VerifyPassword",
        "rules": [
          {
            "type": "required"
          }
        ]
      }
    ]
  }
}
```

```

    {"type": "equalTo",
     "params" : [
       {"name": "ComparePropertyName", "value": "UserPass"}
     ]
    },
    {"name": "UserGroup", "clientFieldName": "UserGroupID",
     "rules" : [
       {"type": "required"}
     ]
    },
    {"name": "Salutation",
     "rules" : [
       {"type": "required", "contexts": "Profile"},
       {"type": "regex", "failureMessage": "Only Dr, Prof, Mr, Mrs, Ms, or Miss (with or without a period) are allowed.",
        "params" : [
          {"name": "Regex", "value": "(Dr|Prof|Mr|Mrs|Ms|Miss) (\\.)??"}
        ]
       }
     ]
    },
    {"name": "FirstName",
     "rules" : [
       {"type": "required"}
     ]
    },
    {"name": "LastName",
     "rules" : [
       {"type": "required", "contexts": "Profile"},
       {"type": "required", "contexts": "Register",
        "params" : [
          {"name": "DependentPropertyName", "value": "FirstName"}
        ]
       }
     ]
    },
    {"name": "LikeOther", "desc": "What do you like?",
     "rules" : [
       {"type": "required", "condition": "MustLikeSomething", "failureMessage": "If you don't like cheese and you don't like chocolate, you must like something!"}
     ]
    },
    {"name": "HowMuch", "desc": "How much money would you like?",
     "rules" : [
       {"type": "numeric"}
     ]
    },
    {"name": "AllowCommunication", "desc": "May we send you information?"
    },
    {"name": "CommunicationMethod",
     "rules" : [
       {"type": "required", "failureMessage": "if you are allowing communication, you must choose a method",
        "params" : [
          {"name": "DependentPropertyName", "value": "AllowCommunication"},
          {"name": "DependentPropertyValue", "value": "1"}
        ]
       }
     ]
    }
  ]
}

```

The conditions Element

```

<conditions>
  <condition name="MustLikeSomething"
  serverTest="getLikeCheese() EQ 0 AND getLikeChocolate() EQ 0"
  clientTest="&#34;[name='LikeCheese']").getValue() == 0 &#34; &#34;[name='LikeChocolate']").getValue() == 0;" />
</conditions>

"conditions" : [
  {"name": "MustLikeSomething",
   "serverTest": "getLikeCheese() EQ 0 AND getLikeChocolate() EQ 0",
   "clientTest": "&#34;[name='LikeCheese']").getValue() == 0 &#34; &#34;[name='LikeChocolate']").getValue() == 0;"
  }
]

```

Here we are defining one condition, and giving it a unique name of *MustLikeSomething*. We then specify how the condition is to be evaluated on both the server and the client.

In the case of the server we are saying:

"This condition is met when the value of *getLikeCheese()* is zero and the value of *getLikeChocolate()* is zero."

While on the client we are saying:

"This condition is met when the value of the html element with a name attribute of *LikeCheese* is zero and the value of the html element with a name attribute of *LikeChocolate* is zero."

Note that because we are placing actual JavaScript code into an xml attribute / json string we must escape the special characters.

This condition, *MustLikeSomething* will be used in a rule definition later in the file.

The contexts Element

```

<contexts>
  <context name="Register" formName="frmRegister" />
  <context name="Profile" formName="frmProfile" />
</contexts>

"contexts" : [
  {"name": "Register", "formName": "frmRegister"},
  {"name": "Profile", "formName": "frmProfile"}
]

```

Here we are defining two contexts and pointing them at particular forms. We are doing this because each form has a unique name, and specifying these here, in the configuration file, means that we won't have to specify the form name when asking the framework to generate client-side validations for us.

If the forms had the same name, e.g., frmUser, there would be no reason to include this *contexts* element at all. It is **not** required to make use of contexts for validation rules, its sole purpose is to point a context at a particular form.

The objectProperties Element

We define all of the properties that the framework needs to know about in the *objectProperties* element. There are two reasons that a property might need to be defined to the framework:

1. It requires that validation rules be defined for it.
2. It needs a description recorded for it. E.g., if it relates to another property with a rule (e.g., via an equalTo rule type).

Each property's rules is followed by a textual description of what the element implies.

```

<property name="UserName" desc="Email Address">
  <rule type="required" contexts="*" />
  <rule type="email" failureMessage="Hey, buddy, you call that an Email Address?" />
</property>

{"name": "UserName", "desc": "Email Address",
 "rules": [
  {"type": "required", "contexts": "*"},
  {"type": "email", "failureMessage": "Hey, buddy, you call that an Email Address?"}
 ]
}

```

- The descriptive name of the property is *Email Address*.
- The UserName property is required, and must be a valid email address.
- Both rules apply to all contexts. Note that you can indicate that a rule belongs to all contexts by specifying *contexts="*"* or by simply leaving the context attribute out of the definition.
- If the UserName does not contain a valid email address, the message *Hey, buddy, you call that an Email Address?* should be returned.

```

<property name="Nickname">
  <rule type="custom" failureMessage="That Nickname is already taken. Please try a different Nickname.">
    <param name="methodName" value="CheckDupNickname" />
    <param name="remoteURL" value="CheckDupNickname.cfm" />
  </rule>
</property>

```

```
{
  "name": "Nickname",
  "rules": [
    {
      "type": "custom", "failureMessage": "That Nickname is already taken. Please try another",
      "params": [
        {
          "name": "methodName", "value": "CheckDupNickname",
          "name": "remoteURL", "value": "CheckDupNickname.cfm"
        }
      ]
    }
  ]
}
```

- The Nickname property will be tested on both the client and the server. On the server the method *CheckDupNickname()*, which resides in the User Business Object, will be evaluated. On the client, an ajax call will be made to the url *CheckDupNickname.cfm*, with the result being evaluated by the jQuery plugin.
- The custom rule applies to all contexts (because the contexts attribute is missing).
- If the custom validation fails, the message *"That Nickname is already taken. Please try a different Nickname."* should be returned.

```
<property name="UserPass" desc="Password">
  <rule type="required" />
  <rule type="rangelength">
    <param name="minlength" value="5" />
    <param name="maxlength" value="10" />
  </rule>
</property>
{"name": "UserPass", "desc": "Password",
 "rules": [
  {"type": "required"},
  {"type": "rangelength",
   "params": [
    {"name": "minlength", "value": "5"},
    {"name": "maxlength", "value": "10"}
   ]
  }
 ]
}
```

- The descriptive name of the property is *Password*.
- The UserPass property is required, and must be between 5 and 10 characters long.

```
<property name="VerifyPassword">
  <rule type="required" />
  <rule type="equalTo">
    <param name="ComparePropertyName" value="UserPass" />
  </rule>
</property>
{"name": "VerifyPassword",
 "rules": [
  {"type": "required"},
  {"type": "equalTo",
   "params": [
    {"name": "ComparePropertyName", "value": "UserPass"}
   ]
  }
 ]
}
```

- Note that although the descriptive name of the property is *Verify Password*, whereas the property name is *VerifyPassword* (without a space), we do not have to supply a *desc* attribute in this scenario as the framework is smart enough to add appropriate spaces when a property name is camelCased.
- The VerifyPassword property is required, and must be the same as the UserPass property.

```
<property name="UserGroup" clientfieldname="UserGroupId">
  <rule type="required" />
</property>
{"name": "UserGroup", "clientFieldName": "UserGroupID",
 "rules": [
  {"type": "required"}
 ]
}
```

- The name of the form field which will contain the value to be evaluated is *UserGroupID*.
- The UserGroup property is required.

```
<property name="Salutation">
  <rule type="required" contexts="Profile" />
  <rule type="regex"
  failureMessage="Only Dr, Prof, Mr, Mrs, Ms, or Miss (with or without a period) are allowed.">
    <param name="Regex" value="^(Dr|Prof|Mr|Mrs|Ms|Miss) (\.)?$" />
  </rule>
</property>
{"name": "Salutation",
 "rules": [
  {"type": "required", "contexts": "Profile"},
  {"type": "regex", "failureMessage": "Only Dr, Prof, Mr, Mrs, Ms, or Miss (with or without a period) are allowed.",
   "params": [
    {"name": "Regex", "value": "^(Dr|Prof|Mr|Mrs|Ms|Miss) (\.)?$"}
   ]
  }
 ]
}
```

- The Salutation property is required, but only in the context of *Profile*.
- The Salutation property must conform to a regular expression.
- If the Salutation does not conform to the regular expression, the message *"Only Dr, Prof, Mr, Mrs, Ms, or Miss (with or without a period) are allowed."* should be returned.

```
<property name="FirstName">
  <rule type="required" contexts="Profile" />
</property>
{"name": "FirstName",
 "rules": [
  {"type": "required"}
 ]
}
```

- The FirstName property is required, but only in the context of *Profile*.

```
<property name="LastName">
  <rule type="required" contexts="Profile" />
  <rule type="required" contexts="Register">
    <param name="DependentPropertyName" value="FirstName" />
  </rule>
</property>
{"name": "LastName",
 "rules": [
  {"type": "required", "contexts": "Profile"},
  {"type": "required", "contexts": "Register",
   "params": [
    {"name": "DependentPropertyName", "value": "FirstName"}
   ]
  }
 ]
}
```

- The LastName property is always required in the context of *Profile*, and is also required in the context of *Register*, but only if a FirstName has also been specified.

```
<property name="LikeOther" desc="What do you like?">
  <rule type="required" condition="MustLikeSomething"
  failureMessage="If you don't like Cheese and you don't like Chocolate, you must like something!">
```

```

</rule>
</property>
{"name":"LikeOther","desc":"What do you like?",
 "rules" : [
  {"type":"required","condition":"MustLikeSomething","failureMessage":"If you don't like cheese and you don't like chocolate, you must like something!"}
 ]
}

```

- The descriptive name of the property is *What do you like?*
- The LikeOther property is required in all contexts, but only if the condition *MustLikeSomething* evaluates to *true*. Note that that condition was defined in the *conditions* element described above.
- If the LikeOther is not provided, the message *"If you don't like Cheese and you don't like Chocolate, you must like something!"* should be returned.

```

<property name="HowMuch" desc="How much money would you like?">
  <rule type="numeric" />
</property>

```

```

{"name":"HowMuch","desc":"How much money would you like?",
 "rules" : [
  {"type":"numeric"}
 ]
}

```

- The descriptive name of the property is *How much money would you like?*
- The HowMuch property must contain a numeric value.

```

<property name="AllowCommunication" desc="May we send you information?"/>
<property name="CommunicationMethod">
  <rule type="required"
    failureMessage="If you are allowing communication, you must choose a communication method.">
    <param name="DependentPropertyName" value="AllowCommunication" />
    <param name="DependentPropertyValue" value="1" />
  </rule>
</property>

```

```

{"name":"AllowCommunication","desc":"May we send you information?"
},
{"name":"CommunicationMethod",
 "rules" : [
  {"type":"required","failureMessage":"if you are allowing communication, you must choose a method",
   "params" : [
    {"name":"DependentPropertyName","value":"AllowCommunication"},
    {"name":"DependentPropertyValue","value":"1"}
   ]
 }
 ]
}

```

- The descriptive name of the AllowCommunication property is *May we send you information?*. Note that this property does not have any rules assigned to it, but is being defined so that the rule that makes use of it (defined on the *CommunicationMethod* property) can generate a friendly validation failure message.
- The CommunicationMethod property is required in all contexts, but only if the AllowCommunication property has been assigned a value of 1.
- If the CommunicationMethod is not provided, the message *"If you are allowing communication, you must choose a communication method."* should be returned.