TSM8-?.?
Validation of Portrayal Inputs

NIWC Atlantic

for

TSM8

Issues

- How to constrain and validate user input to the portrayal?
  - Constrain: Limit the value domain available to the user
    - Disable inactive context parameters
    - Use enumerations to eliminate free-form input (when possible)
  - Validate: Ensure user-entered values are reasonable prior to further processing
    - Ensure input values are within the expected value domain
    - Ensure input values conform to an expected pattern (e.g. ###-####)
    - Ensure input values are logically consistent with respect to one another

Issues

- How to provide user feedback on validation errors?
  - Indicate context parameter(s) which are source of error(s)
    - Supports a machine-readable UI implementation
  - Provide language-independent error message(s)
    - With optional icon
    - Supports a language-independent machine-readable UI

Research

- UML: Object Constraint Language (OCL)
  - Provides constraint and object query expressions
  - example: a person is younger than their parents

    context Person inv: self.parents->forAll(p|p.age > self.age)
- XML: Schematron (XSLT)
  - Rule-based validation language which leverages XPath
  - Assert / Report

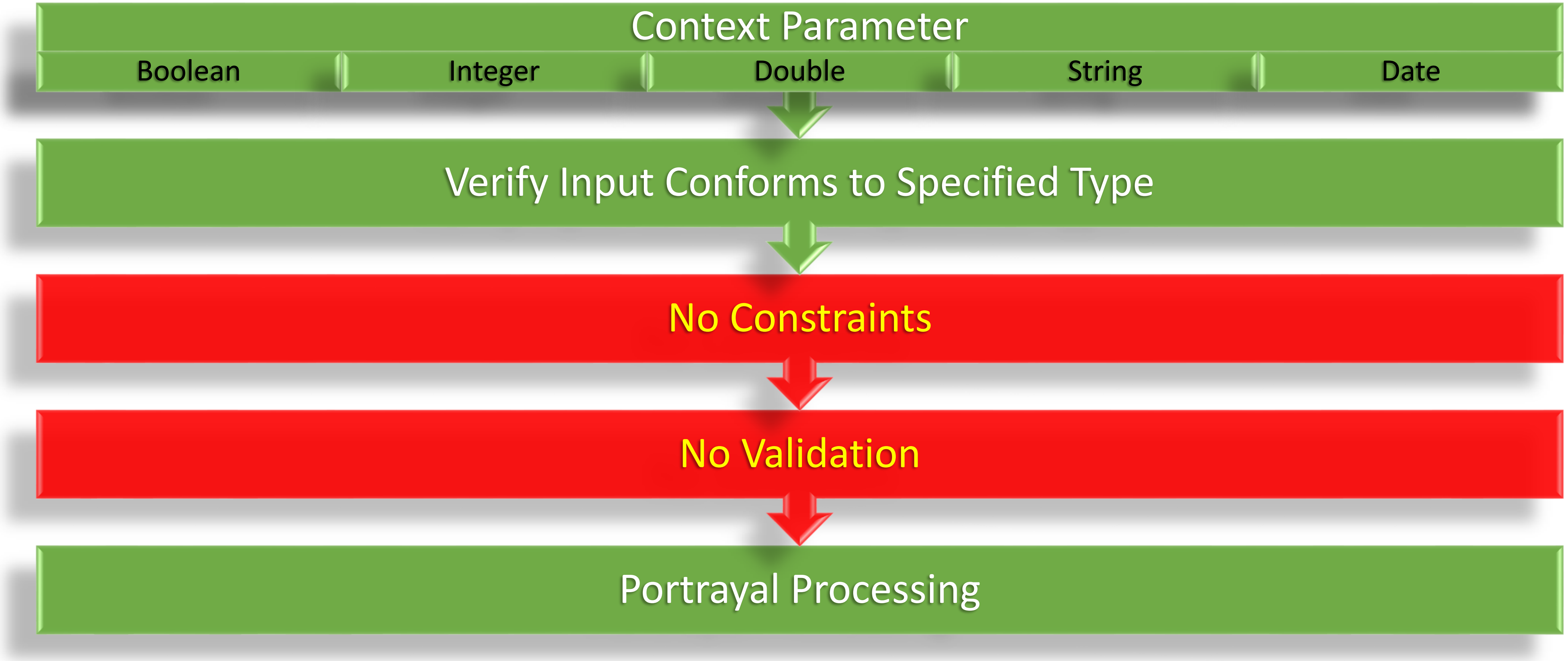
    <assert test="ContractDate &lt; current-date()">Future contracts not allowed.</assert>
- Extensible Application Markup Language (XAML)
  - Initialize structured values and objects

    <Element1 IsEnabled="{Binding ElementName=Element2, Path=IsChecked}"/>

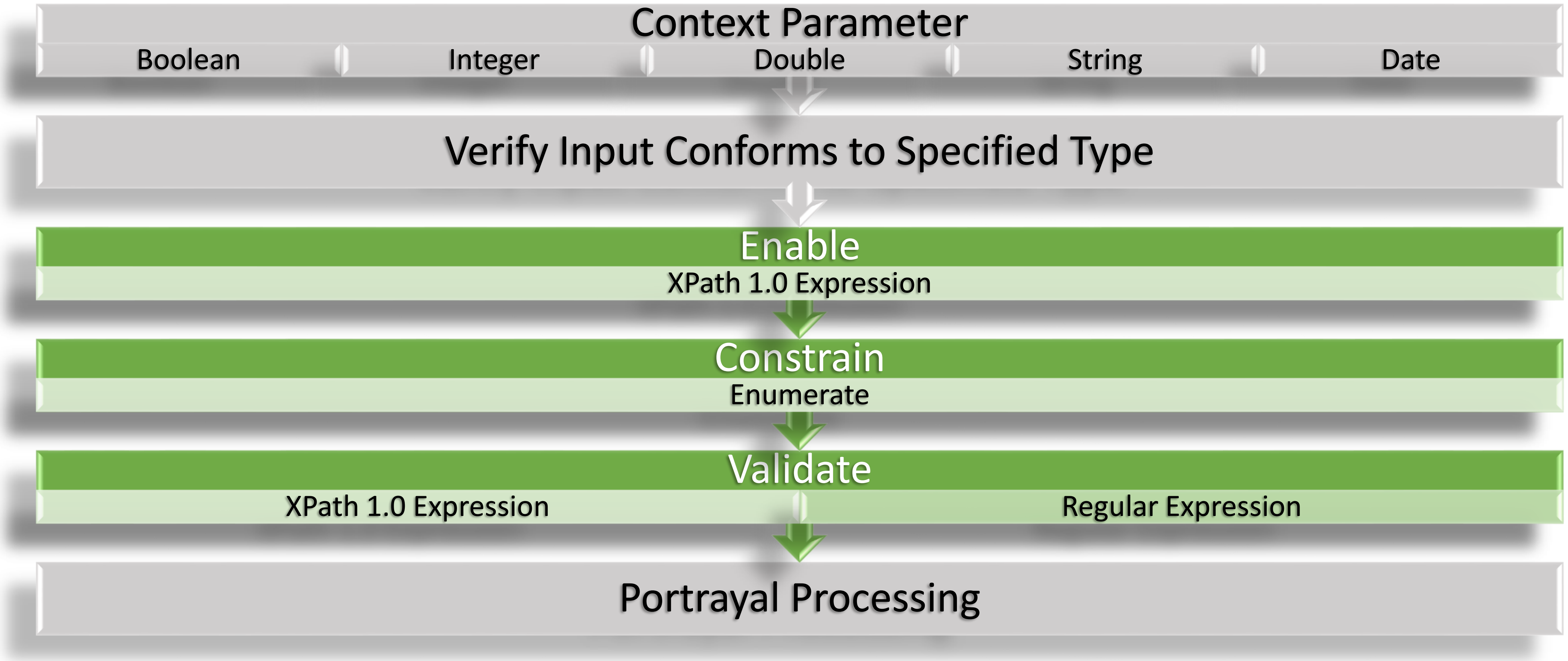
Design Overview

Process and Implementation

Current Process



Proposed Process



Proposed Implementation: Overview

- Update S-100 schema of portrayal catalog <parameter> elements contained within the <context> element
  - new attribute: enable
  - new elements: constrain, validate
- Leverage XPath 1.0 and regular expressions to
  - Disable inactive context parameters
  - Provide validation rules
- Provide machine-readable, language-independent error messages
  - With optional icons
  - Supporting consistent UI implementation

Proposed Implementation: Overview

1. ECDIS uses provided enumerations to restrict user input
   - Combo boxes, etc.

2. ECDIS generates simple XML document encoding context parameter values

```
<parameter1>value1</parameter1>
<parameter2>value2</parameter2>
```

3. ECDIS determines enabled parameters

4. For each enabled context parameter
   - ECDIS processes validation rules and ensures all evaluate to true

Why not use Schematron or XSLT?

- Performance
  - Avoid adding (more) XSLT processing overhead to portrayal generation
- Complexity
  - Both Schematron and XSLT produce a report (an XML document)
    - Must be described by an additional schema (e.g. SVRL)
  - Schematron file must be compiled to XSLT prior to use (typically)
  - Portrayal extensions needed to deliver / update the Schematron or XSLT file
  - No standard provision for language independence
    - (could reference a message similar to alert catalog)
- No advantage over XPath

Design Details

Implementation and examples

Proposed Implementation: enable

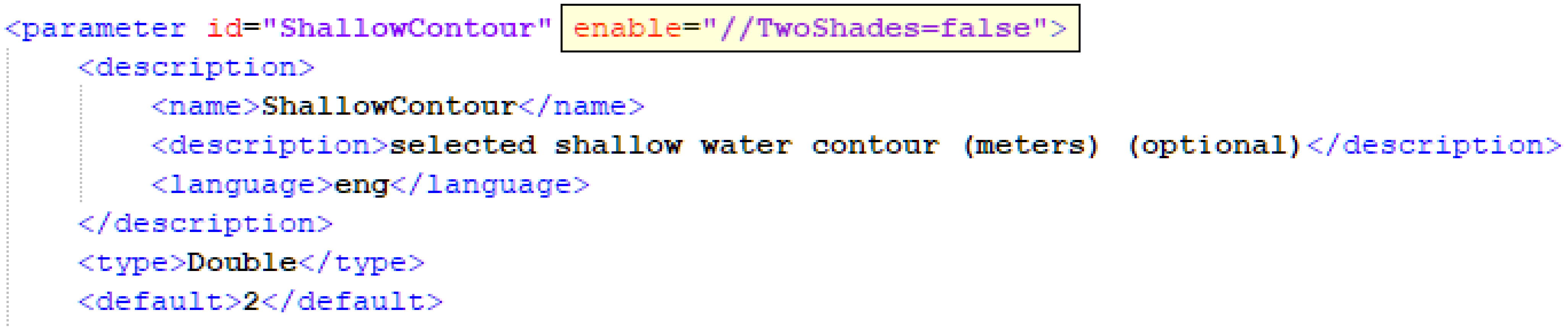
- Attribute on *parameter* or *validate* element specifying an XPath expression
  - Expression must evaluate to true or false
  - Used to conditionally enable a context parameter or validation rule
    - e.g. enable *shallow contour* and *deep contour* when *two shades* is false
- If expression evaluates to false
  - Disable any corresponding UI component
  - Do not perform any associated validation

```
<context>
    …
    <parameter id="XYZ" enable="xpath expression">
        …
    </parameter>
    …
</context>
```

Example: Enable a context parameter

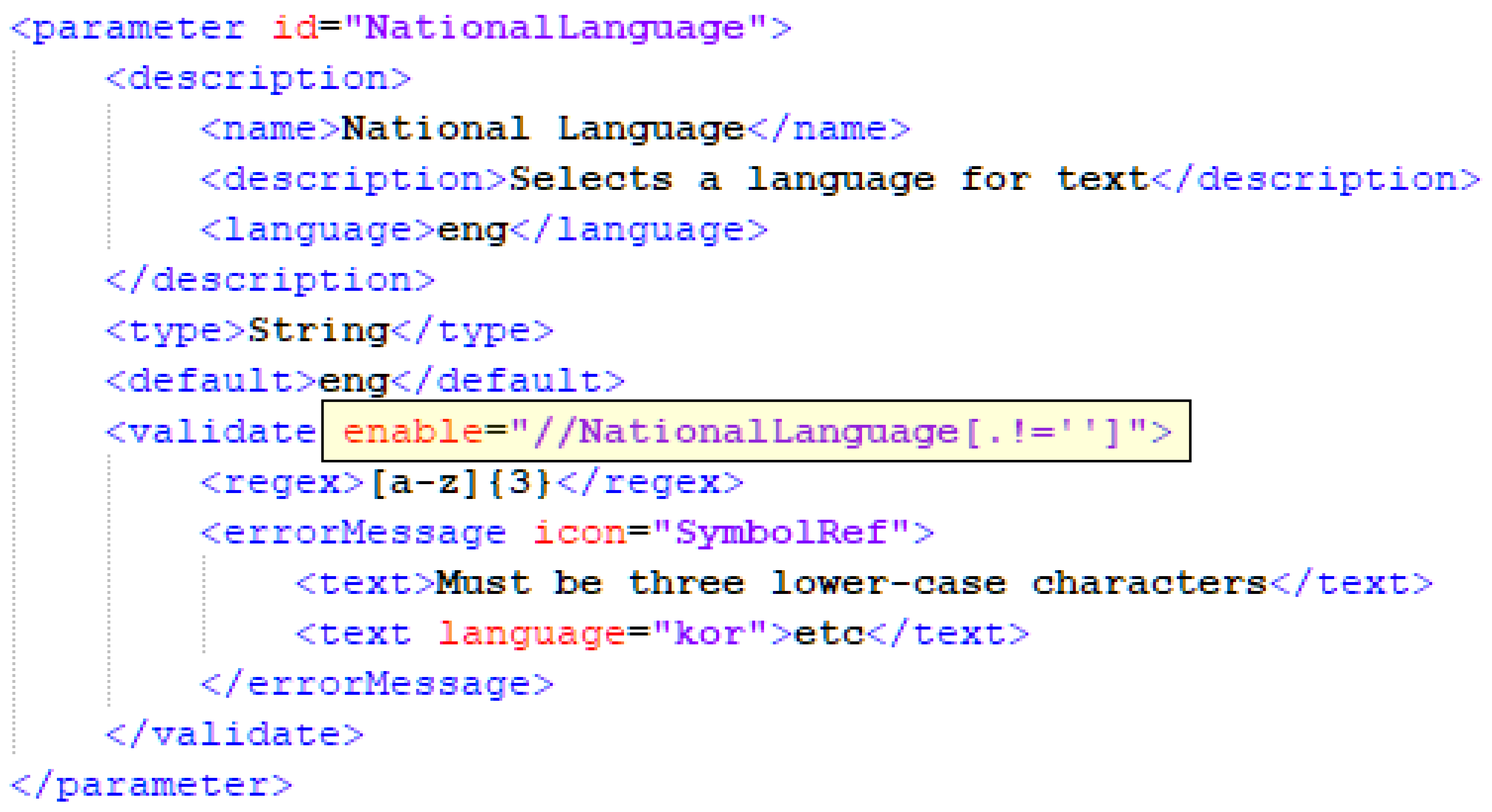
Sample input:

```
…
<TwoShades>true</TwoShades>
<ShallowContour>2</ShallowContour>
<SafetyContour>30</SafetyContour>
<DeepContour>45</DeepContour>
```



```
<parameter id="ShallowContour" enable="//TwoShades=false">
    <description>
        <name>ShallowContour</name>
        <description>selected shallow water contour (meters) (optional)</description>
        <language>eng</language>
    </description>
    <type>Double</type>
    <default>2</default>
```

Example: Enable a validation

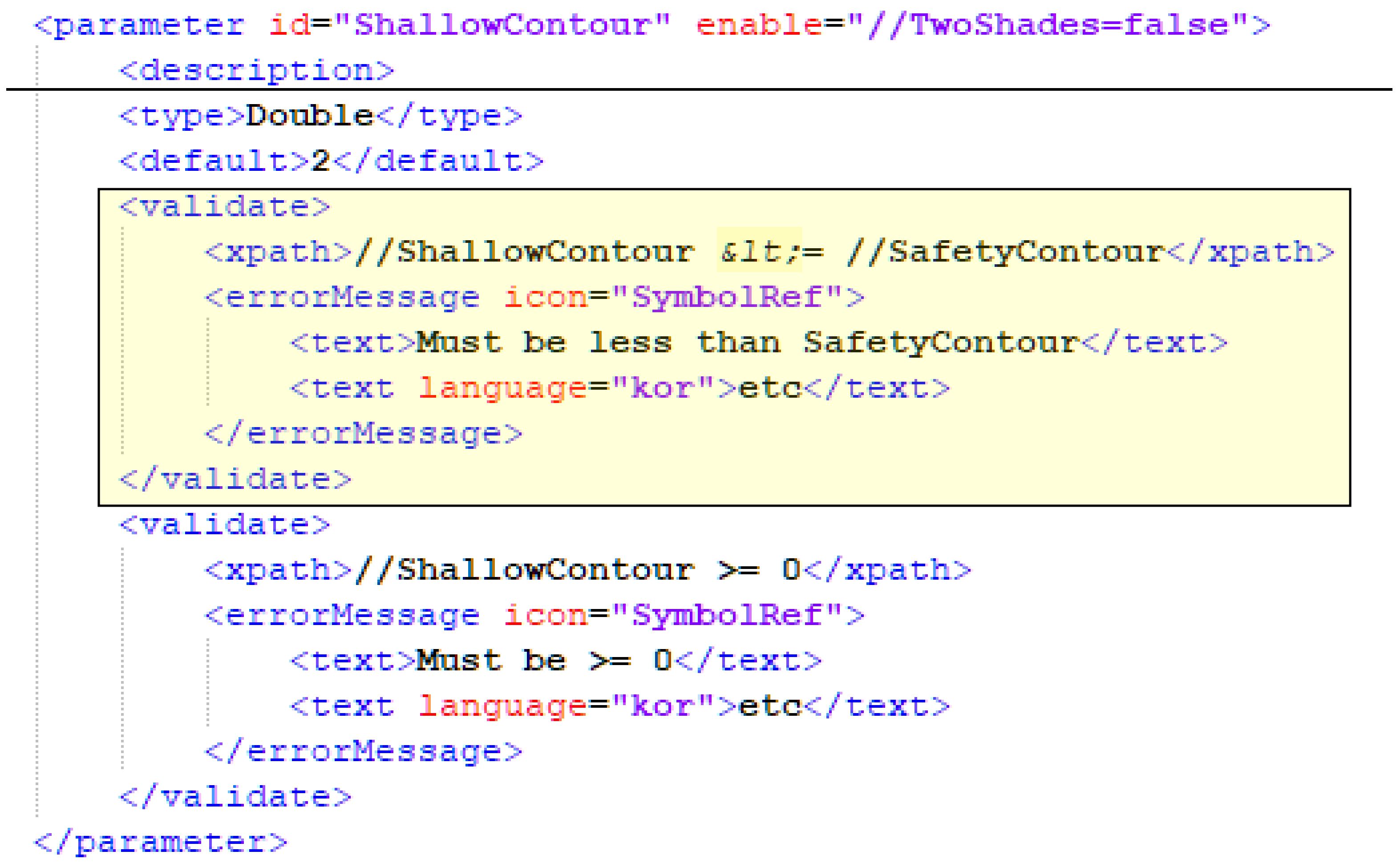


```xml
<parameter id="NationalLanguage">
    <description>
        <name>National Language</name>
        <description>Selects a language for text</description>
        <language>eng</language>
    </description>
    <type>String</type>
    <default>eng</default>
    <validate enable="//NationalLanguage[.!='']">
        <regex>[a-z]{3}</regex>
        <errorMessage icon="SymbolRef">
            <text>Must be three lower-case characters</text>
            <text language="kor">etc</text>
        </errorMessage>
    </validate>
</parameter>
```

Proposed Implementation: <validate>

- Provides a validation rule for a context parameter
- Includes language-independent error message with optional icon
- Validation Rule is either XPath or regex:
  - <xpath>expression</xpath>
    - XPath 1.0 expression which evaluates to true or false
    - Should evaluate to true prior to generating portrayal
  - <regex>expression</regex>
    - W3C XML Standard Part 2 Appendix F (Regular Expressions)
    - Should match prior to generating portrayal
- Individual validation rules can be enabled / disabled via optional enable attribute as previously described

Example: XPath Validation



```xml
<parameter id="ShallowContour" enable="//TwoShades=false">
    <description>
    <type>Double</type>
    <default>2</default>
    <validate>
        <xpath>//ShallowContour &lt;= //SafetyContour</xpath>
        <errorMessage icon="SymbolRef">
            <text>Must be less than SafetyContour</text>
            <text language="kor">etc</text>
        </errorMessage>
    </validate>
    <validate>
        <xpath>//ShallowContour >= 0</xpath>
        <errorMessage icon="SymbolRef">
            <text>Must be >= 0</text>
            <text language="kor">etc</text>
        </errorMessage>
    </validate>
</parameter>
```

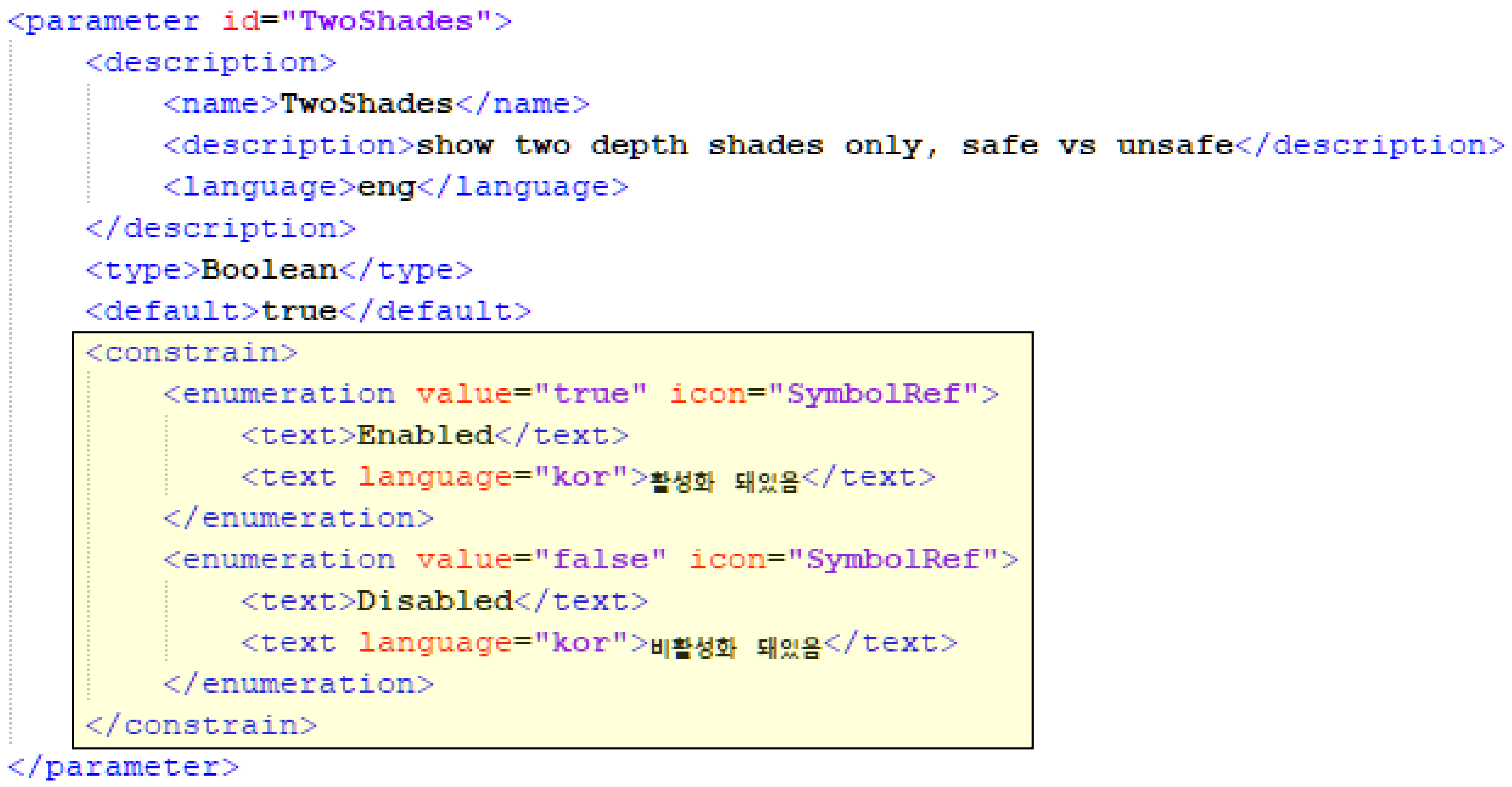
Example: Regular Expression Validation

```xml
<parameter id="NationalLanguage">
    <description>
        <name>National Language</name>
        <description>Selects a language for text</description>
        <language>eng</language>
    </description>
    <type>String</type>
    <default>eng</default>
    <validate>
        <regex>[a-z]{3}</regex>
        <errorMessage icon="SymbolRef">
            <text>Must be three lower-case characters</text>
            <text language="kor">etc</text>
        </errorMessage>
    </validate>
</parameter>
```

Proposed Implementation: <constrain>

- Constrain a context parameter value to a set of enumerated values
- Used to eliminate free-form input for any context parameter type
- Can also be used to provide meaningful selection values
  - E.g. “Rock” vice “2”
- Values are provided in <enumeration> elements
  - Language independent with optional icon

Example: Constrain



```xml
<parameter id="TwoShades">
    <description>
        <name>TwoShades</name>
        <description>show two depth shades only, safe vs unsafe</description>
        <language>eng</language>
    </description>
    <type>Boolean</type>
    <default>true</default>
    <constrain>
        <enumeration value="true" icon="SymbolRef">
            <text>Enabled</text>
            <text language="kor">활성화 돼있음</text>
        </enumeration>
        <enumeration value="false" icon="SymbolRef">
            <text>Disabled</text>
            <text language="kor">비활성화 돼있음</text>
        </enumeration>
    </constrain>
</parameter>
```

Schema

Additions and Changes

Schema Change: ContextParameter

```
<!-- Class for a context parameter -->
<xs:complexType name="ContextParameter">
    <xs:complexContent>
        <xs:extension base="CatalogItem">
            <xs:sequence>
                <xs:element name="type" type="ParameterType"/>
                <xs:element name="default" type="xs:anyType"/>
                <xs:element name=“constrain" type=“ConstraintType"/>
                <xs:element name=“validate" type=“ValidationType"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
    <xs:attribute name=“enable” type=“xs:string”/>
</xs:complexType>
```

Schema Addition: ConstraintType

```
<xs:complexType name="TextType">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute
        name="language“
        type="xs:string“
        default="eng"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:complexType name="EnumerationType">
  <xs:complexContent>
    <xs:sequence>
      <xs:element
        name="text“
        type="TextType"/>
    </xs:sequence>
```

```
    </xs:complexContent>
    <xs:attribute
      name="value“
      type="xs:anyType" use="required"/>
    <xs:attribute
      name="icon“
      type="s100Symbol:IdString"/>
  </xs:complexType>
```

```
<xs:complexType name="ConstraintType">
  <xs:complexContent>
    <xs:sequence>
      <xs:element
        name="enumeration“
        type="EnumerationType"/>
    </xs:sequence>
  </xs:complexContent>
</xs:complexType>
```

Schema Addition: ValidationType

```
<!-- Class for a validation error -->
<xs:complexType name="ValidationErrorType">
  <xs:complexContent>
    <xs:sequence>
      <xs:element
        name="text“
        type="TextType"/>
    </xs:sequence>
  </xs:complexContent>
  <xs:attribute
    name="enable“
    type="xs:string"/>
  <xs:attribute
    name="icon“
    type="s100Symbol:IdString"/>
</xs:complexType>
```

```
<!-- Class for a validation rule -->
<xs:complexType name="ValidationType">
  <xs:complexContent>
    <xs:sequence>
      <xs:choice>
        <xs:element
          name="xpath" type="xs:string"/>
        <xs:element
          name="regex" type="xs:string"/>
      </xs:choice>
      <xs:element
        name="errorMessage“
        type="ValidationErrorType"/>
    </xs:sequence>
  </xs:complexContent>
</xs:complexType>
```

Summary

- Supports validation of inputs to S-100 portrayal rules
  - Standards compliant
  - Supports best practices
  - Performant
  - Simple implementation
- Enhances portrayal robustness
  - “The ability to withstand or overcome adverse conditions or rigorous testing.”
- Enhances security
  - First line of defense against code injection techniques

Limitations

- Can’t use validation of one parameter to affect other parameters
  - Stateless – no defined priority / order of evaluation
    - Order of evaluation is implementation dependent
  - Duplicate validation rules in multiple parameters to achieve same effect
- Minimal support for context parameter type: *Date*
  - No support for S-100 truncated or recurring dates
  - XPath 1.0 does not support date comparison
    - Simple numeric comparisons can be supported
      - Validate to a specific format (e.g. YYYY-MM-DD), portrayal rule converts to a number
  - Note: date dependent portrayal does not require context parameter(s)
- No portrayal registry support (S100_PR_ContextParameter)
  - Because validation rules may be product specific, but context parameters are not

Actions

Prepare Change Forms

Present to Working Group