



ECHO Forms Specification

Version 0.7

REVISION HISTORY

Date	Version	Brief Description	Author
08/18/2006	0.1	Initial Version	Jason Gilman
08/29/06	0.2	Making updates from review and implementation.	Jason Gilman
10/04/06	0.3	Updated the forms and the autopopulate extension to the final namespaces for the v9 release.	Michael Pilone
03/23/07	0.4	Added the validation reference table in the appendix.	Michael Pilone
03/29/07	0.5	Added information about the limitations related to relevancy and pruning the instance.	Michael Pilone
11/27/07	0.6	Updated select control documentation.	Jason Gilman
12/02/08	0.7	Added new type for Output control to create a link.	Jason Gilman

1. INTRODUCTION.....	6
1.1. Technologies Used	6
1.1.1 XML	6
1.1.2 XPath	6
1.1.3 XML Schema.....	6
1.2. Language and Client Type Neutral	6
1.3. ECHO Forms Document Overview	7
1.3.1 Example ECHO Forms document.....	7
2. THE FORM	8
2.1. ECHO Form Namespace	9
2.2. Target Namespace.....	9
2.3. ECHO Forms Types.....	9
2.3.1 System Level ECHO Forms Lifecycle	9
2.4. Imports.....	10
3. MODEL	10
3.1. Instance	10
3.1.1 Namespaces in the Instance	10
3.2. Extension.....	11
3.2.1 Example	11
4. USER INTERFACE.....	12
4.1. Controls.....	12
4.1.1 Common Features and Components	13
4.2. Typed Controls.....	16
4.2.1 Common Features and Components	16
4.2.2 Input.....	16
4.2.3 Output	17
4.2.4 Select	18
4.2.5 Range	19
4.2.6 Secret	21
4.2.7 TextArea	21
4.3. Grouping Controls	22
4.3.1 Common Features and Components	22
4.3.2 Group.....	23
4.3.3 Repeat	23

4.4. Reference Controls.....	26
4.4.1 Common Features and Components	27
4.4.2 Controlref.....	27
4.4.3 Selectref.....	27
4.5. System Level ECHO Forms and Advanced Clients	28
4.5.1 Example of Advanced and Generic Client handling System Level ECHO Form	28
4.6. XPath Context	29
4.6.1 XML Example	30
4.6.2 XPath Formatting.....	30
4.6.3 Namespace Context	31
4.6.4 Context Node.....	31
5. CLIENT FORM HANDLING.....	32
5.1. Activity Diagrams	32
5.1.1 Load and Display ECHO Form.....	33
5.1.2 User Interaction (Real Time Client).....	34
5.1.3 User Interaction (Request-Response Client)	35
5.1.4 Common Stages	36
5.2. Design and Implementation Recommendations.....	37
5.2.1 Generate Types from the ECHO Form Schema	37
5.2.2 Create Business Objects to represent Controls	37
5.2.3 Use Visitor Pattern on Business Objects.....	37
6. AUTHORIZING ECHO FORMS GUIDELINES AND BEST PRACTICES.....	40
6.1. Model Guidelines.....	40
6.1.1 Write an XML Schema for the instance document.	40
6.1.2 Use version numbers in XML namespaces.	40
6.2. Control Guidelines	40
6.2.1 Use ECHO system level form controls as much as possible.	40
6.2.2 Limit the depth of groups and repeats.....	41
6.2.3 Use Ids in your controls	41
6.2.4 Use XPath and regular expression constraints where appropriate.	41
6.2.5 Use descriptive error messages in alerts.	41
6.2.6 Utilize help attributes on controls.....	41
6.2.7 Test forms in a client.	41
6.2.8 Cascade Dependencies.....	41
7. COMPLETE ECHO FORMS EXAMPLE.....	42
7.1. System Level ECHO Forms Example.....	42
7.2. Provider Level ECHO Forms Example.....	47
7.3. Sample GUI Screenshots.....	49
7.3.1 FTP Push and RangeLoc Selected	49
7.3.2 8MM Tape Selected.....	50
7.3.3 Temporal Subsetting Selected.....	51

7.3.4	Parameter Subsetting Selected	51
7.3.5	Polygon Subsetting Selected	52
7.4.	Generated Instance	54
8.	AUTOPOPULATE EXTENSION	55
8.1.	Autopopulate Example	55
8.2.	Autopopulate Elements.....	55
8.2.1	Expressions	55
8.2.2	Expression	55
8.3.	Processing Model	56
8.3.1	ECHO Processing Model.....	56
8.3.2	Client Processing Model.....	57
8.4.	ECHO Form Rules and Guidelines.....	57
9.	APPENDIX.....	58
9.1.	Validation Reference Table.....	58

1. INTRODUCTION

This document covers ECHO Forms. ECHO Forms is based on the W3C standard XForms 1.0 (<http://www.w3.org/TR/xforms/>). Much of this specification is derived from the concepts and wording of the XForms specification. ECHO Forms represents the next generation of Options. By splitting the Options into two parts – model and user interface it separates the structure of the information providers want to receive in orders from how that information is displayed in an ECHO client.

1.1. Technologies Used

In order to develop a client to ECHO or write an ECHO Forms document the author must be familiar with the following technologies.

1.1.1 XML

ECHO Forms documents are written in XML (<http://www.w3.org/XML/>). Clients must be able to parse an ECHO Forms document and create and modify the XML instance data sent in an ECHO Forms document. It is recommended that an advanced XML parser be used on the client like DOM (<http://www.w3.org/DOM/>). DOM is a platform- and language-neutral interface for allowing programs to access and update the structure of documents.

1.1.2 XPath

ECHO Forms relies heavily on XPath to bind the user interface to the instance data and to express constraints on the instance data. XPath is a W3C specification (<http://www.w3.org/TR/xpath>).

1.1.3 XML Schema

ECHO Forms includes some limited support for some of the built in types of XML Schema (<http://www.w3.org/TR/xmlschema-2/#built-in-datatypes>). Unlike XForms, ECHO Forms does not require clients to parse XML schema at runtime. There is a schema available for ECHO Forms which describes the structure of an ECHO Forms document.

1.2. Language and Client Type Neutral

ECHO Forms is a language and client type neutral specification. This means that a client to ECHO Forms can be written in any language that supports the technologies used. Any type of client is supported. The main target of ECHO Forms is considered to be a client with a GUI but a client could be written that worked in a console. Automated clients can be written to handle ECHO Forms as well. The only requirement is that the XML produced by a client is valid according for the ECHO Forms document.

1.3. ECHO Forms Document Overview

This section shows a simple example of an ECHO Forms document and describes the different parts of an ECHO Form.

1.3.1 Example ECHO Forms document

```
<?xml version="1.0" encoding="utf-8"?>
<form xmlns="http://echo.nasa.gov/v9/echoforms"
      targetNamespace="http://myorganization.gov/echoforms"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <model>
    <instance>
      <prov:options xmlns:prov="http://myorganization.gov/orderoptions">
        <prov:filename>data.txt</prov:filename>
        <prov:filesize>10</prov:filesize>
      </prov:options>
    </instance>
  </model>

  <ui>
    <input ref="prov:filename" type="xsd:string" label="File Name">
      <constraints>
        <constraint>
          <xpath>string-length(prov:filename) < 25</xpath>
          <alert>File names must be less than 25 characters</alert>
        </constraint>
        <constraint>
          <pattern>^[A-Za-z]+[A-Za-z0-9]*\.\?[A-Za-z0-9]*$</pattern>
          <alert>
            File names must start with a letter and
            not contain illegal characters
          </alert>
        </constraint>
      </constraints>
    </input>

    <range ref="prov:filesize" type="xsd:int" start="0"
           step="10" end="1000" label="File Size (MB)">
    </range>
  </ui>
</form>
```

An ECHO client would display the ECHO Form in a fashion similar to the following:



The screenshot shows a graphical user interface for an ECHO Form. It features two main input fields:

- File Name:** A text input field with the value "data.txt".
- File Size (MB):** A range input field with a value of "10". To the right of the input field, the following constraints are displayed:

Minimum:	0
Maximum:	1000
Step:	10

In the example above **form** has two parts, a **model** and a **ui** (which stands for user interface). The **model** contains an **instance**. The **instance** in an ECHO Form represents the structure of the XML that is the result of a client processing an ECHO Form. It also contains default values which the controls in the user interface are initialized with.

The user interface above contains a list of controls that are displayed to the user. The example user interface contains two controls, an **input** field and a **range**. You will notice some common things between the controls. They both have a **ref**, **type**, and a **label**. The **ref** is an XPath string that refers into the instance to the XML Node that holds the data the control is collecting. The **type** indicates what XML data type the control is collecting. The **label** contains a short string to describe the control.

Clients can use the type to determine how to display the field. For example if the type was `xsd:dateTime` a client may have shown a calendar widget versus using a text field for `xsd:string`. Clients can also use the type to validate the user input. If the type is `xsd:int` a client would know that user input of "red" would be invalid.

ECHO Forms can also validate input by using **constraints**. In the example above the input control has two constraints. A constraint contains an alert and either an XPath or pattern containing a regular expression. The alert is displayed to the user if the XPath evaluates to false or the regular expression does not match the input. The first constraint contains an XPath that checks that the filename entered is less than 25 characters. (Notice that the "<" is used instead of the less than sign because the XPath must be XML escaped (<http://www.w3.org/TR/REC-xml/#dt-escape>)) The second constraint example contains a regular expression pattern that checks that the file name starts with a letter, only contains letters and numbers, and may contain one period.

A client might display something like the following if the input failed the constraint.

Once a client has finished collecting information from the user it simply has to fill in the values the user entered into the instance XML and send it to ECHO. If the user changed none of the default values then the client would send the following to ECHO.

```
<prov:options xmlns:prov="http://myorganization.gov/orderoptions">
  <prov:filename>data.txt</prov:filename>
  <prov:filesize>10</prov:filesize>
</prov:options>
```

2. THE FORM

```
<form xmlns="http://echo.nasa.gov/v9/echoforms"
  targetNamespace="http://myorganization.gov/echoforms">
```

```
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

```
...
```

```
</form>
```

The form element is the root element of any ECHO Forms document.

2.1. ECHO Form Namespace

The form element is where the ECHO Forms namespace should be declared. The ECHO Forms namespace is `http://echo.nasa.gov/v9/echoforms`.

2.2. Target Namespace

The form element has one attribute called `targetNamespace`. This defines the target namespace of the form which is the namespace that all the user interface controls are in. This namespace is important when referring to controls imported from other ECHO Forms documents. See Reference Controls for more information on importing controls.

2.3. ECHO Forms Types

ECHO Forms documents can be created by either a system administrator or a provider. System level ECHO Forms define common models and user interfaces. The system level ECHO Forms can be imported in a provider level ECHO Form. Providers can use the system level ECHO Forms as building blocks to create their own forms. Providers can also import their other ECHO Forms but not forms created by other providers. This prevents a provider from breaking dependent provider's forms if they modify one of their own forms.

2.3.1 System Level ECHO Forms Lifecycle

1. System Form Created with new version number.
2. Provider's create dependent forms
3. New System Form added based on previous version
4. Providers transition to new version of form
5. Old version is deleted.

System level form changes are managed by versioning. When a system level form is created it is given a version as part of its namespace. Providers may then write other ECHO Forms that import that system form. The original system form is copied, modified, and added to ECHO as a new form with a different version number if a change needs to be made to it. Once providers have updated their forms to use the newest version the original one is deleted by Ops. This lifecycle prevents clients or providers from breaking by changing the basic definitions defined in the system ECHO Forms.

See 5 Client Form Handling for more information on client processing of ECHO System Forms.

2.4. Imports

```
<import name="StandardForm-v1.xml" scope="SYSTEM"/>
```

ECHO Forms Documents can import other ECHO Forms documents registered in ECHO. Import statements have two parts, a name and scope. The name indicates the file name of the ECHO Forms document. Scope is either SYSTEM or PROVIDER. The scope indicates whether the imported document is a system ECHO Form created by an administrator or an ECHO Form created previously by the provider.

3. MODEL

The model element contains the instance and allows for extensibility to ECHO Forms.

3.1. Instance

```
...  
<model>  
  <instance>  
    <prov:options xmlns:prov="http://myorganization.gov/orderoptions">  
      <prov:filename>data.txt</prov:filename>  
      <prov:filesize>10</prov:filesize>  
    </prov:options>  
  </instance>  
</model>  
...
```

The instance represents both a model of the data that is the result of an ECHO Form as well as initial values for the model. There should be only one element inside the instance. The element in the instance should be valid such that if it were copied into its own document it would represent a valid XML document. This means that the element should define all namespaces and prefixes required for it and its sub elements. See the next section for information on namespaces in the instance.

3.1.1 Namespaces in the Instance

The instance elements can be defined in any namespace other than the ECHO Forms namespace. XPath statements throughout the ECHO Forms document will need to reference elements in specific namespaces. All of the namespace prefixes used by XPath statements in the ECHO Forms document should be defined in the root element of the instances as a simplification to make client development easier. See the example below.

3.1.1.1 Namespace Example 1

```
...  
<model>  
  <instance>  
    <prov:options xmlns:prov="http://myorganization.gov/orderoptions">
```

```

        <prov:filename>data.txt</prov:filename>
        <prov:filesize>10</prov:filesize>
    </prov:options>
</instance>
</model>
<ui>
    <input ref="prov:filename" type="xsd:string" label="File Name"/>
</ui>

```

...

The section of the ECHO Form above defines a simple instance and one input. The input ref "prov:filename" is an XPath statement that refers to the filename element in the instance. "prov" is a namespace prefix that's defined in the options element of the instance. It's simpler for clients to determine what all the namespace prefixes are for XPaths throughout the document by defining all the namespace prefixes in the root element of the instance. Clients can ignore any other namespaces declared in the document for the purposes of XPath processing.

3.1.1.2 Namespace Example 2

...

```

<model>
  <instance>
    <prov:options xmlns:prov="http://myorganization.gov/orderoptions"
      xmlns:other="http://myorganization.gov/orderoptions">
      <prov:filename >data.txt</prov:filename>
      <prov:filesize>10</prov:filesize>
    </prov:options>
  </instance>
</model>
<ui>
  <input ref="other:filename" type="xsd:string" label="File Name"/>
</ui>

```

...

Two namespace prefixes, "prov" and "other", are defined in example 2. Notice that the URI of the namespaces, "<http://myorganization.gov/orderoptions>", is the same for both prefixes. The filename element is declared in the namespace identified by "prov" and the input refers to filename in a namespace identified by "other". Because "other" and "prov" actually refer to the same namespace this example is valid.

3.2. Extension

ECHO Forms is created such that it is extensible. Extensions are things that may be processed by a client or different parts of ECHO but may not apply to every use of ECHO Forms. An extension has a QName to identify it and can contain any XML not in the ECHO Forms namespace.

3.2.1 Example

This shows an example using the Autopopulate extension. See page 55, section 8 Autopopulate Extension for more information.

```

<model>
  <instance>

```

```
<prov:options xmlns:prov="http://myorganization.gov/orderoptions">
  <prov:filename>data.txt</prov:filename>
  <prov:filesize>10</prov:filesize>
  <prov:itemid/>
</prov:options>
</instance>

<extension name="auto:autopopulate"
  xmlns:auto="http://echo.nasa.gov/v9/echoforms/autopopulate">
  <auto:expressions>
    <auto:expression ref="prov:options/prov:granulesize"
      metadata="/results/provider/result/GranuleURMetaData/DataGranule/SizeMBD
ataGranule/text()" />
  </auto:expressions>
</extension>
</model>
...
```

4. USER INTERFACE

The user interface of ECHO Forms is described in the form using controls. Control types vary to support the collection of various types of information.

4.1. Controls

The controls are described in XML and rendered by a client application. Although this document suggests how the controls should be rendered, different clients may need to render the controls in different ways. For example, in an HTML based application, multiple HTML controls may need to be used to render a single ECHO Forms control such as a date input. For more interactive clients, such as a thick client .NET application, a complex date picker may be used.

The controls are associated to the instance document using XPath. The controls will be populated with the values in the default instance, and will update the instance as the data is modified in the control. Many of the controls support strictly typed data and may be rendered differently by the client based on the data type.

The controls in the user interface section are listed linearly with no layout information. A client must keep the order of the controls when rendering the form in order to maintain the semantic meaning behind the position of the controls. A client may decide to split the controls into multiple columns in a table, or in multiple dialogs. This behavior is acceptable, as long as related elements are visually associated.

4.1.1 Common Features and Components

The user interface (UI) controls share a number of common components such as attributes and child elements. If a control does not support a common component or the control applies a special meaning or restriction to the common component it will be detailed in the description of the control.

Computed expressions vs. fixed properties

- Fixed properties are static values that the XForms Processor evaluates only once. Such properties consist of literals, and are not subject to XPath evaluation.
- Computed expressions are XPath expressions that provide a value to the ECHO Forms processor. Such values are recalculated at certain times as specified by the ECHO Forms Processing Model (See 5 Client Form Handling). These expressions encode dynamic properties, often constraints, such as the dependency among various data items. Computed expressions are not restricted to examining the value of the instance data node to which they apply. XPath expressions provide the means to traverse the instance data. Unless otherwise stated, all XPaths used in the common components will be evaluated in the standard ECHO Forms control XPath context (see 4.5 XPath Context).

4.1.1.1 Ref Attribute

- Computed Expression: Yes
- Legal Values: any XPath expression that returns an element or attribute node
- Default Value: None
- Inheritance rules: See 4.5 XPath Context

The ref attribute is used to bind a control to a node in the instance document. If a control has a ref attribute, the XPath must be valid for the instance document in the form model. Therefore, all controls that have a ref in the form must have a node in the default model instance. The ref attribute is the key in linking the UI with the model in an ECHO Forms form.

4.1.1.2 Id Attribute

- Computed Expression: No
- Legal Values: anything of type xs:NCNAME
- Default Value: None
- Inheritance rules: Not inherited

The id attribute allows a control to have a specific name that it can be referenced by. This is useful for error messages or creating control references. (See 4.4 Reference Controls). The whole identifier for a control consists of its id and the targetNamespace of the form it's in.

4.1.1.3 Relevant

- Computed Expression: Yes
- Legal Values: any XPath expression that returns a boolean value

- Default Value: true()
- Inheritance rules: If the parent is not relevant then the children are not relevant as well.

The relevant attribute is used to indicate the relevance of a given control. If the control is relevant, the control should be displayed by the client and all other attributes should be evaluated (such as required and read only). If a control is not relevant, it should be hidden by the client but the instance should not be modified. When an instance document is submitted, any non-relevant nodes (the relevant XPath evaluates to false) must be removed from the instance. Due to a limitation in the specification, it is important to express all dependencies for a control when using a ref attribute. See 6.2.8 Cascade Dependencies for more information.

4.1.1.4 Required

- Computed Expression: Yes
- Legal Values: any XPath expression that returns a boolean value
- Default Value: false()
- Inheritance rules: Not inherited.

The required attribute indicates that the instance document must contain a value for the control before the form may be submitted. A value of false indicates that the instance may or may not contain a value for the control. Required is only evaluated if the control is relevant.

4.1.1.5 ReadOnly

- Computed Expression: Yes
- Legal Values: any XPath expression that returns a boolean value
- Default Value: false()
- Inheritance rules: If a parent is read only then the children are read only as well.

The readonly attribute indicates if the instance data is read only or mutable. If the attribute evaluates to true, the client must prevent the user from modifying the instance data. Note that because readonly is a computed expression, it may evaluate to true at one time and false another time. Therefore depending on the XPath used, one cannot guarantee that the instance data may not have change at some point in its lifespan. Obviously this would not be the case with a consistent XPath such as 'true()'.

Unlike relevant, readonly controls should be displayed to the user, however the user must not be able to edit the data.

Hint: To prevent a user from ever modifying part of the instance do not create a control that references that node. Use the Output control if the data needs to be displayed to the user but never modified. ReadOnly is best used if a control's read only status should change based on other values within the instance.

4.1.1.6 Label

- Computed Expression: No

- **Legal Values:** any xsd:string value. It is recommended that only a few words be used. New line characters are not permitted.
- **Default Value:** None
- **Inheritance rules:** Not inherited

A label attribute contains a string that will be used as the label of the control. Although the client can choose the best way to associate the label with the control, the label should be a limited number of words and is normally placed on the left or top of the data portion of the control.

4.1.1.7 Help

- **Computed Expression:** No
- **Legal Values:** any xsd:string. It is recommended that a single, short sentence be used.
- **Default Value:** None
- **Inheritance rules:** Not inherited.

A help child element is a simple text element that contains a string that will be used as the help of the control. Although the client can choose the best way to associate the help with the control, the help should be a single, short sentence and is normally rendered as a tool tip or status bar message for the data portion of the control.

4.1.1.8 Constraint

A constraint child element specifies a predicate that needs to be satisfied for the associated instance data node to be considered valid. A control can have any number of constraints which are evaluated in the order that they appear. Because all the constraints must be satisfied, multiple constraints are 'and-ed' together. A constraint contains either a regular expression pattern or an XPath expression and an alert message.

4.1.1.8.1 XPath Constraint

- **Computed Expression:** Yes
- **Legal Values:** any XPath expression that returns a boolean value
- **Default Value:** None

A xpath child element is a an XPath expression that must evaluate to true to satisfy the constraint. The XPath may contain new line and tab characters to support readability. New line and tab characters should be replaces with a space character before evaluating the XPath.

4.1.1.8.2 Pattern Constraint

- **Computed Expression:** No
- **Legal Values:** any regular expression compatible with Perl 5
- **Default Value:** None

A pattern child element is a regular expression that must match the node data to satisfy the constraint. The pattern will be applied to the un-encoded string value of the data node. Therefore XML elements in the value will be un-escaped, however if a pattern is applied to a

xsd:dateTime for example, the string will represent the schema encoded dateTime. For more information on regular expressions, please consult <http://www.regular-expressions.info/tutorial.html>.

4.1.1.9 Alert

- Computed Expression: No
- Legal Values: any xsd:string. It is recommended that a single, short sentence be used.
- Default Value: None

A alert child element is a message that will be displayed to the user if the constraint is not satisfied. This message should contain a description of how the user can satisfy the constraint. For example, a password constraint alert may say, "Your password must be at least 8 characters and contain numbers and letter." It is recommended that the client display this message in a noticeable means such as using bright colors or a notification dialog.

4.2. Typed Controls

Typed controls are strongly typed using XML Schema types. These types indicate the format as well as the valid values that may occur in the instance document.

4.2.1 Common Features and Components

4.2.1.1 Type

- Computed Expression: No
- Legal Values: a QName of any supported XML Schema type
- Default Value: None

The type attribute indicates the type of the data in the instance document. Clients can also use the type information to customize the control by rendering it differently for each supported type. The type attribute is required on all typed controls unless otherwise noted by the control.

4.2.2 Input

- Types Supported: string, double, dateTime, long, int, short, boolean

The input control allows free-form input from the user. The client should attempt to provide an accurate rendering of the control for the given data type. For example, for a boolean type a checkbox should be used. For a dateTime type, a date picker should be used.

4.2.2.1 Common Component Variations

None

4.2.2.2 Specific Components

None

4.2.2.3 XML Example

```
<input type="xsd:string" relevant="true()"
  required="true()" ref="po:name" readonly="false()"
  label="First and Last Name"/>
```

4.2.2.4 Client Rendering Example



First and Last Name
John Doe

4.2.3 Output

- Types Supported: string, double, dateTime, long, int, short, boolean, anyURI

This form control renders a value from the instance data, but provides no means for entering or changing data. It is used to display values from the instance. The output control can be used to display the value at a particular location in the instance by using a binding expression; it can also be used to display the result of evaluating an XPath expression by specifying the XPath expression to be evaluated via attribute value instead of ref. Note that attributes ref and value on element output are mutually exclusive. It is recommended when the type of "anyURI" is used that the client display the component as a clickable link.

4.2.3.1 Common Component Variations

The ref attribute is mutually exclusive with the value attribute.

The readonly attribute is ignored.

The required attribute is ignored.

4.2.3.2 Specific Components

Value

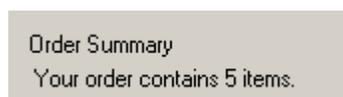
- Computed Expression: Yes
- Legal Values: any XPath expression that returns a string value
- Default Value: None

The value attribute contains an XPath expression that will be evaluated. The resulting string of the evaluation will be displayed. This attribute is mutually exclusive with the ref attribute. If the value attribute is used, the type attribute must be type string.

4.2.3.3 XML Example

```
<output type="xsd:string" relevant="true()"
  value="concat(string('Your order contains '), 5, string(' items.'))"
  label="Order Summary"/>
```

4.2.3.4 Client Rendering Example



Order Summary
Your order contains 5 items.

4.2.3.5 anyURI XML Example

```
<output type="xsd:anyURI"
  value="'http://www.nasa.gov'"
  label="NASA"/>
```

4.2.3.6 anyURI Client Rendering Example



4.2.4 Select

- Types Supported: string, double, dateTime, long, int, short

This form control allows the user to make one or multiple selections from a set of choices. A select control uses an attribute to indicate if it is a single or multiple select. A select may also be open, which means that a user may type in a value and select it. In the case of an open select, the items in the instance which indicate selection may not be limited to the items in the original list of choices.

4.2.4.1 Common Component Variations

The ref attribute will refer to the root element which will hold the selected value elements. The name of the selected value elements is indicated with a specific attribute.

The required attribute indicates that at least one item must be selected. If the required attribute is false, an instance document with no selected items is valid. A constraint may be used to enforce a minimum number of selected items (for example, enforcing the selection of at least three items in a selection).

4.2.4.2 Specific Components

4.2.4.2.1 Open

- Computed Expression: No
- Legal Values: 'true' or 'false'
- Default Value: false

The value attribute contains a Boolean value which indicates if the selection is open to user input. If the selection is open, the client must allow the user to enter values into the selection and select those new values. The editor for the value input should be type specific (for example, a date picker for the dateTime type). If the selection is not open, the client must ensure that only items in the list of choices are selected. If the user does not enter or select a value then there should be no value elements in the instance document for this select.

4.2.4.2.2 ValueElementName

- Computed Expression: No
- Legal Values: any valid XML Schema NCName
- Default Value: None

The `valueElementName` attribute contains an `NCName` which is the name of the child element of the `ref` element that will contain the selected values. This element should only occur in the instance document to provide default selections. If there is no default value for the select then the select element in the instance should not contain any children. The client application will add and remove instances of this element as the selection changes. The element will contain the value text of the selected item or items.

4.2.4.2.3 Multiple

- Computed Expression: No
- Legal Values: 'true' or 'false'
- Default Value: false

The `multiple` attribute contains a Boolean value which indicates if the selection allows multiple items to be selected. If multiple selections are supported, the client must allow the user to select more than one item in the list of choices.

4.2.4.2.4 Item

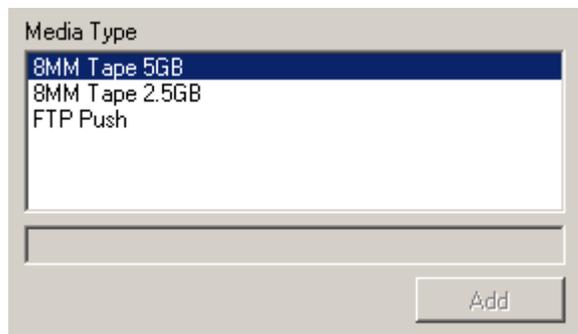
- Computed Expression: No

The `item` child element defines the valid items that may occur in the selection. Each `item` element contains a `value` and `label` child elements. The `label` will be displayed to the user by the client application for each item. The `value` is the value that will occur in the model instance if the item is selected. `Value` must be of length greater than or equal to 1.

4.2.4.3 XML Example

```
<select ref="po:medias" type="xsd:string" multiple="false"
  open="false" valueElementName="media" label="Media Type">
  <item label="8MM Tape 5GB" value="TAPE8MM_5GB"/>
  <item label="8MM Tape 2.5GB" value="TAPE8MM_2AND1HALFGGB"/>
  <item label="FTP Push" value="FTPPUSH"/>
</select>
```

4.2.4.4 Client Rendering Example



The screenshot shows a window titled "Media Type" containing a list box. The list box has three items: "8MM Tape 5GB", "8MM Tape 2.5GB", and "FTP Push". The "8MM Tape 5GB" item is selected and highlighted in blue. Below the list box is a text input field and an "Add" button.

4.2.5 Range

- Types Supported: double, dateTime, long, int, short

This form control allows selection from a sequential range of values. A range control has a minimum value, maximum value, and a step size. The selected value must be between the minimum and maximum and occur on a valid step increment. A client will normally render a range control as a slider, spinner, or up-down box. The client must use the type information in the range control to render the proper controls to the user. The user should be told of the minimum, maximum, and step size.

4.2.5.1 Common Component Variations

None

4.2.5.2 Specific Components

4.2.5.2.1 Start

- Computed Expression: No
- Legal Values: any valid value in the type of the range
- Default Value: None

The start attribute contains the minimum value of the range. This value is inclusive and must be the same type as the range control.

4.2.5.2.2 End

- Computed Expression: No
- Legal Values: any valid value in the type of the range
- Default Value: None

The end attribute contains the maximum value of the range. This value is inclusive and must be the same type as the range control.

4.2.5.2.3 Step

- Computed Expression: No
- Legal Values: any valid positive value in the type of the range except in the case of a `dateTime`, in which case the valid value is any positive integer
- Default Value: None

The step attribute contains the amount that the range will move as the value is adjusted. The user is required to pick a value which occurs on a step based on the start value of the range. The client should attempt to force the user onto the step values to limit user error. The step value must be smaller than $(end - start)$. In the case of a `dateTime` range type, the step value indicates the number of seconds.

4.2.5.3 XML Example

```
<range ref="po:lowersearchdate" start="2000-06-15T01:15:23Z"  
  end="2010-06-15T01:15:23Z" step="5"  
  type="xsd:dateTime" label="Lower Search Date"/>
```

```
<range ref="po:acceptableerror" start="42"  
  end="110" step="2.12344444"  
  type="xsd:double" label="Acceptable error"/>
```

4.2.5.4 Client Rendering Example

Lower Search Date

Thursday , June 15, 2000 01:15:23 AM

Minimum: Thursday, June 15, 2000 01:15:23 AM
Maximum: Tuesday, June 15, 2010 01:15:23 AM
Step: 5 Seconds

Acceptable error: 42.000000000000
Minimum: 42
Maximum: 110
Step: 2.12344444

4.2.6 Secret

- Types Supported: string, double, long, int, short

The secret control allows free-form input from the user. This control is identical to the input control with the following exceptions: it does not support the boolean or date types, user input must be obscured using an echo character such as a '*'. The secret control is normally used like an input control but for sensitive data such as user names or passwords.

4.2.6.1 Common Component Variations

None

4.2.6.2 Specific Components

None

4.2.6.3 XML Example

```
<secret ref="po:password" type="xsd:string" label="Access Password"/>
```

4.2.6.4 Client Rendering Example

Access Password

4.2.7 TextArea

- Types Supported: string

The textarea control allows free-form input from the user. This control is similar to the input control, except that it allows for multi-line input. The textarea control only supports the string type.

4.2.7.1 Common Component Variations

None

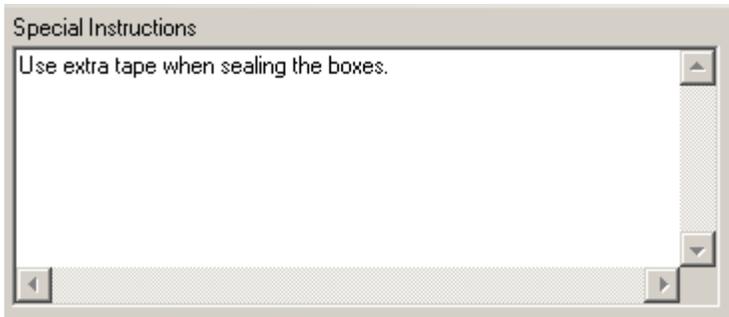
4.2.7.2 Specific Components

None

4.2.7.3 XML Example

```
<textarea ref="po:specialInstructions" type="xsd:string" required="true()" label="Special Instructions"/>
```

4.2.7.4 Client Rendering Example



4.3. Grouping Controls

Grouping controls are used to combine together controls to form logical and visually grouped elements. Grouping controls can be nested.

4.3.1 Common Features and Components

This section lists some common features and components for grouping controls

4.3.1.1 Relevance

Relevant, as defined in section 4.1.1.3, indicates whether a control is relevant to be displayed in the GUI or sent in the XML. All the children of a grouping control are not relevant if the parent control is not relevant. The children of a grouping control MAY be relevant according to their individual relevant attributes if the parent control is relevant.

4.3.1.2 Ref Attribute

The Ref Attribute, as defined in section 4.1.1.1, defined the node in the instance document that the control was bound to. Grouping controls can be bound to the instance document as well. The Ref attributes of child controls are relative to the Ref of the grouping control. See section 4.6 XPath Context.

4.3.1.3 Required

Required does not get inherited in grouping controls. Whether or not a grouping control is required has no bearing on the required state of the children. This essentially means that the required field on a grouping control is ignored since there is no user collected data for a grouping control besides its children.

4.3.1.4 Read Only

The ReadOnly attribute is inherited by children of a grouping control. If the grouping control is read only then all of the children are read only as well. The children of a grouping control MAY be read only according to their individual attributes if the parent control is not read only.

4.3.2 Group

A group combines together other controls to show they belong together. A group is often displayed as a bounding box with a label. This shows the user that the elements of the group are part of a single unit.

4.3.2.1 Common Component Variations

The ref attribute is optional for a group. It can not reference an attribute. If it is used then the rules defined in 4.6 XPath Context applies.

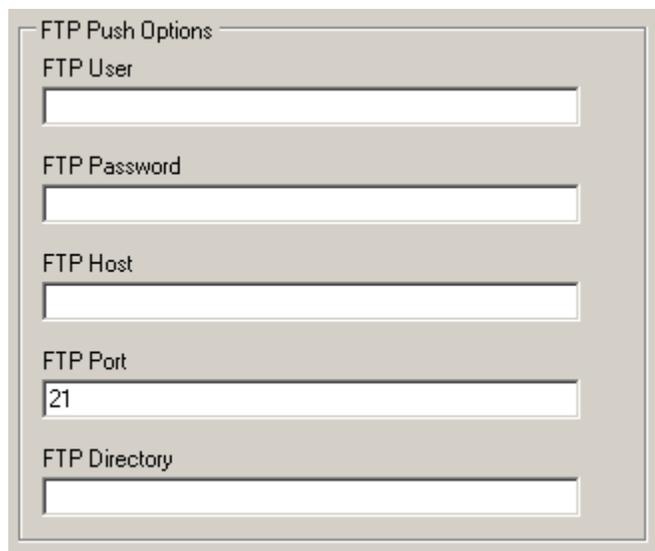
4.3.2.2 Specific Components

None

4.3.2.3 XML Example

```
<group ref="po:ftppush" id="ftpPushGroup" label="FTP Push Options">
  <input ref="po:user" type="xsd:string" label="FTP User"/>
  <secret ref="po:password" type="xsd:string" label="FTP Password"/>
  <input ref="po:host" type="xsd:string" label="FTP Host"/>
  <input ref="po:port" type="xsd:int" label="FTP Port"/>
  <input ref="po:directory" type="xsd:string" label="FTP Directory"/>
</group>
```

4.3.2.4 Client Rendering Example



The image shows a client-rendered form for the 'FTP Push Options' group. The group is enclosed in a light gray border with a title bar at the top. Inside the group, there are five input fields, each with a label above it: 'FTP User', 'FTP Password', 'FTP Host', 'FTP Port', and 'FTP Directory'. The 'FTP Port' field contains the value '21'. The 'FTP Password' field is a secret input, indicated by a small icon in the top left corner of the field.

4.3.3 Repeat

A repeat allows the definition of repeating structures such as points within a polygon. As grouping controls repeats can be nested to define multiple levels of repetition such as many polygons being entered each with many points.

Repeat may be the most complex control for clients to handle. Clients must be able to allow the user to add or remove the repeated items.

4.3.3.1 Common Component Variations

The ref attribute is can not reference an attribute for a repeat. Ref in a repeat should reference the element to which the elements in the template should be added.

4.3.3.2 Specific Components

4.3.3.2.1 template

- Computed Expression: No
- Legal Values: Any XML in namespace other than ECHO Forms namespace
- Default Value: None

Template is an element in repeat that is similar to instance in the model in that it can contain any XML and defines a model for the controls in the repeat. The template is used as a template for creating new elements to add as children to the node that ref points to. Template can contain only one root element. All of the children of a repeat reference a node in the repeats template.

4.3.3.2.2 minItems

- Computed Expression: No
- Legal Values: Any valid XML Schema int
- Default Value: 0

The minItems attribute contains the minimum number of items that may appear in the repeat. The items refer to the instances of the template. The minItems attribute can be used to force a lower limit number of instances in the repeat before it is considered valid. It is recommended that if the repeat contains the minimum number of items, a client should not allow the user to remove any more items. Setting the minItems and maxItems to the same value creates a closed repeat where items cannot be added or removed, but simply edited. The model instance in the form must have enough items in the repeat to satisfy this constraint.

4.3.3.2.3 maxItems

- Computed Expression: No
- Legal Values: Any valid XML Schema int
- Default Value: Max integer

The maxItems attribute contains the maximum number of items that may appear in the repeat. The items refer to the instances of the template. The maxItems attribute can be used to force an upper limit number of instances in the repeat before it is considered invalid. It is recommended that if the repeat contains the maximum number of items, a client should not allow the user to add any more items. Setting the minItems and maxItems to the same value creates a closed repeat where items cannot be added or removed, but simply edited. The model instance in the form must have fewer items in the repeat to satisfy this constraint.

4.3.3.3 XML Example

This example shows two repeats nested. This repeat allows many polygons each with many points to be created. The constraint in the inner repeat requires a certain number of repeated points.

```

...
<model>
  <instance>
    <spatial:options xmlns:spatial="http://nasa.gov/spatial">
      <spatial:polygons>
        <spatial:polygon>
          <spatial:points>
            <spatial:point>
              <spatial:x>0</spatial:x>
              <spatial:y>0</spatial:y>
            </spatial:point>
            <spatial:point>
              <spatial:x>0</spatial:x>
              <spatial:y>0</spatial:y>
            </spatial:point>
            <spatial:point>
              <spatial:x>0</spatial:x>
              <spatial:y>0</spatial:y>
            </spatial:point>
          </spatial:points>
        </spatial:polygon>
      </spatial:polygons>
    </spatial:options>
  </instance>
</model>
...
<repeat ref="spatial:polygons" minItems="0" maxItems="200" label="Polygons">
  <template>
    <polygon xmlns="http://nasa.gov/spatial">
      <points>
        <point>
          <x>0</x>
          <y>0</y>
        </point>
        <point>
          <x>0</x>
          <y>0</y>
        </point>
        <point>
          <x>0</x>
          <y>0</y>
        </point>
      </points>
    </polygon>
  </template>

  <repeat ref="spatial:points" minItems="3" maxItems="5" label="Point">
    <template>
      <point xmlns="http://nasa.gov/spatial">
        <x>0</x>
        <y>0</y>
      </point>
    </template>

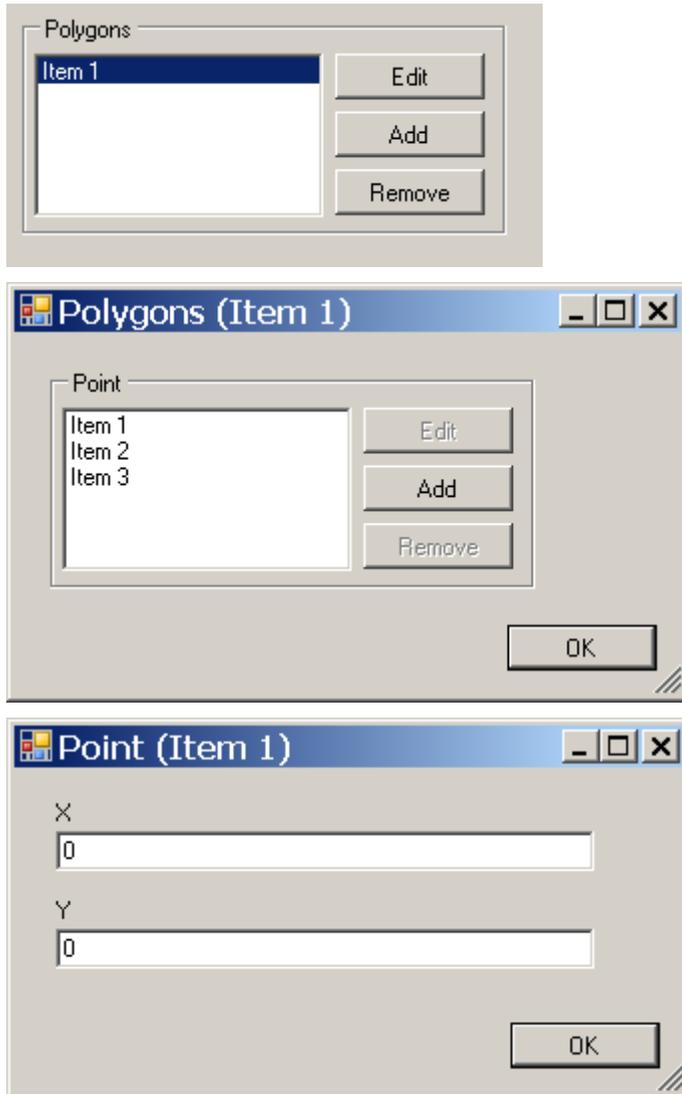
    <input ref="spatial:x" type="xsd:int" label="X"/>

```

```
<input ref="spatial:y" type="xsd:int" label="Y"/>
</repeat>
</repeat>
```

4.3.3.4 Client Rendering Example

This client example shows editing each item of a repeat in a separate window.



4.4. Reference Controls

Reference controls are used to refer to existing, common control definitions in the same or other ECHO Forms documents. Using reference controls, it is possible to define a control once, and then reuse that control in multiple locations. Controls are referenced through their id attribute which is a qualified name. Reference controls may not refer to other reference controls.

4.4.1 Common Features and Components

4.4.1.1 Idref

- Computed Expression: No
- Legal Values: any QName which refers to a defined control id
- Default Value: None

The idref attribute is a QName which indicates the control to reference. The control referenced must be defined with the given id.

4.4.2 Controlref

The controlref reference control indicates that the control referenced should replace this reference control in the form. Any attributes defined in the controlref element will replace those defined in the referenced control. All constraints defined in a controlref element will be appended to those of the referenced control. A controlref element may reference any control with an id attribute. If the control referenced is in a separate form, the other form must be imported using the standard ECHO Forms import element.

4.4.2.1 Common Component Variations

Common component attributes used in a controlref will replace those of the control referenced. In this way, it is possible to replace the relevant, required, label, and ref components of the referenced control. If a controlref does not define one of the common control attributes, the value defined on the referenced control should be used.

4.4.2.2 Specific Components

None

4.4.2.3 XML Example

```
<controlref ref="po:ftppush" idref="echo:ftpPushGroup"
  required="true()">
  <constraints>
    <constraint>
      <xpath>string-length(password) > 4</xpath>
      <alert>Your password must be longer than 4 characters</alert>
    </constraint>
  </constraints>
</controlref>
```

4.4.3 Selectref

The selectref reference control indicates that the control referenced should replace this reference control in the form. The selectref element behaves identically to controlref except that it can only refer to select controls. A selectref element can limit the number of items available for selection in the referenced select control. Therefore, using a selectref, it is

possible to share common selections even if all the users of the select control do not support all the items in the original select.

If a selectref is required to contain all the items in the referenced select control, it is recommended that a controlref be used instead of repeating all the items in the referenced select.

4.4.3.1 Common Component Variations

Selectref has the same variations as controlref (see **Error! Reference source not found.**).

4.4.3.2 Specific Components

4.4.3.2.1 Item

- Computed Expression: No

The item child element defines the valid items that may occur in the selection. Each item element contains a value and label child elements. The label will be displayed to the user by the client application for each item. The value is the value that will occur in the model instance if the item is selected. All of the values specified must occur in the selection referenced. Therefore it is possible to only restrict a selection, but not to add new values to a referenced selection. If an item does not have a label, the label defined in the referenced select control item should be used.

4.4.3.3 XML Example

```
<selectref ref="po:medias" idref="system:mediaTypeSelect">
  <item label="8MM Tape 5GB" value="TAPE8MM_5GB"/>
  <item label="FTP Push" value="FTPPUSH"/>
</selectref>
```

4.5. System Level ECHO Forms and Advanced Clients

ECHO Forms defines a way to write user interface that clients can handle generically at runtime. It saves clients from having to code specific GUIs to each provider's schema. This is a powerful feature that allows for changes in provider definitions to be handled dynamically by clients. While powerful this can sometimes lead to "generic looking" clients. System Level ECHO Forms allow clients to provide a better user experience.

As defined in section 2.3.1 System Level ECHO Forms Lifecycle, a system level ECHO Form will contain basic types that providers can reuse in their own forms. Once a system level ECHO Form has been created it will never be changed. Only new versions of it will be added. Advanced clients can be coded to use a different user interface that what is detailed in the system level form as long as the resulting XML produced is the same.

4.5.1 Example of Advanced and Generic Client handling System Level ECHO Form

4.5.1.1 System Level Form

A system level form defines a group of inputs for a bounding box. The bounding box has an upper left point and a lower right point. This would be 4 input controls in a single group control, 2 input controls being for each point.

```

...
<group ref="spatial:boundingbox" id="boundingBox" label="Bounding Box">
  <input ref="spatial:ullat" type="xsd:double" label="Upper Left Latitude"/>
  <input ref="spatial:ullon" type="xsd:double" label="Upper Left Longitude"/>
  <input ref="spatial:lrlat" type="xsd:double" label="Lower Right Latitude"/>
  <input ref="spatial:lrlon" type="xsd:double" label="Lower Right Longitude"/>
</group>

```

...

4.5.1.2 Provider's Form

A provider defines a form that imports the system level form. They also define a control reference to the bounding box.

```

...
<controlref ref="prov:myboundingbox" idref="echo:boundingBox"
  label="Bounding box for spatial subsetting"/>
...

```

...

4.5.1.3 How a Generic Client handles provider form

A generic client would handle the provider form by first locating the system level control referenced in the provider form. It would then display the group with the 4 input boxes and labels. A user would see the 4 input boxes and go get their map.

4.5.1.4 How an Advanced Client handles the provider form

An advanced client would load the provider form. It would then check the control references to see if it had any specialized controls for the controlref's in the provider form. The advanced client finds that it has a specialized widget that's associated with the id referenced by the control ref. Instead of displaying the group with 4 input boxes the advanced client will display its specialized bounding box map widget.

4.5.1.5 Differences in user experience

The differences in user experience is obvious once you realize that the user using the generic client has to get out a map to find the correct points and the generic client will lack the validation that can be done with the advanced client. Because the advanced client knows that its looking at a bounding box instead of a group and 4 input boxes it can check things like making sure the area is correct for the granule being subset.

The advanced client provides fewer errors from passing through to the provider and less work for the user.

4.6. XPath Context

The XPath context defines the context in which all XPaths in the form are executed. The context definition is essential to make the forms behave correctly and to ensure that the author of the form has the same understanding of the context as the client application rendering the form.

The XPath context is composed of two parts:

1. The list of available namespaces and the corresponding aliases defined for the namespaces

2. The node in the XML tree that is currently being processed

The client application is responsible for establishing the correct XPath context before executing any XPath expression.

4.6.1 XML Example

The following example will be used to explain the XPath context:

```
<?xml version="1.0" encoding="utf-8"?>
<form xmlns="http://echo.nasa.gov/v9/echoforms"
      targetNamespace=" http://myForm/v1"
      xmlns:myForm=" http://myForm/v1"
      xmlns:echo="http://echo.nasa.gov/echoforms/controls/v1"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <model>
    <instance>
      <test xmlns="http://myForm/v1" xmlns:myForm="http://myForm/v1"
          xmlns:echo="http://echo.nasa.gov/echoforms/controls/v1">
        <name>Bob Smith</name>
        <latBox>
          <upperleftlat/>
          <lowerrightlat/>
        </latBox>
        <lonBox>
          <upperleftlon/>
          <lowerrightlon/>
        </lonBox>
        <media>TAPE8MM_5GB</media>
      </test>
    </instance>
  </model>

  <ui>
    <input ref="myForm:name" type="xsd:string" label="Name"/>

    <group ref="myForm:latBox" label="Lat Box">
      <input ref="myForm:upperleftlat" type="xsd:double"
        label="Upper Left Latitude"/>
    </group>

    <group label="Lon Box">
      <input ref="myForm:lonBox/myForm:upperleftlon" type="xsd:double"
        label="Upper Left Longitude"/>
    </group>
  </ui>

</form>
```

4.6.2 XPath Formatting

To increase the readability and to simplify the authoring of forms, XPaths used in ECHO Forms (such as in constraints) may span multiple lines and contain TAB characters. Some XPath processors may have trouble evaluating XPaths which contain new line or TAB characters, therefore the client is responsible for replacing these characters with a single

space character before evaluation. Note that if an XPath is used in an attribute, it cannot span multiple lines per the XML specification.

4.6.3 Namespace Context

The namespace context for all XPaths used in an ECHO form is the same. The context is composed of all namespaces and prefixes defined on the root element of the model instance document. For example, in the above ECHO Forms form, the namespace context would contain all the namespaces and prefixes defined on the <test> element. So the namespace map would be:

- [default ns] -> http://myForm/v1
- myForm -> http://myForm/v1
- echo -> http://echo.nasa.gov/echoforms/controls/v1

These prefixes will be available in any XPaths used in the form definition. Note that the namespaces may also need to be defined on the root form element if the prefix is used in the XML document outside of the instance (for example, in a control reference idref attribute).

4.6.4 Context Node

The context node defines the current node in the XML tree that is being processed. A context node is used as the root of all relative XPath expressions. It is recommended that all XPath expressions in controls be relative to the context node. This is extremely important when defining controls that may be referenced from other forms. If an absolute XPath is used, the referenced control would dictate the structure of the form referencing the control which is not desirable.

By default, the context node is always the root node of the instance document. Therefore, all root level controls (controls which are children of the ui element) have a context node of the root node of the document. A control is relative to the context node of its closest ancestor that defines a ref node. For example, if a control is in a container control, such as a group or repeat, and the group or repeat contains a ref attribute, the child control is relative to the group or repeat node.

In the example form above, the name input control is using a relative XPath to refer to the name element. The absolute XPath for this control is

```
{http://myForm/v1}test/{http://myForm/v1}name.
```

The input in the latitude group is using a relative XPath as well, however because the containing group control defines a ref node, the input is relative to this node. Therefore the absolute XPath for this control is

```
{http://myForm/v1}test/{http://myForm/v1}latBox/{http://myForm/v1}upperleftlat.
```

The input in the longitude group is using a relative XPath as well, however because the containing group control does not define a ref node, the input is relative to the root of the instance. Therefore the absolute XPath for this control is

```
{http://myForm/v1}test/{http://myForm/v1}lonBox/{http://myForm/v1}upperleftlon.
```

Although XPath expressions are always evaluated relative to this context node, it is possible to do absolute XPath expressions using the standard XPath syntax or using one of the many XPath axis components. Absolute XPath expressions may be useful when defining constraints, required, or relevant attributes. A client must support executing any valid XPath in the ECHO Forms form and should not assume that all XPath expressions are relative or absolute.

5. CLIENT FORM HANDLING

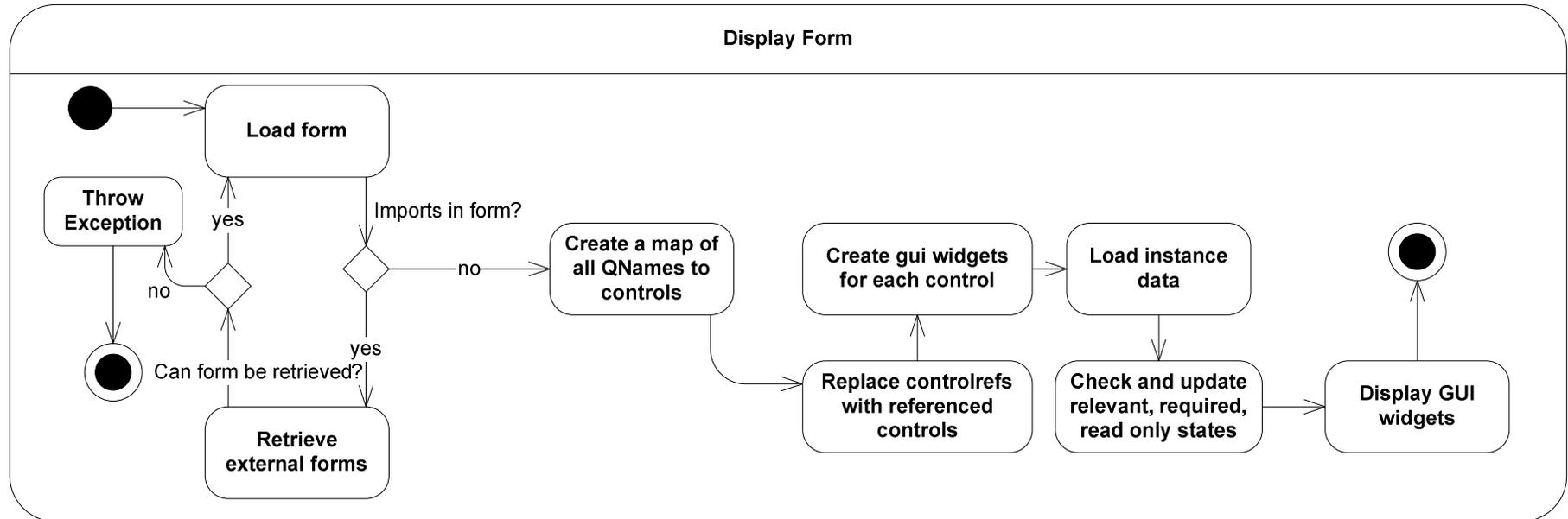
The section explains best practices for handling and processing an ECHO Forms document.

5.1. Activity Diagrams

This section includes 3 activity diagrams for handling an ECHO Form document.

5.1.1 Load and Display ECHO Form

This activity diagram shows how a client would load an ECHO form document and display it to the user.



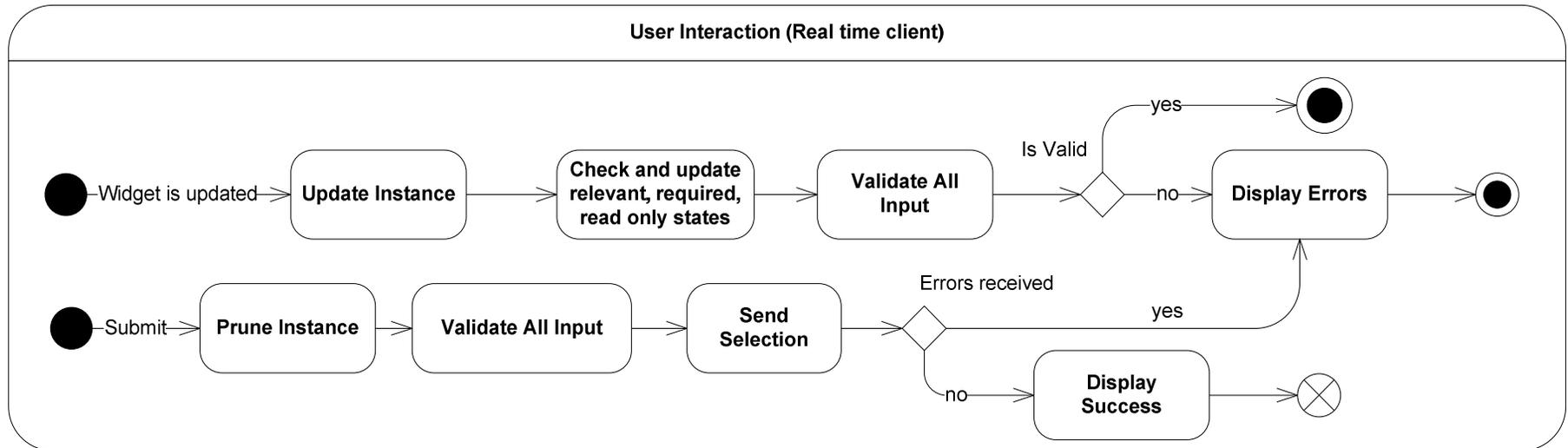
5.1.1.1 Stages Explanation

- Load Form
 - The client will receive an ECHO Form as part of another object from ECHO. It should attempt to load the form here.
- Retrieve External Forms
 - If the form includes imports the other ECHO Forms can be retrieved from ECHO
- Create a map of QNames to Controls
 - QName means Qualified Name and is an XML term. A qualified name usually includes a prefix, namespace, and a localname. In this case QName refers to the qualified names of controls. The control's id is its local name. The namespace is the target namespace of the ECHO Form that the control is in.
 - In order to find the controls by Id a map needs to be made of the QNames to the controls so they can be looked up in later steps.
- Replace controlrefs with Referenced Controls
 - In this step the map created in the prior step is used to find the referenced controls.
 - The controlrefs should be replaced with a copy of the controls they reference.
 - Controlrefs can override some fields of the referenced controls so the copies should be updated with overridden values.
- Create GUI Widgets for each Control
 - For each control in the form a GUI widget of the correct type should be created.
 - A GUI Widget is some component in the client language that can display a field for the user to enter information.
 - If a client has created advanced widgets for specific system form definitions they should be created here if the control has the correct id. (See 4.5 System Level ECHO Forms and Advanced Clients)
- Load Instance Data
 - The GUI Widgets should be updated with the values from the instance.
- Display GUI Widgets
 - The GUI Widgets should be displayed
 - le (open window with GUI Widgets, create html from widgets, etc)

5.1.2 User Interaction (Real Time Client)

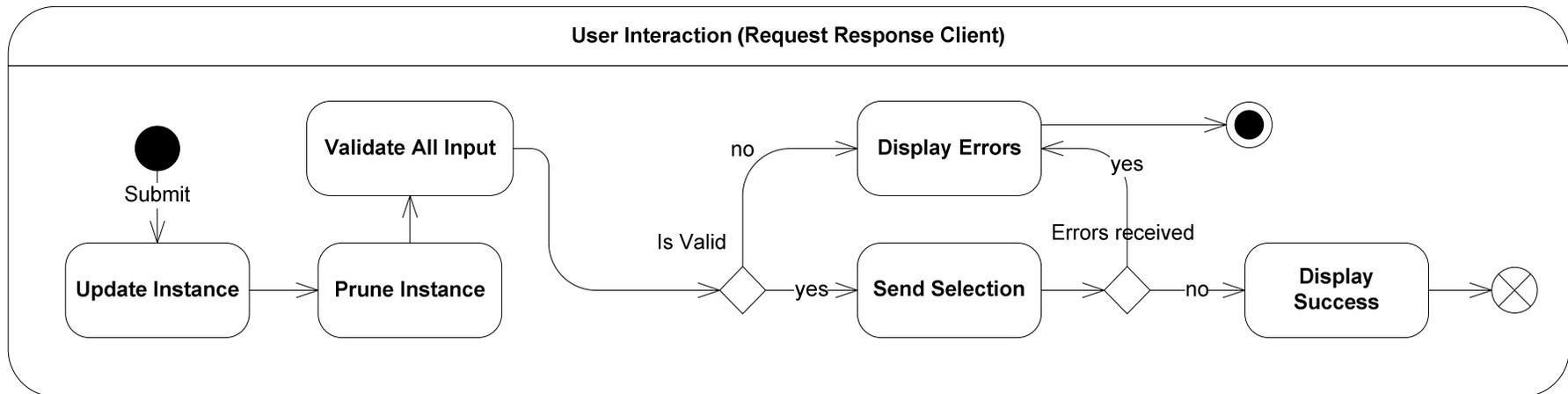
This activity diagram shows how a real time client can respond to interaction from the user updating a field and submitting the form.

5.1.2.1 Widget Updated and Submit Diagram



5.1.3 User Interaction (Request-Response Client)

This activity diagram shows how a client that can not validate the controls as the user enters the information.



5.1.4 Common Stages

These are stages that are common among the activity diagrams

- Check and Update Relevant, Required, and Read Only States
 - If a control is not relevant the GUI widgets should not be displayed
 - The widget may need to know if it is a required field or not
 - Read only widgets should be displayed but not be editable by the user.
- Update Instance
 - The instance should be updated with the values the user entered
- Validate All Input
 - This is optional as ECHO will always update the selections received from clients
 - A better client experience can be obtained by validating on the client
 - All the controls should be validated to check that the user input was valid
- Prune Instance
 - Remove the nodes from the instance that are not relevant. Relevancy is determined by the relevant attributed on controls. (See page 13, section 4.1.1.3 Relevant.) Only those nodes that are explicitly not relevant as indicated by the control should be pruned. Some nodes may exist in the form but not have controls associated with them. These nodes should be sent as part of the selection. The pruning should be performed against the full instance document (not the one that is being generated as a result of pruning). Because of the relevancy design, pruning should only need to be performed in a single pass (no looping). Due to a limitation in the specification, it is possible that a pruned instance does not validate while the non-pruned instance did. Section 6.2.8 Cascade Dependencies describes how form authors should work around the limitation.
- Send Selection
 - Send the instance xml to ECHO
- Display Errors
 - Errors should be displayed next to the GUI Widgets that were invalid
- Display Success
 - Display a success message to the user indicating ECHO accepted the submission

5.2. Design and Implementation Recommendations

This section lists recommendations for client implementation. These recommendations mostly only apply to clients written in object oriented languages.

5.2.1 Generate Types from the ECHO Form Schema

Different languages have tools available to generate types from XML Schema. Generating types gives you a set of objects that can be unmarshalled from ECHO Form schema to work with instead of parsing raw XML. It is easier to deal with an object called EchoForm versus parsing text.

5.2.2 Create Business Objects to represent Controls

Business objects allow you to create an abstraction for an ECHO Forms. Generated types give you a good baseline to convert into business objects. Generated types can not be modified in code usually and you may want to represent the object hierarchy differently than the tool generated them. Having business objects allows you to create the exact object hierarchy you prefer and to add any methods onto the business objects that you may need.

5.2.3 Use Visitor Pattern on Business Objects

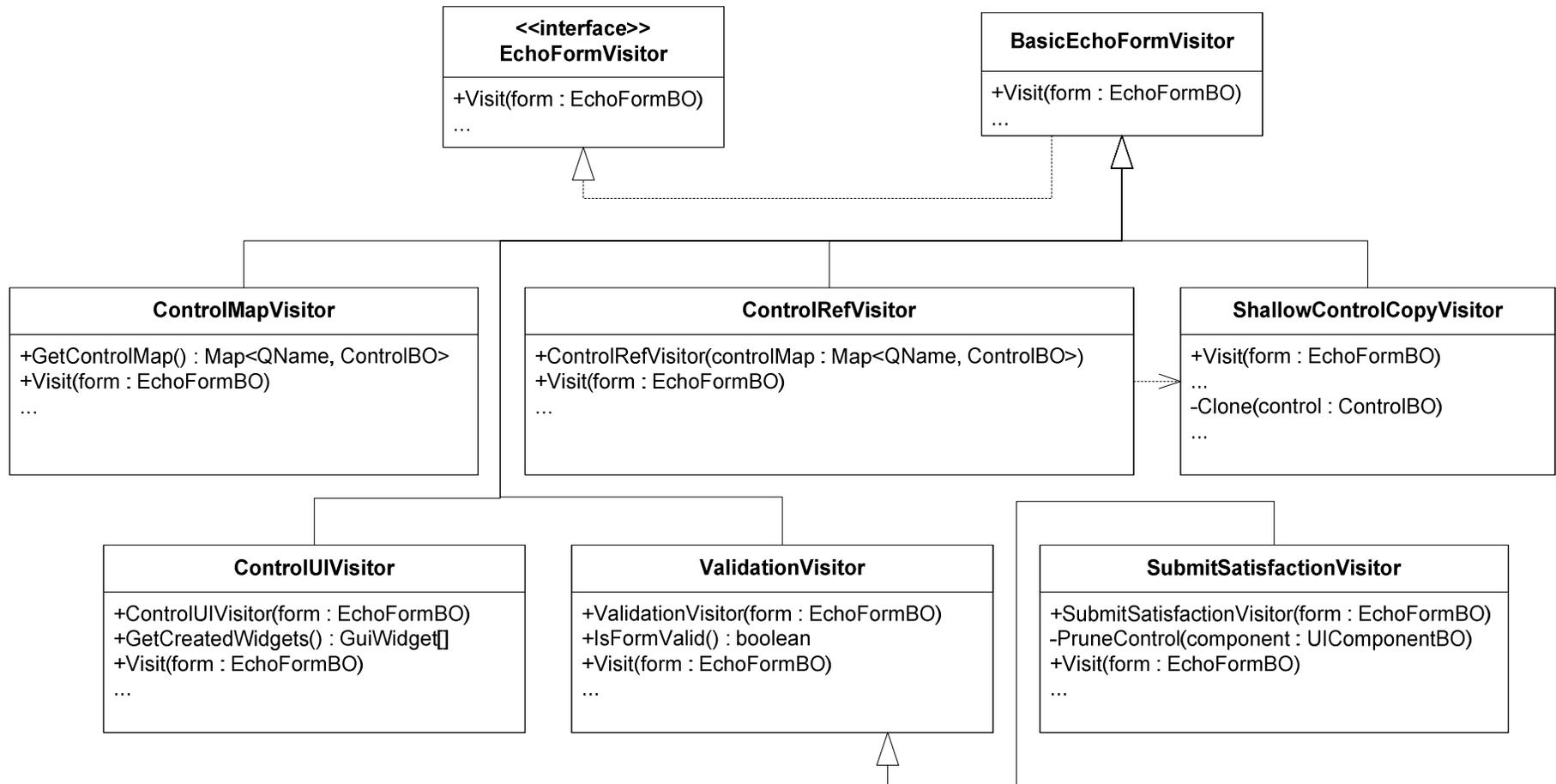
The visitor pattern, as documented in Gamma et al., allows you to represent an operation to be performed on the business objects as a single class without having to check the type of business object and cast it.

- BasicEchoFormVisitor
 - This is the base class for all visitors. It implements the traversal of the controls by calling accept on them.
- ControlMapVisitor
 - This visits the controls and creates a map of the Qualified Name to control bos
- ControlRefVisitor
 - This visitor uses the map to replace the controlrefs with the real referenced controls. It uses the ShallowControlCopyConstructor to clone the original nodes.
- ShallowControlCopyVisitor
 - This can create a copy of controls. It creates a shallow copy as in it only duplicates the first layer. Any child controls of the original are also the same children of the new duplicate node.
- ControlUIVisitor
 - This visitor creates GUI Widgets for each control business object it meets
- ValidationVisitor

- This visitor validates all of the controls to make sure that the values in the instance conform to the constraints and specific rules of each control.
- SubmitSatisfactionVisitor
 - This visitor prepares the instance for submission. It validates all the controls and prunes any controls which are not needed because they are not relevant.

5.2.3.1 Suggested Visitor Class Diagram

This is a class diagram of some suggested visitors for a client.



6. AUTHORIZING ECHO FORMS GUIDELINES AND BEST PRACTICES

This section contains a list of guidelines and best practices for writing ECHO Forms. These are not requirements but they lead to a better user experience, less errors, easier post processing of the selection XML.

6.1. Model Guidelines

These are guidelines for writing the Model and Instance portions of an ECHO Forms document.

6.1.1 Write an XML Schema for the instance document.

XML Schema is the good way to describe what an XML document. It is well defined and is supported by many tools. Business objects can be generated from a schema to process XML selections from an ECHO Instance. In the future ECHO Forms may support XML schema as an extension to allow ECHO and clients to perform further checks on the selection XML.

6.1.2 Use version numbers in XML namespaces.

XML namespaces provide a good way to identify the elements of your instance and ECHO Forms. As the controls of your form change or the structure of your instance a way is needed to indicate which version of a form or instance a client or provider is receiving. A version number can be included in an XML namespace to version it. The complete example uses this and there are several online resources that provide information about this best practice. An article from IBM is located here <http://www-128.ibm.com/developerworks/xml/library/x-tipnamsp.html>.

6.2. Control Guidelines

6.2.1 Use ECHO system level form controls as much as possible.

Use ECHO system level form controls allows clients to display more advanced controls and improve the user experience. (See section 4.5 System Level ECHO Forms and Advanced Clients.)

6.2.2 Limit the depth of groups and repeats.

As the depth of groups and repeats increase so does the apparent complexity of the GUI for the user. Try to limit groups to about 5 levels and repeats to 2 or 3 levels to keep the user interface from looking to complex.

6.2.3 Use Ids in your controls

See page 13, section 4.1.1.2 for more information on the Id Attribute. This will give better error messages from ECHO that clients can handle and it allows reuse of that control through control references. (See page 26, section 4.4 Reference Controls)

6.2.4 Use XPath and regular expression constraints where appropriate.

See page 15, section 4.1.1.8 Constraint. This will ensure that the data collected by the client is valid.

6.2.5 Use descriptive error messages in alerts.

This will help users determine what they need to do to correct errors. See page 16, section 4.1.1.9 Alert.

6.2.6 Utilize help attributes on controls.

This will help user's understand what they are supposed to input. See page 15, section 4.1.1.7 Help.

6.2.7 Test forms in a client.

A newly created form should be tested in a client to ensure that the XML it creates is correct and that the user interface is usable.

6.2.8 Cascade Dependencies

Due to a specification limitation in the validation and relevancy checking required for clients that process ECHO Forms, it is important that relevancy and required attributes on controls are defined in such a way that they include all dependencies. This

requires treating each control as independent and not expecting cascading dependencies. For example, if control A depends on control B and control C depends on control B, control C implicitly has a dependency on both A and B. This dependency must be expressed when using relevancy tags on control C. If the dependencies are not properly expressed, a client may incorrectly display a control to the user and submit an invalid selection. A simple workaround for these situations is to use a group control to logically group controls that have related dependencies. This limitation may be addressed in a future version of the specification.

7. COMPLETE ECHO FORMS EXAMPLE

This section contains a complete ECHO Forms document and screenshots from a sample client GUI.

7.1. System Level ECHO Forms Example

This is a full example of a system level ECHO Forms document.

```
<?xml version="1.0" encoding="utf-8"?>
<form xmlns="http://echo.nasa.gov/v9/echoforms"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://echo.nasa.gov/echoforms/systemforms/v1">

  <model>
    <instance>
      <systemoptions xmlns="http://echo.nasa.gov/echoforms/systemforms/v1"
                    xmlns:echo="http://echo.nasa.gov/echoforms/systemforms/v1">
        <mediaoptions>
          <mediatype/>
          <ftppush>
            <user/>
            <password/>
            <host/>
            <port>21</port>
            <directory/>
          </ftppush>
          <compression/>
        </mediaoptions>
        <subsetoptions>
          <subsettype/>
          <rangeloc>
            <east_longitude>0.0</east_longitude>
          </rangeloc>
        </subsetoptions>
      </instance>
    </model>
  </form>
```

```

        <north_latitude>0.0</north_latitude>
        <south_latitude>0.0</south_latitude>
        <west_longitude>0.0</west_longitude>
    </rangeloc>
    <temporal>2000-01-01T00:00:00</temporal>
    <parameter/>
    <polygons/>
</suboptions>
</systemoptions>
</instance>
</model>
<ui>
  <group id="mediaOptionsGroup" ref="echo:mediaoptions" label="Media Options">
    <select id="mediaTypeSelect" ref="echo:mediatype"
      multiple="false" open="false" valueElementName="value"
      type="xsd:string" label="Media Type">
      <item label="8MM Tape 5GB" value="TAPE8MM_5GB"/>
      <item label="8MM Tape 2.5GB" value="TAPE8MM_2AND1HALFGB"/>
      <item label="FTP Push" value="FTPPUSH"/>
      <item label="FTP Pull" value="FTPPULL"/>
      <item label="Compact Disc (CDROM)" value="CDROM"/>
      <item label="Digital Versatile Disc (DVD)" value="DVD"/>
      <item label="Digital Linear Tape (DLT) 20GB" value="DLT_20GB"/>
      <item label="Digital Linear Tape (DLT) 30GB" value="DLT_30GB"/>
    </select>

    <group id="ftpPushGroup" ref="echo:ftppush" relevant="echo:mediatype/echo:value = 'FTPPUSH'"
      label="FTP Push Fields">

      <output type="xsd:anyURI" value="'http://www.echo.nasa.gov/ftppush.html'"
        label="FTP Push Information" />

      <input ref="echo:user" type="xsd:string" label="FTP User"/>

      <secret ref="echo:password" type="xsd:string" label="FTP Password"/>

      <input ref="echo:host" type="xsd:string" label="FTP Host"/>

      <input ref="echo:port" type="xsd:int" label="FTP Port">
        <help>The port used when making an FTP connection to the server.</help>
      </input>

      <input ref="echo:directory" type="xsd:string" label="FTP Directory"/>
    </group>
  </ui>

```

```

<select id="compressionSelect" ref="echo:compression" open="false"
  multiple="false" type="xsd:string" valueElementName="value"
  relevant="echo:mediatype/echo:value = 'FTPPULL'"
  label="FTP Compression Type">
  <item label="Gzipped Tape Archive Format (TARGZ)" value="TARGZ"/>
  <item label="Bzipped Tape Archive Format (TARBZ)" value="TARBZ"/>
  <item label="PK/Win Zip Format" value="WINZIP"/>
</select>
</group>

<group id="subsetOptionsGroup" ref="echo:subsoptions" label="Subsetting Options">
  <select id="subsetTypeSelect" ref="echo:subsettype" multiple="false"
    open="false" valueElementName="value" type="xsd:string" label="Subset Type">
    <item label="Range Loc" value="RANGELOC"/>
    <item label="Temporal" value="TEMPORAL"/>
    <item label="Parameter" value="PARAMETER"/>
    <item label="Spatial Polygon" value="POLYGON"/>
  </select>

  <!-- parameter subsetting -->
  <input id="parameterInput" ref="echo:parameter" type="xsd:string"
    relevant="echo:subsettype/echo:value = 'PARAMETER'" label="Parameter">
    <constraints>
      <constraint>
        <pattern>^[A-Za-z]+</pattern>
        <alert>Parameter names must start with a letter</alert>
      </constraint>
      <constraint>
        <pattern>^[A-Za-z0-9]+$</pattern>
        <alert>Parameter names can only contain letters and numbers</alert>
      </constraint>
    </constraints>
    <help>Enter a parameter to subset by</help>
  </input>

  <!-- RangeLOC subsetting -->
  <group id="rangeLocGroup" ref="echo:rangeloc"
    relevant="echo:subsettype/echo:value = 'RANGELOC'" label="Range Loc Spatial Fields">
    <input type="xsd:double" ref="echo:east_longitude" label="East Longitude">
      <constraints>
        <constraint>
          <xpath>(echo:east_longitude > -180) and (echo:east_longitude < 180)</xpath>
          <alert>The east longitude must be between -180 and 180.</alert>
        </constraint>
      </constraints>
    </input>
  </group>

```

```
        </constraint>
    </constraints>
    <help>This longitude defines the eastern border of the bounding box</help>
</input>

<input type="xsd:double" ref="echo:west_longitude" label="West Longitude">
    <constraints>
        <constraint>
            <xpath>(echo:west_longitude > -180) and (echo:west_longitude < 180)</xpath>
            <alert>The west longitude must be between -180 and 180.</alert>
        </constraint>
    </constraints>
    <help>This longitude defines the western border of the bounding box</help>
</input>

<input type="xsd:double" ref="echo:north_latitude" label="North Latitude">
    <constraints>
        <constraint>
            <xpath>(echo:north_latitude > -90) and (echo:north_latitude < 90)</xpath>
            <alert>The North Latitude must be between -90 and 90.</alert>
        </constraint>
    </constraints>
    <help>This latitude defines the northern border of the bounding box</help>
</input>

<input type="xsd:double" ref="echo:south_latitude" label="South Latitude">
    <constraints>
        <constraint>
            <xpath>(echo:south_latitude > -90) and (echo:south_latitude < 90)</xpath>
            <alert>The South Latitude must be between -90 and 90.</alert>
        </constraint>
    </constraints>
    <help>This latitude defines the southern border of the bounding box</help>
</input>
</group>

<!-- temporal subsetting -->
<input id="temporalInput" ref="echo:temporal" type="xsd:dateTime"
    relevant="echo:subsettype/echo:value = 'TEMPORAL'" label="Temporal">
    <help>Enter a date to subset by</help>
</input>

<!-- polygon subsetting -->
<repeat id="polygonsRepeat" ref="echo:polygons"
```

```

    relevant="echo:subsettype/echo:value = 'POLYGON'" label="Polygons Spatial Subset">
<template>
  <polygon xmlns="http://echo.nasa.gov/echoforms/systemforms/v1">
    <points>
      <point>
        <latitude>0.0</latitude>
        <longitude>0.0</longitude>
      </point>
      <point>
        <latitude>0.0</latitude>
        <longitude>0.0</longitude>
      </point>
      <point>
        <latitude>0.0</latitude>
        <longitude>0.0</longitude>
      </point>
    </points>
  </polygon>
</template>

<output value="count(..//echo:polygon)" label="Current Number of Polygons" type="xsd:int"/>

<output value="count(../*/*/*echo:point)" label="Total Points" type="xsd:int"/>

<repeat ref="echo:points" label="Points">
  <constraints>
    <constraint>
      <xpath>count(echo:points/echo:point) >= 3</xpath>
      <alert>A polygon requires 3 points</alert>
    </constraint>
  </constraints>
  <help>Choose at least three longitude and latitude points to create a polygon to subset by.</help>
  <template>
    <point xmlns="http://echo.nasa.gov/echoforms/systemforms/v1">
      <latitude>0.0</latitude>
      <longitude>0.0</longitude>
    </point>
  </template>

  <input ref="echo:latitude" type="xsd:double" label="Latitude">
    <constraints>
      <constraint>
        <xpath>(echo:latitude > -90) and (echo:latitude < 90)</xpath>
        <alert>Latitude must be between -90 and 90</alert>
      </constraint>
    </constraints>
  </input>

```

```

        </constraint>
      </constraints>
    </input>

    <input ref="echo:longitude" type="xsd:double" label="Longitude">
      <constraints>
        <constraint>
          <xpath>(echo:longitude > -180) and (echo:longitude < 180)</xpath>
          <alert>Longitude must be between -180 and 180.</alert>
        </constraint>
      </constraints>
    </input>
  </repeat>
</repeat>
</group>
</ui>
</form>

```

7.2. Provider Level ECHO Forms Example

This is an example of a provider's ECHO Forms document. It imports the system form example and uses some of the controls defined there.

```

<?xml version="1.0" encoding="utf-8"?>
<form xmlns="http://echo.nasa.gov/v9/echoforms"
      xmlns:echo="http://echo.nasa.gov/echoforms/systemforms/v1"
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://provider.nasa.gov/echoforms/myoptions/v1">
  <import name="SystemFormExample"/>
  <model>
    <instance>
      <myoptions xmlns="http://provider.nasa.gov/orderoptions"
                xmlns:prov="http://provider.nasa.gov/orderoptions"
                xmlns:sys="http://echo.nasa.gov/echoforms/systemforms/v1">
        <mediatype>
          <value>FTPPUSH</value>
        </mediatype>
        <ftppush>
          <sys:user/>
          <sys:password/>
          <sys:host/>
        </ftppush>
      </myoptions>
    </instance>
  </model>
</form>

```

```

        <sys:port>21</sys:port>
        <sys:directory/>
    </ftppush>
    <subset>
        <sys:subsettype/>
        <sys:rangeloc>
            <sys:east_longitude>0.0</sys:east_longitude>
            <sys:north_latitude>0.0</sys:north_latitude>
            <sys:south_latitude>0.0</sys:south_latitude>
            <sys:west_longitude>0.0</sys:west_longitude>
        </sys:rangeloc>
        <sys:temporal>2000-01-01T00:00:00</sys:temporal>
        <sys:parameter/>
        <sys:polygons/>
    </subset>
</myoptions>
</instance>
</model>
<ui>
    <group label="Media Options">
        <selectref idref="echo:mediaTypeSelect" label="Offered Media Delivery Types" ref="prov:mediatype">
            <item value="FTPPUSH"/>
            <item value="TAPE8MM_5GB"/>
            <item value="TAPE8MM_2AND1HALFGB"/>
        </selectref>
        <controlref idref="echo:ftpPushGroup" ref="prov:ftppush" relevant="prov:mediatype/prov:value='FTPPUSH'">
            <constraints>
                <constraint>
                    <xpath>prov:ftppush/sys:port = '21' or prov:ftppush/sys:port = '1821'</xpath>
                    <alert>We can only do FTP Push to ports 21 and 1821.</alert>
                </constraint>
            </constraints>
        </controlref>
    </group>
    <controlref idref="echo:subsetOptionsGroup" ref="prov:subset">
        <constraints>
            <constraint>
                <xpath>count (prov:subset/sys:polygons/sys:polygon) &lt;= 1</xpath>
                <alert>We only subsetting by a single polygon.</alert>
            </constraint>
        </constraints>
    </controlref>
</ui>
</form>

```

7.3. Sample GUI Screenshots

These are some screenshots from a GUI displaying the provider form.

7.3.1 FTP Push and RangeLoc Selected

This image shows the viewer with FTP Push media type and RangeLoc subsetting selected.

The screenshot displays the 'Echo Forms Viewer' application window. The interface is divided into several sections:

- File**: A menu bar at the top.
- Media Options**: A section titled 'Offered Media Delivery Types' containing a list box with 'FTP Push' selected. Below the list box is an 'Add' button.
- Subsetting Options**: A section containing a list box with 'Range Loc' selected. Below the list box is an 'Add' button.
- FTP Push Fields**: A section with the following fields:
 - [FTP Push Information](#) (blue link)
 - FTP User:
 - FTP Password:
 - FTP Host:
 - FTP Port:
 - FTP Directory:
- Range Loc Spatial Fields**: A section with the following fields:
 - East Longitude:
 - West Longitude:
 - North Latitude:
 - South Latitude:

7.3.2 8MM Tape Selected

This image shows 8MM Tape 5GB selected. Notice that the FTP Push fields are hidden because they are not relevant.

The screenshot shows the 'Echo Forms Viewer' application window. It features a menu bar with 'File' and two main panels. The left panel, titled 'Media Options', contains a section 'Offered Media Delivery Types' with a list box containing 'FTP Push', '8MM Tape 5GB' (which is selected), and '8MM Tape 2.5GB'. Below this list is an 'Add' button. The right panel, titled 'Subsetting Options', contains a 'Range Loc' list box with 'Temporal', 'Parameter', and 'Spatial Polygon' options. Below this is another 'Add' button. At the bottom of the right panel is a section titled 'Range Loc Spatial Fields' with four input fields: 'East Longitude' (120.0), 'West Longitude' (119.0), 'North Latitude' (50.0), and 'South Latitude' (49.0).

7.3.3 Temporal Subsetting Selected

This image shows Temporal subsetting selected. A calendar widget is used for the input field.

The screenshot shows the 'Echo Forms Viewer' interface. On the left, under 'Media Options', the 'Offered Media Delivery Types' list includes 'FTP Push', '8MM Tape 5GB' (highlighted), and '8MM Tape 2.5GB'. An 'Add' button is at the bottom. On the right, under 'Subsetting Options', the list includes 'Range Loc', 'Temporal' (highlighted), 'Parameter', and 'Spatial Polygon'. An 'Add' button is at the bottom. Below the subsetting options, a 'Temporal' dropdown menu is set to 'Saturday , January 01, 2000 12:00:00 AM'.

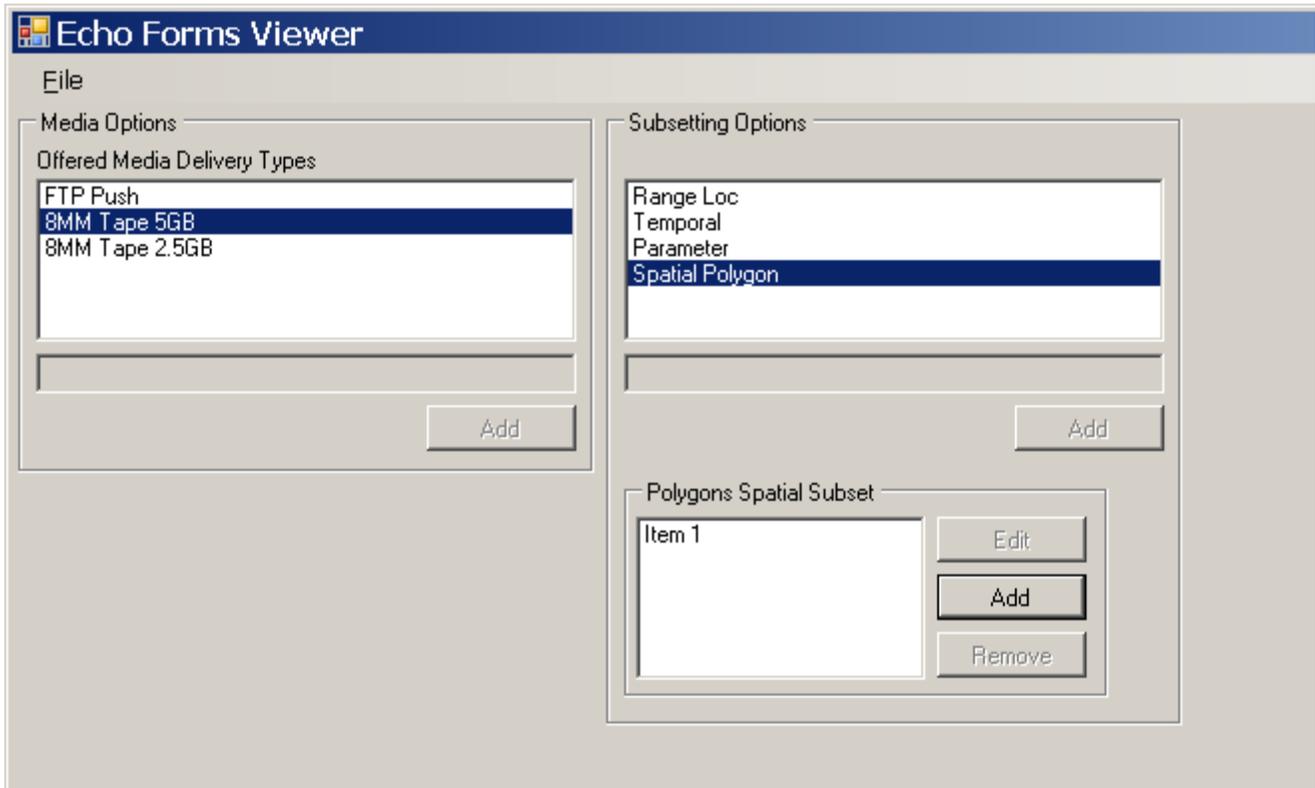
7.3.4 Parameter Subsetting Selected

This image shows Parameter subsetting selected. A regular string text box is used for the parameter field.

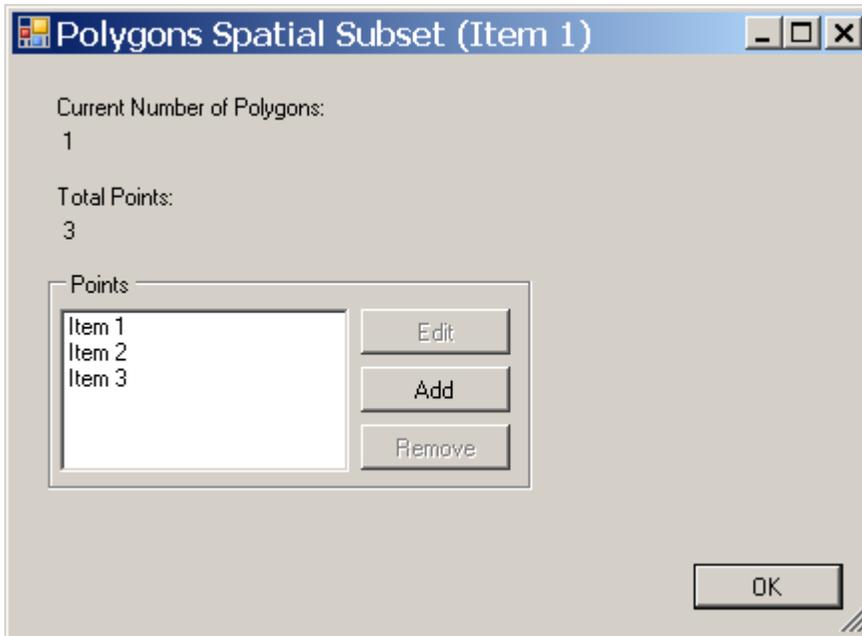
The screenshot shows the 'Echo Forms Viewer' interface. On the left, under 'Media Options', the 'Offered Media Delivery Types' list includes 'FTP Push', '8MM Tape 5GB' (highlighted), and '8MM Tape 2.5GB'. An 'Add' button is at the bottom. On the right, under 'Subsetting Options', the list includes 'Range Loc', 'Temporal', 'Parameter' (highlighted), and 'Spatial Polygon'. An 'Add' button is at the bottom. Below the subsetting options, a 'Parameter' text box contains the text 'someParameterName'.

7.3.5 Polygon Subsetting Selected

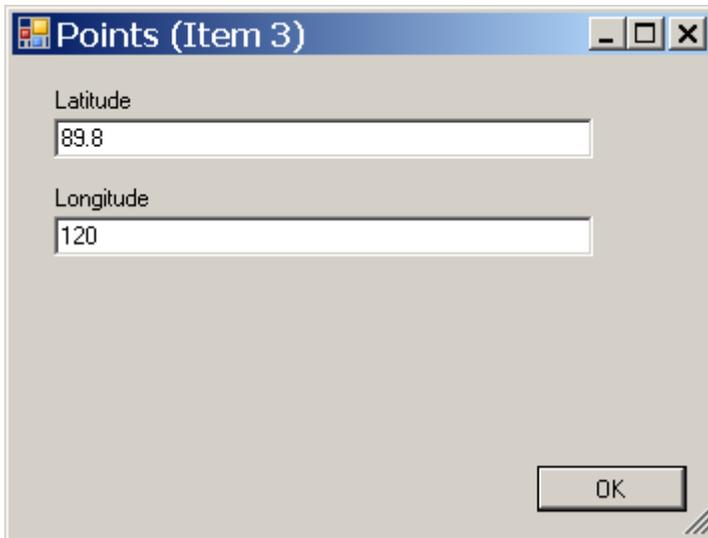
This image shows Spatial Polygon selected. Repeats are handled in this client by putting the controls in the repeat in a new window.



This image shows the window for editing a polygon



This image shows the window for editing a point



7.4. Generated Instance

This shows a sample generated instance for the provider form if the user had selected FTP Push and Polygon subsetting.

```
<myoptions xmlns="http:provider.nasa.gov/orderoptions"
  xmlns:prov="http:provider.nasa.gov/orderoptions"
  xmlns:echo="http://echo.nasa.gov/echoforms/systemforms/v1">
  <mediatype>
    <value>FTPPUSH</value>
  </mediatype>
  <ftppush>
    <echo:user>user</echo:user>
    <echo:password>secretpassword</echo:password>
    <echo:host>user.ftp.nasa.gov</echo:host>
    <echo:port>21</echo:port>
    <echo:directory>/tmp/user/data</echo:directory>
  </ftppush>
  <subset>
    <echo:subsettype>
      <echo:value>POLYGON</echo:value>
    </echo:subsettype>
    <echo:polygons>
      <polygon xmlns="http://echo.nasa.gov/echoforms/systemforms/v1">
        <points>
          <point>
            <latitude>45.7</latitude>
            <longitude>34.8</longitude>
          </point>
          <point>
            <latitude>29.6</latitude>
            <longitude>45.9</longitude>
          </point>
          <point>
            <latitude>89.8</latitude>
            <longitude>120</longitude>
          </point>
        </points>
      </polygon>
    </echo:polygons>
  </subset>
</myoptions>
```

8. AUTOPOPULATE EXTENSION

This section details the Autopopulate extension to ECHO Forms. See section 3.2 Extension for more information on extending ECHO Forms. Autopopulate is a capability that allows XPath strings into the metadata of a collection or granule to put into the instance XML.

8.1. Autopopulate Example

The autopopulate extension defines an extension tag that looks like the following.

```

...
<instance>
  <options xmlns="http://provider" xmlns:prov="http://provider">
    ...
    <granulesize/>
    ...
  </options>
</instance>

<extension name="auto:autopopulate"
  xmlns:auto="http://echo.nasa.gov/v9/echoforms/autopopulate">
  <auto:expressions>
    <auto:expression ref="/prov:options/prov:granulesize"
      metadata="/results/provider/result/GranuleURMeta-dataGranule/text()" />
  </auto:expressions>
</extension>
...

```

8.2. Autopopulate Elements

This section contains a description of the elements of the autopopulate extension.

8.2.1 Expressions

The root of the extension is the expressions element. This contains a list of one or more autopopulate expressions to evaluate.

8.2.1.1 XML Example

```

<auto:expressions>
...
</auto:expressions>

```

8.2.2 Expression

An expression element represents one autopopulate XPath to evaluate. The expression element must occur in an expressions element.

8.2.2.1 Ref Attribute

- Computed Expression: Yes

- Legal Values: any XPath expression that returns an element or attribute node
- Default Value: None
- Inheritance rules: See 4.5 XPath Context

The ref attribute indicates which node in the instance the resolved metadata elements will be added to. In the example below it points to the granulesize element in the instance.

8.2.2.2 Metadata Attribute

- Computed Expression: Yes
- Legal Values: any XPath expression into the catalog item metadata
- Default Value: None
- Inheritance rules: No inheritance

Metadata is an XPath expression that will be resolved against the metadata of a collection or granule. It can refer to a node, node set, or string value.

8.2.2.3 XML Example

```
<auto:expression ref="/prov:options/prov:granulesize"
  metadata="/results/provider/result/GranuleURMetaData/DataGranule/SizeMBData
  Granule/text()" />
```

8.3. Processing Model

This section describes the responsibilities of a client and ECHO when processing an ECHO Forms document with the Autopopulate Extension.

8.3.1 ECHO Processing Model

8.3.1.1 ECHO Forms Document Created

When the form is created ECHO will validate that the ECHO Forms document in the following manner.

1. ECHO validates the form according to ECHO Forms specification.
2. ECHO validates any autopopulate expressions
 - a. ECHO checks that the ref XPath is valid and refers to element in instance.
 - b. ECHO checks that the metadata XPath is valid.

8.3.1.2 ECHO Forms is Document Sent to Client

When a client requests an ECHO Forms document that contains autopopulate expressions ECHO will do nothing differently than the normal case. The form will be sent containing the original autopopulate expressions.

8.3.1.3 Client Sends the Selection XML to ECHO

When a client sends a selection to ECHO, ECHO will validate it in the usual manner. There is no special validation of selections for forms with autopopulate expressions.

8.3.1.4 ECHO Sends the Selection XML to the Provider

Before ECHO sends an option selection XML document to the provider for a form with autopopulate expressions, ECHO will perform the following steps:

1. ECHO loads the form associated with the selection XML.
2. For each autopopulate expression in the form
 - a. ECHO evaluates the ref XPath against the selection XML.
 - b. If the XPath resolved to an element then ECHO will replace the contents of the element with any results by resolving the metadata XPath against the metadata of the granule or collection.
 - 1) If nothing was returned by metadata XPath then the contents of element will be empty
 - a) XPath: /result/notinxml
 - b) Element: <granulesize></granulesize>
 - 2) If a string was returned then the string will be the contents of the element from the selection XML
 - a) XPath: /results/provider/result/GranuleURMetaData/DataGranule/SizeMBDataGranule/text()
 - b) Element: <granulesize>142.7129</granulesize>
 - 3) If one or more elements were returned then they will become the contents of the element in the selection XML.
 - a) XPath: /results/provider/result/GranuleURMetaData/DataGranule/SizeMBDataGranule
 - b) Element: <granulesize> <SizeMBDataGranule>142.7129</SizeMBDataGranule> </granulesize>

8.3.2 Client Processing Model

Clients are not required to do any processing of autopopulate expressions. As with the normal ECHO Forms handling, a client should send all elements in the model that were not referenced by controls back to ECHO in the selection.

8.4. ECHO Form Rules and Guidelines

This section describes some rules and guidelines for using the Autopopulate extension in ECHO Forms. MUST means it is a requirement. SHOULD means it is a guideline.

1. The ref XPath MUST evaluate to an element in the instance. They can not be used with XML attributes.
 - a. Example: /prov:options/prov:granulesize is good. /prov:options/@prov:granulesize is not allowed.
 - b. The reason for this is that the contents of the node may contain elements from the metadata and only XML elements may contain other XML elements.
2. The metadata XPath MUST be a valid XPath.
 - a. ECHO can not ensure that the XPath into the metadata is semantically correct. This is the form's author's responsibility.
3. Form controls SHOULD NOT reference the same element as an autopopulate expression.

- a. In the ECHO Processing Model for autopopulate expressions the contents of the element referred to by the expression will be replaced with the values from the metadata. If a control referenced the same element then any user inputted values would be lost when ECHO processed the form.

9. APPENDIX

9.1. Validation Reference Table

This table summarizes how validation is performed on form selections with relation to the required and relevant states. This information is defined in detail in the specification sections; however this table summarizes the relationships. If a cell in the table does not have a value, then the value is not important to the outcome.

Each of the result types causes a different behavior in the form processing:

1. Error: causes ECHO to throw an exception indicting that the form selection is invalid
2. Ignore: causes ECHO to ignore the control and not perform further validation on the selection for that control
3. Validate: causes ECHO to perform further validation on the control selection including type specific and bound checking.

The columns of the table indicate various states for a control at the time of validation. The columns are defined as:

1. Element present: the control has a ref tag and an element exists in the selection
2. Has a value: the element located by the ref tag contains a value for the control. For most typed controls, this is true if the element contains text with a length greater than zero after trimming. For select controls, this is true if the element contains a value child element (ignoring the contents of the value child element)
3. Supports a value: the control type supports a value, such as inputs or selects
4. Required: the value of the required attribute on the control
5. Relevant: the value of the relevant attribute on the control
6. Combinations: the number of state combinations that can produce the row result. This is determined by the number of insignificant fields for the given row

Result	Element Present	Has a Value	Supports a Value	Required	Relevant	Combinations
Error	T				F	8
Error	F			T	T	4
Ignore	F				F	8
Validate	T	T	T		T	2
Error	T	F	T	T	T	1
Validate	T		F		T	4
Ignore	F			F	T	4

Ignore	T	F	T	F	T	1
---------------	---	---	---	---	---	---