Configuring the XML file for the sample external data service

The sample external data service is deployed with an XML file that defines what case types and properties the sample data service manages. You can use the XML file to control the external data service.

XML file structure

The following sample XML file defines an external system that provides information for five properties of a single case type:

- PropOne: An integer property with common static information such as minimum and maximum values.
- State: A string property that controls the information of the City property. State has a choice list with the possible states that can be selected.
- City: A string property that changes depending on the value of the State property. The City property has a different choice list for each value of the State property.
- MVInt: A multi-value integer property that controls the information of the MVString property.
- MVString: A multi-value string property that changes depending on the value of MVInt. The scenario for dynamically modifying the property configurations based on multi-value properties is provided for example purposes only.

```
<?xml version="1.0" encoding="UTF-8"?>
<ExternalDataSystem>
    <Properties>
        <!-- Static info -->
        <Property identifier="P1">
            <SymbolicName>SOL_PropOne</SymbolicName>

            <!-- integer/string/id/boolean/float/datetime -->
            <PropertyType>integer</PropertyType>

            <!-- single/multi -->
            <Cardinality>single</Cardinality>

            <!-- optional. Indicates to always return this value
                 in the payload with a display mode of readonly.
              -->
            <!-- RenderedReadOnlyValue -->

            <!-- optional -->
            <MaximumValue>10</MaximumValue>

            <!-- optional -->
    <MinimumValue>1</MinimumValue>

    <!-- optional -->
    <!-- MaximumLength -->

    <!-- optional, the Required element -->
    <!-- Required -->

    <HasDependentProperties>false</HasDependentProperties>

    <!-- optional -->
    <!-- ChoiceList -->

    <!-- optional. The value to use when fetching the
         Properties the first time for a new object.
      -->
    <!-- ValueIfNew -->

    <!-- optional. The value or handling when
         the current value is invalid due to other constraints.
      -->
    <ValueIfInvalid handling="returnMessage" />

    <!-- optional. Specifies a value both for the initial
         Payload of a new object and when the current value is
         invalid.
      -->
    <!-- ValueIfNewOrInvalid/ -->
</Property>
<Property identifier="St">
    <SymbolicName>SOL_State</SymbolicName>
    <PropertyType>string</PropertyType>
    <Cardinality>single</Cardinality>
    <MaximumLength>2</MaximumLength>
    <Required>true</Required>
    <HasDependentProperties>true</HasDependentProperties>
    <ChoiceList>
        <DisplayName>StateChoiceList</DisplayName>
        <Choices>
            <Choice>
                <Name>New York</Name>
                <Value>NY</Value>
            </Choice>
            <Choice>
                <Name>California</Name>
                <Value>CA</Value>
            </Choice>
            <Choice>
                <Name>Nevada</Name>
                <Value>NV</Value>
            </Choice>
        </Choices>
    </ChoiceList>
</Property>

<!-- Dynamic info -->
<!-- City, when State=NY -->
```

```
<Property identifier="Ct@St=NY">
    <SymbolicName>SOL_City</SymbolicName>
    <PropertyType>string</PropertyType>
    <Cardinality>single</Cardinality>
    <Hidden>false</Hidden>
    <Required>true</Required>
    <HasDependentProperties>false</HasDependentProperties>
    <ChoiceList>
        <DisplayName>CityChoiceList</DisplayName>
        <Choices>
            <Choice>
                <Name>Buffalo</Name>
                <Value>Buffalo</Value>
            </Choice>
            <Choice>
                <Name>New York</Name>
                <Value>New York</Value>
            </Choice>
            <Choice>
                <Name>Rochester</Name>
                <Value>Rochester</Value>
            </Choice>
        </Choices>
    </ChoiceList>
    <ValueIfNewOrInvalid>New York</ValueIfNewOrInvalid>
</Property>

<!-- City, when State=CA -->
<Property identifier="Ct@St=CA">
    <SymbolicName>SOL_City</SymbolicName>
    <PropertyType>string</PropertyType>
    <Cardinality>single</Cardinality>
    <Hidden>false</Hidden>
    <Required>true</Required>
    <HasDependentProperties>false</HasDependentProperties>
    <ChoiceList>
        <DisplayName>CityChoiceList</DisplayName>
        <Choices>
            <Choice>
                <Name>Los Angeles</Name>
                <Value>Los Angeles</Value>
            </Choice>
            <Choice>
                <Name>San Diego</Name>
                <Value>San Diego</Value>
            </Choice>
            <Choice>
                <Name>San Francisco</Name>
                <Value>San Francisco</Value>
            </Choice>
        </Choices>
    </ChoiceList>
    <ValueIfNew>Los Angeles</ValueIfNew>
    <ValueIfInvalid handling="returnMessage" />
</Property>

<!-- City, when State=NV -->
```

```
<Property identifier="Ct@St=NV">
    <SymbolicName>SOL_City</SymbolicName>
    <PropertyType>string</PropertyType>
    <Cardinality>single</Cardinality>
    <Hidden>false</Hidden>
    <Required>true</Required>
    <HasDependentProperties>false</HasDependentProperties>
    <ChoiceList>
        <DisplayName>CityChoiceList</DisplayName>
        <Choices>
            <Choice>
                <Name>Las Vegas</Name>
                <Value>Las Vegas</Value>
            </Choice>
            <Choice>
                <Name>Reno</Name>
                <Value>Reno</Value>
            </Choice>
        </Choices>
    </ChoiceList>
    <ValueIfNew>Las Vegas</ValueIfNew>
    <ValueIfInvalid handling="replaceValue" />
</Property>

<!-- MVString when MVInt = [0,100] -->
<Property identifier="MVs@mvi=0_100">
    <SymbolicName>SOL_MVString</SymbolicName>
    <PropertyType>string</PropertyType>
    <Cardinality>multi</Cardinality>
    <MaximumLength>24</MaximumLength>
    <Required>true</Required>
    <HasDependentProperties>false</HasDependentProperties>
    <ChoiceList>
        <DisplayName>MVStringChoiceList</DisplayName>
        <Choices>
            <Choice>
                <Name>One</Name>
                <Value>One</Value>
            </Choice>
            <Choice>
                <Name>Two</Name>
                <Value>Two</Value>
            </Choice>
            <Choice>
                <Name>Three</Name>
                <Value>Three</Value>
            </Choice>
            <Choice>
                <Name>Ten</Name>
                <Value>Ten</Value>
            </Choice>
            <Choice>
                <Name>Eleven</Name>
                <Value>Eleven</Value>
            </Choice>
            <Choice>
                <Name>Twelve</Name>
                <Value>Twelve</Value>
            </Choice>
        </Choices>
    </ChoiceList>
    <ValueIfInvalid handling="replaceValue" />
</Property>

<!-- MVString when MVInt includes 200 -->
<Property identifier="MVs@mvi_inc_200">
    <SymbolicName>SOL_MVString</SymbolicName>
    <PropertyType>string</PropertyType>
    <Cardinality>multi</Cardinality>
    <MaximumLength>24</MaximumLength>
    <Required>true</Required>
    <HasDependentProperties>false</HasDependentProperties>
    <ChoiceList>
        <DisplayName>MVStringChoiceList</DisplayName>
        <Choices>
            <Choice>
                <Name>One</Name>
                <Value>One</Value>
            </Choice>
            <Choice>
                <Name>Two</Name>
                <Value>Two</Value>
            </Choice>
            <Choice>
                <Name>Three</Name>
                <Value>Three</Value>
            </Choice>
            <Choice>
                <Name>Twenty</Name>
                <Value>Twenty</Value>
            </Choice>
            <Choice>
                <Name>Thirty</Name>
                <Value>Thirty</Value>
            </Choice>
            <Choice>
                <Name>Forty</Name>
                <Value>Forty</Value>
            </Choice>
        </Choices>
    </ChoiceList>
    <ValueIfInvalid handling="returnMessage" />
</Property>

<!-- MVString when MVInt includes 300 -->
<Property identifier="MVs@mvi_inc_300">
    <SymbolicName>SOL_MVString</SymbolicName>
    <PropertyType>string</PropertyType>
    <Cardinality>multi</Cardinality>
    <MaximumLength>24</MaximumLength>
    <Required>true</Required>
    <HasDependentProperties>false</HasDependentProperties>
    <ChoiceList>
        <DisplayName>MVStringChoiceList</DisplayName>
```

```
            <Choices>
                <Choice>
                    <Name>One</Name>
                    <Value>One</Value>
                </Choice>
                <Choice>
                    <Name>Two</Name>
                    <Value>Two</Value>
                </Choice>
                <Choice>
                    <Name>Three</Name>
                    <Value>Three</Value>
                </Choice>
                <Choice>
                    <Name>Fifty</Name>
                    <Value>Fifty</Value>
                </Choice>
                <Choice>
                    <Name>Sixty</Name>
                    <Value>Sixty</Value>
                </Choice>
                <Choice>
                    <Name>Seventy</Name>
                    <Value>Seventy</Value>
                </Choice>
            </Choices>
        </ChoiceList>
        <ValueIfInvalid
           handling="returnMessageNoIndividualItems" />
    </Property>

</Properties>

<PropertySets>
    <!-- Set to include MVString property when MVInt = [0,100] -->
    <PropertySet identifier="Pset_MVs@mvi=0_100">
        <StaticProperties>
            <PropertyRef identifier="MVs@mvi=0_100" />
        </StaticProperties>
    </PropertySet>

    <!-- Set to include MVString property when MVInt
         includes 200 -->
    <PropertySet identifier="Pset_MVs@mvi_inc_200">
        <StaticProperties>
            <PropertyRef identifier="MVs@mvi_inc_200" />
        </StaticProperties>
    </PropertySet>

    <!-- Set to include MVString property when MVInt
         includes 300 -->
    <PropertySet identifier="Pset_MVs@mvi_inc_300">
        <StaticProperties>
            <PropertyRef identifier="MVs@mvi_inc_300" />
        </StaticProperties>
    </PropertySet>
```

```
        <!-- Set to include default MVString property -->
        <PropertySet identifier="Pset_MVs_def">
            <StaticProperties>
                <!-- Just for kicks include this one as an
                     inline property -->
                <!-- MVString, default when MVInt doesn't match
                     any other conditions -->
                <Property>
                    <SymbolicName>SOL_MVString</SymbolicName>
                    <PropertyType>string</PropertyType>
                    <Cardinality>multi</Cardinality>
                    <MaximumLength>24</MaximumLength>
                    <Required>true</Required>

<HasDependentProperties>false</HasDependentProperties>
                    <ChoiceList>
                        <DisplayName>MVStringChoiceList</DisplayName>
                        <Choices>
                            <Choice>
                                <Name>One</Name>
                                <Value>One</Value>
                            </Choice>
                            <Choice>
                                <Name>Two</Name>
                                <Value>Two</Value>
                            </Choice>
                            <Choice>
                                <Name>Three</Name>
                                <Value>Three</Value>
                            </Choice>
                        </Choices>
                    </ChoiceList>
                    <ValueIfInvalid handling="replaceValue">
                        <Value>One</Value>
                        <Value>Three</Value>
                    </ValueIfInvalid>
                </Property>
            </StaticProperties>
        </PropertySet>

        <!-- The top property set -->
        <PropertySet identifier="Top">
            <StaticProperties>
                <!-- PropOne -->
                <PropertyRef identifier="P1" />
                <!-- State -->
                <PropertyRef identifier="St" />
                <!-- MVInt -->
                <!-- Just for kicks, define this one inline -->
                <Property>
                    <SymbolicName>SOL_MVInt</SymbolicName>
                    <PropertyType>integer</PropertyType>
                    <Cardinality>multi</Cardinality>
                    <MaximumValue>1000</MaximumValue>
                    <MinimumValue>0</MinimumValue>

<HasDependentProperties>true</HasDependentProperties>
```

```
                    <ValueIfNewOrInvalid>
                        <Value>0</Value>
                        <Value>100</Value>
                    </ValueIfNewOrInvalid>
                </Property>
            </StaticProperties>

            <DynamicPropertySets>
                <DynamicPropertySet>

<ConditionalPropertyName>SOL_State</ConditionalPropertyName>
                    <ConditionalPropertySets>
                        <ConditionalPropertySet>
                            <!-- One of: -->
                            <!-- Equals, can be single or
                                 multi-value -->
                            <Equals>NY</Equals>

                            <!-- Or, Between. Only for single
                                 value. -->
                            <!-- Between>
                                <LowerValue>some_value1</LowerValue>
                                <UpperValue>some_value2</UpperValue>
                            < / Between -->

                            <!-- Or, Includes. Only for a muti-value
                                 Prop but specifies a single value
                                 to test against. -->
                            <!-- Includes>some_value</Includes -->

                            <!-- Inline set to include city property
                                 when state=NY -->
                            <PropertySet>
                                <StaticProperties>
                                    <PropertyRef identifier="Ct@St=NY"
/>
                                </StaticProperties>
                            </PropertySet>

                        </ConditionalPropertySet>
                        <ConditionalPropertySet>
                            <Equals>CA</Equals>
                            <!-- Inline set to include city property
                                 when state=CA -->
                            <PropertySet>
                                <StaticProperties>
                                    <PropertyRef identifier="Ct@St=CA"
/>
                                </StaticProperties>
                            </PropertySet>
                        </ConditionalPropertySet>
                        <ConditionalPropertySet>
                            <Equals>NV</Equals>
                            <!-- Inline set to include city property
                                 when state=NV -->
                            <PropertySet>
                                <StaticProperties>
```

```
                                    <PropertyRef identifier="Ct@St=NV"
/>
                                </StaticProperties>
                            </PropertySet>
                        </ConditionalPropertySet>
                    </ConditionalPropertySets>
                    <!-- Set to include default city property -->
                    <DefaultPropertySet>
                        <StaticProperties>
                            <!-- Just for kicks, define this property
                                 inline.
                                 City, default info if no other
                                 constraint match. Rendered read-only
                                 and hidden. -->
                            <Property>
                                <SymbolicName>SOL_City</SymbolicName>
                                <PropertyType>string</PropertyType>
                                <Cardinality>single</Cardinality>
                                <RenderedReadOnlyValue/>
                                <Hidden>true</Hidden>
                                <Required>true</Required>

<HasDependentProperties>false</HasDependentProperties>
                            </Property>
                        </StaticProperties>
                    </DefaultPropertySet>
                </DynamicPropertySet>
                <DynamicPropertySet>

<ConditionalPropertyName>SOL_MVInt</ConditionalPropertyName>
                    <ConditionalPropertySets>
                        <ConditionalPropertySet>
                            <Equals>
                                <Value>0</Value>
                                <Value>100</Value>
                            </Equals>

                            <PropertySetRef
                             identifier="Pset_MVs@mvi=0_100" />
                        </ConditionalPropertySet>
                        <ConditionalPropertySet>
                            <Includes>200</Includes>
                            <PropertySetRef
                               identifier="Pset_MVs@mvi_inc_200" />
                        </ConditionalPropertySet>
                        <ConditionalPropertySet>
                            <Includes>300</Includes>
                            <PropertySetRef
                              identifier="Pset_MVs@mvi_inc_300" />
                        </ConditionalPropertySet>
                    </ConditionalPropertySets>
                    <DefaultPropertySetRef identifier="Pset_MVs_def" />
                </DynamicPropertySet>
            </DynamicPropertySets>
        </PropertySet>
    </PropertySets>
```

```
<CaseTypes>
    <CaseType>
        <Name>SOL_MyCase</Name>
        <PropertySetRef identifier="Top" />
    </CaseType>
</CaseTypes></ExternalDataSystem>
```

The following list shows the structure of the information that the XML file configures:

- The external data system
    - A case type
        - The property set for this case type.
            - Each property set can include static information for one or more properties.
            - Each property set can also include dynamic property sets
                - A dynamic set is based on the working value of a single property.
                - A condition, such as “equals” or “between”, matches the property working value.
                    - If the condition matches, a property set is used that can be a further nested property set with static information or dynamic property sets of its own or both.
                - More conditions can exist.
                - The default property set to use if none of the conditions match.
                    - Can be a further nested property set with static information or dynamic property sets of its own or both.
            - More dynamic property sets can exist.
    - More case types can exist.

Property values in XML elements

Various XML elements have property values. The values must match the format appropriate for the type of property, for example:

- Integer: 123; 589432; -45; 0
- Floating point: 123.56; 45.0; 0.0
- ID: {guid}
- String: Any valid string
- Boolean: true; false
- Date-time, W3C format in the GMT time zone: 2011-12-31T23:59:59Z
- A null value is indicated by an XML element with no content, for example, <RenderedReadOnlyValue/>

Several elements can accept multiple values when the property that is being configured has a cardinality of “multi.” Multiple values are indicated by multiple <value> subelements. The content of each <value> element is of the format described above, appropriate for the type of property. A multi-value property cannot have a null value for any of its individual elements. For example, for a multi-value integer property, a list of values will look something like this:

```
<RenderedReadOnlyValue>
    <Value>10</Value>
    <Value>20</Value>
    <Value>30</Value>
</RenderedReadOnlyValue>
```

If the top-level element is empty or has no content, the list is considered an empty list, for example:

```
<RenderedReadOnlyValue/>
```

Properties and property elements

A Property element defines the information for a single property. A Property element can be enclosed inside the top-level Properties element with an identifier so that it can be referenced from one or more places elsewhere in the XML file. Alternatively, if the property information is used in only one place, the Property element can be defined inline where it is used. The previous sample XML illustrates both approaches.

The same property, which corresponds to a specific symbolic name, can be represented by multiple Property elements because the information for that property can be dynamically determined based on other property values. Therefore, a different Property element can be selected based on specific dynamic matching criteria.

Each Property element can have the identifier attribute. Follow these guidelines when you use the identifier attribute:

- The identifier attribute must be included if the Property element is a stand-alone element inside the Properties element.
- The identifier is used to reference this particular Property element from other XML elements.
- The attribute value can be any textual identifier, and it must be unique among all other Property elements.
- The identifier attribute cannot be included if the Property element is included inline in some other part of the XML file. In that case, the element cannot be referenced in more than one place.

A Property element has the following subelements that are used for processing the value of a property in the payload data. External data services can specify the working value of a property in the response payload if they determine another value is appropriate.

- RenderedReadOnlyValue. An optional element.

  - External data services can specify that a property should be rendered read-only, which effectively fixes the value to what the external service specifies. This means the payload will include a Value attribute that matches this value and a DisplayMode=readonly attribute.
  - If this element is specified, the working value of the property will be modified if necessary to match this value.
  - The content of this element is a property value in the format appropriate for its type or an element with no content to specify the null value.
  - If the property is a multi-value property, no or empty content indicates an empty list. Specify multiple items with multiple <Value> subelements:
    ```
        <RenderedReadOnlyValue>
            <Value>10</Value>
            <Value>20</Value>
            <Value>30</Value>
        </RenderedReadOnlyValue>
    ```
  - The RenderedReadOnlyValue element takes precedence over all other value handling elements.
- ValueIfNew. Represents the value to use when information is fetched for a new case.
  - When information for a new case is being fetched, this element causes a new value to be specified if necessary. A new value is specified if the default value of the case type, which is normally used as the initial value for a new case, does not match the value of this element.
  - This element applies to a single or multiple-value property.
  - If the element has no content, it indicates a null initial value (or an empty multi-value list). This is typically not intended because you typically want to override only the normal default value if that value is determined to be invalid. A missing ValueIfNew element combined with ValueIfInvalid accomplishes the desired behavior. So specify a null (or empty list) value only if you want to force a null value to always be used for a new case.
  - For a multiple cardinality property, specify multiple items with multiple <Value> subelements:
    ```
      <ValueIfNew>
          <Value>10</Value>
          <Value>20</Value>
          <Value>30</Value>
      </ValueIfNew>
    ```
  - The ValueIfNew element takes precedence over the other elements used for handling an invalid value when the initial information is being fetched for a new case.
- ValueIfInvalid. Represents how the external data system will process the value when the current value is determined to be invalid based on other constraints, for example, minimum and maximum values, choice lists, or maximum length).
  - The element has the following attribute:
    - handling. This is an optional attribute that indicates the type of handling:

        - returnMessage. This is the default value. If the value is determined to be invalid, the value is left unchanged and a custom validation error message is returned in the payload for this property. If the property is a multi-value property, the payload also includes the list of individual items that are invalid.
        - returnMessageNoIndividualItems. This is similar to the returnMessage option except that if the property is a multi-value property, the list of individual items is not returned in the payload.
        - replaceValue. If the value is determined to be invalid, the value is replaced with the value indicated by the content of this element.
- When the external data service processes the current properties, it will check the current value for validity based on the property cardinality:
    - The value of a single-value property or each item of a multi-value property are checked against the following constraints if applicable: MaximumValue; MinimumValue; MaximumLength; ChoiceList;
    - If the property is also configured with <Required>true</Required>, then a null single cardinality value or empty multiple cardinality value is also considered invalid, but whether any action is performed in this situation depends on the value of the handling attribute.
- If “replaceValue” is the value of the “handling” attribute:
    - For a single cardinality property, if the current value is invalid, it will be modified to match this element’s value. The value will be modified to null if this element has no content.
    - For a multiple cardinality property, each item of the current multi-item value will be checked for validity. Any invalid items are removed from the list. Only if all items are removed, the entire list of values will be modified to match this element’s list of values. If this element has no content, the value will be modified to an empty list.
    - If the property is configured with <Required>true</Required>, a null single cardinality value or empty multiple cardinality value will also be modified to match this element’s value. If the element has no content, the value will be left as null or empty.
    - For a multiple cardinality property, specify multiple items with multiple <Value> subelements:
      ```
      <ValueIfInvalid>
          <Value>10</Value>
          <Value>20</Value>
          <Value>30</Value>
      </ValueIfInvalid>
      ```

  - If "returnMessage" is the value of the handling attribute, the current value is checked for validity as described above, but the value is not replaced. Instead, an appropriate error message is returned in the payload for this property. If the property is a multi-value property, the payload also includes the list of individual items that are invalid.
    - An error message is not returned if the property is configured with <Required>true</Required> and the value is null or empty.
  - If "returnMessageNoIndividualItems" is the value of the handling attribute, an appropriate error message is returned as for the "returnMessage" value. However if the property is a multi-value property, the list of individual items is not returned. This sets up a test case where the validation error exists for the property as a whole but not specific items.
- ValueIfNewOrInvalid. Represents the value to use when information is being fetched for a new case or if the current working value of a property is invalid. This element is equivalent to using the following two elements:
  ```
  <ValueIfNew>some_value</ValueIfNew>
  <ValueIfInvalid handling="replaceValue">
          some_value</ValueIfInvalid>
  ```
  The same value is specified for both elements.

Each Property element can have the following other subelements:

- SymbolicName. This is a required element whose content is the symbolic name of a property in Content Engine.
- PropertyType. This is a required element whose content is integer, string, id, boolean, float, or datetime. The element must match the type of the property in Content Engine.
- Cardinality. A required element whose content is single or multi. The element must match the cardinality of the property in Content Engine.
- RequiresUniqueElements. This is an optional element whose content is true or false. The default is false. It is only applicable if the cardinality is multi. This option should typically not be used. This configurable data system includes some support for Content Engine multi-value properties whose RequiresUniqueElements element is true, which means each item in the list is unique and the list is unordered. The main difference is in comparing values. For RequiresUniqueElements=false, items in the list must be in the same order for two lists to be considered equal, but for RequiresUniqueElements=true, order does not matter. However, other case management software currently does not support unordered or unique multi-value properties so that option should not be used.
- Hidden. An optional element. Specifies whether the property is hidden in the user interface.
  - Whatever value is specified in Content Engine can be overridden by this element. If the element is not specified, the underlying Content Engine value is unaffected.

- MaximumValue. An optional element. If this element is specified, an appropriate value for the type of property is integer, floating point, or date-time.
    - If this element is specified, the value cannot be larger than an underlying Content Engine value.
- Minimum Value. An optional element. If this element is specified, an appropriate value for the type of property is integer, floating point, date-time.
    - If this element is specified, it cannot be smaller than an underlying Content Engine value.
- MaximumLength. An optional element that can be specified for a string property.
    - If this element is specified, it cannot be longer than an underlying Content Engine value.
- Required. An optional element that is true or false. The only reason to specify this element is to override a property that is not required in Content Engine as true.
    - You cannot specify false if the underlying required value in Content Engine is true.
- HasDependentProperties. The content of the element must be true or false. Setting this element to true triggers a POST method to be called to get updated information whenever the user edits the value.
- ChoiceList. An optional element. This is valid for a string or integer property. If this element is specified, it contains the following subelements:
    - DisplayName. The display name of the choice list, but it will probably not be shown in an interface.
    - Choices element. Contains Choice subelements for each choice. Each Choice element contains the following subelements:
        - Name. The display label for this choice.
        - Value. The value of this choice. The content is formatted as appropriate for the type of property, such as string or integer.

Property sets

A PropertySet element defines some collection of specific property configurations. A PropertySet element can be enclosed inside the top-level PropertySets element with an identifier so it can be referenced from one or more places elsewhere in the XML. Alternatively, if the property set information is only used in one place, the PropertySet element can be defined inline where it is used. The sample XML above illustrates both approaches.

Each PropertySet element has the following attribute and subelements:

- identifier attribute.
    - Like the attribute for the Property element, it is a unique identifier used to reference this Property Set.
    - The attribute must be included if the PropertySet element is a stand-alone element inside of the PropertySets element.

    - The attribute value can be any textual identifier. It must be unique among all of the other PropertySet elements.
    - The identifier attribute cannot be included if the PropertySet element is included inline in some other part of the XML. In that case, the element cannot be referenced in more than one place.
- StaticProperties. An optional element. Contains one or more subelements that represent the static property information of this property set.
    - PropertyRef. Has one attribute, identifier, that matches the identifier of a <Property> element. That property information is included as part of the static information.
    - Property. A Property element can also be included inline if the property information is only used in this one place.
- DynamicPropertySets. An optional element. Holds one or more DynamicPropertySet subelements. Each DynamicPropertySet element has the following subelements:
    - ConditionalPropertyName. Holds the symbolic name of a property that the matching criteria for this dynamic set are based on.
    - ConditionalPropertySets. Contains one or more ConditionalPropertySet subelements. Each ConditionalPropertySet element has the following subelements:
        - One criteria element can be specified:
            - Equals. Indicates a criteria match if the current working property value matches this value. The content is the value formatted as appropriate for the property type.
                - If the property is a multi-value property, no or empty content indicates an empty list. Specify multiple items with multiple <Value> subelements:

```
<Equals>
    <Value>10</Value>
    <Value>20</Value>
    <Value>30</Value>
</Equals>
```

            - Between. This is only applicable to a single cardinality property. Indicates a criteria match if the current working property value is between 2 values. This is an inclusive between. Contains the following 2 subelements:
                - LowerValue. The lower range of values to compare against.
                - UpperValue. The upper range of values to compare against.
            - Includes. This is only applicable to a multiple cardinality property. This element always specifies a single value and indicates a criteria match if that value is included in the list of the current working property value.
        - PropertySetRef. Has an “identifier” attribute that is the identifier of a PropertySet element. Includes that property set when the

        specified criteria matches. That property set can have just a static set of property information or can have further dynamic property sets.
        - PropertySet. As an alternative to PropertySetRef, if the property set is only used in this one place, an inline PropertySet element can be specified instead.
    - DefaultPropertySetRef. The default property set to use if none of the conditional property sets match. Has an “identifier” attribute that is the identifier of a PropertySet element. Includes that property set as the default set. That property set can have just a static set of property information or can have further dynamic property sets.
    - DefaultPropertySet. As an alternative to DefaultPropertySetRef, if the property set is only used in this one place, an inline DefaultPropertySet element can be specified instead. The content of the DefaultPropertySet element is the same as a PropertySet element.

Case types

The CaseTypes element holds one or more CaseType subelements. Each CaseType element has the following subelements:

- Name. The symbolic name of the case folder class.
- PropertySetRef. Has an “identifier” attribute that is the identifier of a PropertySet element. This is the property set for this case type. It can have a static set of property information as well as dynamic property sets with further nested property sets with static and/or dynamic information, etc.
- PropertySet. As an alternative to PropertySetRef, if the property set is only used in this one place, an inline PropertySet element can be specified instead.

Restrictions and constraints

- All stand-alone Property elements inside the Properties element must have unique identifier attributes.
- All stand-alone PropertySet elements at the top-level of the PropertySets element must have unique identifier attributes.
- Any inline Property element in a location other than the Properties element cannot have an identifier attribute.
- Any inline PropertySet element in a location other than the top level of the PropertySets element cannot have an identifier attribute.
- At least one of StaticProperties or DynamicPropertySets elements must be specified in a PropertySet element.
- Only backward references to Property elements and PropertySet elements are allowed in the XML.

- The properties represented in the StaticProperties element of a PropertySet element must not overlap with the properties that are represented by the DynamicPropertySets element.
    - The properties that are represented by the DynamicPropertySets element are determined by evaluating each DynamicPropertySet element and each PropertySet element, which are referenced in each ConditionalPropertySet and DefaultPropertySetRef element. The properties that are represented include any static properties that those PropertySet elements reference and possibly any nested DynamicPropertySet elements.
- Inside the DynamicPropertySets element, the properties that are represented by each DynamicPropertySet element must not overlap with any other DynamicPropertySet element. This means that each DynamicPropertySet represents a set of properties that is isolated from other DynamicPropertySet properties and the StaticProperties.
- Inside an individual DynamicPropertySet element, each of the ConditionalPropertySet elements and the DefaultPropertySetRef element must represent the same set of properties. This means that each of the ConditionalPropertySet and DefaultPropertySetRef elements must represent a stable set of properties no matter what input property values are submitted.