Proposal: Runtime Properties

Describes the feature as-is-built 2021-09-09

Motivation

The Daffodil Java and Scala API's require the use of user allocated classes that are passed into the Daffodil parse/unparse functions. Examples of such objects include implementations of the `InputSourceDataInputStream` , `InfosetInputter` , and `InfosetOutputter` interfaces. These interfaces are designed so that users can implement custom inputs and outputs that require no knowledge from Daffodil aside from the interface functions. For example, a user could implement a custom `InfosetInputter` and `InfosetOutputter` to support EXI or YAML infoset representations, which Daffodil does not natively support, without Daffodil needing to know the implementation details.

It may sometimes be desirable for these custom implementations to have configurable behavior based on annotations applied to the DFDL schema. Some use cases include:

- Define "dirty words" on the schema for particular elements which should be redacted/removed when output from, or input to, an infoset
- Define a basic transformation to be applied on some infoset elements, e.g. uppercase/lowercase
- Define a complex transformation, such as converting a simple element string to actual XML nodes (i.e. non-escaped XML)

In each of these cases, it is not Daffodil that performs such redactions/transformations, but the `InfosetInputter` and/or `InfosetOutputter` . But such classes are not necessarily schema aware, so likely have no information about which elements must be transformed or how to perform those transformations.

The following proposal suggests a way to add generic annotations to a DFDL schema which are then passed into the `InfosetInputter` and `InfosetOutputter` , allowing custom input/output behavior to be specified on the DFDL schema.

Implementation

A new extension property is added called `dfdlx:runtimeProperties`. The value of this property is a space-separated list of key/value pairs, with each keys and pairs separated by an equals sign (=). For example:

```
<xs:element name="xs:string" dfdlx:runtimeProperties="key1=value1 key2=value2" ... />
```

Because the use-cases only include transforming the infoset text, this property is valid only on simple types.

At schema compilation time, this key/value pairs are parsed and converted to a Map. If a duplicate key is found in this list, the previous is discarded. If `dfdlx:runtimeProperties` is not defined, then an empty Map is used.

The Map, whether empty or not, is added as a new member of the `ElementRuntimeData` class called `runtimeProperties`. Because this property is valid on on simple elements, this member for complex `ElementRuntimeData` instances is always the empty Map.

Parse

When an `InfosetOutputter` is supposed to output a simple element, Daffodil calls the `startSimple` method, passing in the `DISimple`. For `InfosetOutputter` implementations that wish to alter how the simple text is output, the `runtimeProperties` map can be accesses from the `erd` member of the `DISimple` parameter. For example using the Scala API:

```
class MyInfosetOutputter extends InfosetOutputter {
  ...
  override def startSimple(simple: DISimple): Boolean = {
    val runtimeProperties = simple.erd.runtimeProperties
    val key1Value = runtimeProperties.getOrDefault("key1", "defaultValue1")
    val key2Value = runtimeProperties.getOrDefault("key2", "defaultValue2")
    val simpleText = simple.dataValueAsString
    // redact or trasform simpleText based on key1/key2 values, and then output simpleText
    ...
  }
  ...
}
```

Unparse

When Daffodil needs the simple text of a simple element during unparse, it calls the `getSimpleText` method on the `InfosetInputter`. A new function is added to the `InfosetInputter` API, with the same `getSimpleText` name as the existing function, but it takes two parameters. The first is the `NodeInfo.Kind`, like the existing `getSimpleText` function. The second parameter is the runtime properties `Map` . To allow for backwards compatibility, a defult implementation of this function is added which calls the existing `getSimpleText` function with a single argument:

```
abstract class InfosetInputter ... {
  def getSimpleText(primNode: NodeInfo.Kind, runtimeProperties: Map[String,String]): String = {
    getSimpleText(primNode)
  }
}
```

For `InfosetInputter` implementations that want to redact/transform simple text before returning it to Daffodil to be unparsed, they can override this new method and use the `runtimeProperties` . For example:

```
class MyInfosetInputter extends InfosetInputter {
  override def getSimpleText(primNode: NodeInfo.Kind, runtimeProperties: Map[String,String]): String = {
    val simpleText = ... // get the simple text for this current event
    val key1Value = runtimeProperties.getOrDefault("key1", "defaultValue")
    val key2Value = runtimeProperties.getOrDefault("key2", "defaultValue")
    // redact or transform simpleText base on key1/key2 values, and return
    val transformedSimpleText = ...
    transformedSimpleText
  }
}
```

Example Implementation: stringAsXml

A primary use case for this feature is the ability to parse a simple element with an `xs:string` type expected to contain XML content. Rather than escaping the XML string and treating it like simple content, we instead want to output it as if it were part of the XML infoset. Similarly, when unparsing, we want to treat all the children of a particular infoset element as if it raw text so that it unparses as a normal string.

This was implemented in Daffodil 3.4.0 in as part as commit 3b213ce30b.