apigee

Aug, 2015

Foundational Training

Advanced Mediations

XML/JSON Differences

- XML has a root element, JSON does not need to
  - JSON can have an unnamed object or array at the top level
- JSON is typed, XML is not

```
{ "aNumber": 14, "aString": "str", "aBoolean": true, "aNull": null }
```

```
<StringOrBoolean>true</StringOrBoolean>
<StringOrNumber>14</StringOrBoolean>
<StringOrNull>null</StringOrNull>
```

- JSON null and null string are different, XML no distinction
  - "" vs. null
  - <XmlNull></XmlNull>
- XML has attributes, JSON does not
- XML has namespaces, JSON does not
- XML can span multiple lines, JSON cannot
  - JSON on multiple lines is pretty-printed JSON, don't return that from a proxy

XMLToJSON and JSONToXML

- XMLToJSON can try to guess data types, but it can be wrong
  - Configurable: RecognizeNumber, RecognizeBoolean, RecognizeNull
    - Still may guess wrong (I want ZIP codes to be strings, but they will be made numbers if RecognizeNumber is true)
  - Has to guess for arrays of objects
    - 0 objects = null (want it to be [ ])
    - 1 object = { ... } (want it to be [{ ... }]
    - 2 objects = [{ ... }, { ... }]
- JSONToXML
  - Easier direction
  - Can use JavaScript/Python to manipulate JSON before converting, or use XSL to manipulate XML after converting

JSONToXML Details

Policy Options

The options for this policy can be used to handle scenarios where JSON does not convert to valid xml or you want to customize the output of the transformation.

Example JSON: { “Some%%%%Name”, “Steve”}

Converted : <Some%%%%Name>Steve</Some%%%%Name> --Invalid XML

You can use **<InvalidCharsReplacement>_</InvalidCharsReplacement>:**

Customized Conversion: <Some_Name>Steve</Some_Name>

Additional Policy Options

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<JSONToXML async="false" continueOnError="false" enabled="true" name="JsonToXml">
    <DisplayName>JsonToXml</DisplayName>
    <Options>
        <NullValue>NULL</NullValue>
        <NamespaceBlockName>#namespaces</NamespaceBlockName>
        <DefaultNamespaceNodeName>$default</DefaultNamespaceNodeName>
        <NamespaceSeparator>:</NamespaceSeparator>
        <TextNodeName>#text</TextNodeName>
        <AttributeBlockName>#attrs</AttributeBlockName>
        <AttributePrefix>@</AttributePrefix>
        <InvalidCharsReplacement>_</InvalidCharsReplacement>
        <ObjectRootElementName>root</ObjectRootElementName>
        <ArrayRootElementName>array</ArrayRootElementName>
        <ArrayItemElementName>item</ArrayItemElementName>
    </Options>
    <OutputVariable>response</OutputVariable>
    <Source>response</Source>
</JSONToXML>
```

XMLToJSON Details

Policy Options

- **RecognizeNumber**, **RecognizeBoolean**, **RecognizeNull** - this will try to recognize the xml values and set the appropriate JSON data types
- **NullValue** - allows you to define how to represent null in the json output
- **NamespaceBlockName**, **DefaultNamespaceNodeName**, **NamespaceSeperator** - used to define Namespaces sourced in the XML
- **TextNodeName**, **TextAlwaysAsAProperty** - options that let you define how to handle mixed text and element scenarios.
- **AttributeBlockName, AttributePrefix** - allows you to define how attributes in your XML can be translated to JSON parameters
- **OutputPrefix**, **OutputSuffix** - allows you to add text to the beginning and/or end of the json output

Typical Policy Configuration

```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <XMLToJSON async="false" continueOnError="false" enabled="true" name="XmlToJson">
3     <DisplayName>XmlToJson</DisplayName>
4     <OutputVariable>response</OutputVariable>
5     <Source>response</Source>
6 </XMLToJSON>
7
```

Returning SOAP as a RESTful Response

XSL for cleaning up the SOAP payload or XPATH via Extract Variables policy?

- Use XPATH if you just need a couple of fields.
- XSLT (XSL) if you need to extract some value or entity with a dynamic or iterative value
- XSLT is generally used for complex XML payload mediations. You can also convert to JSON and then use JSONPath or manipulate the JSON object via Javascript

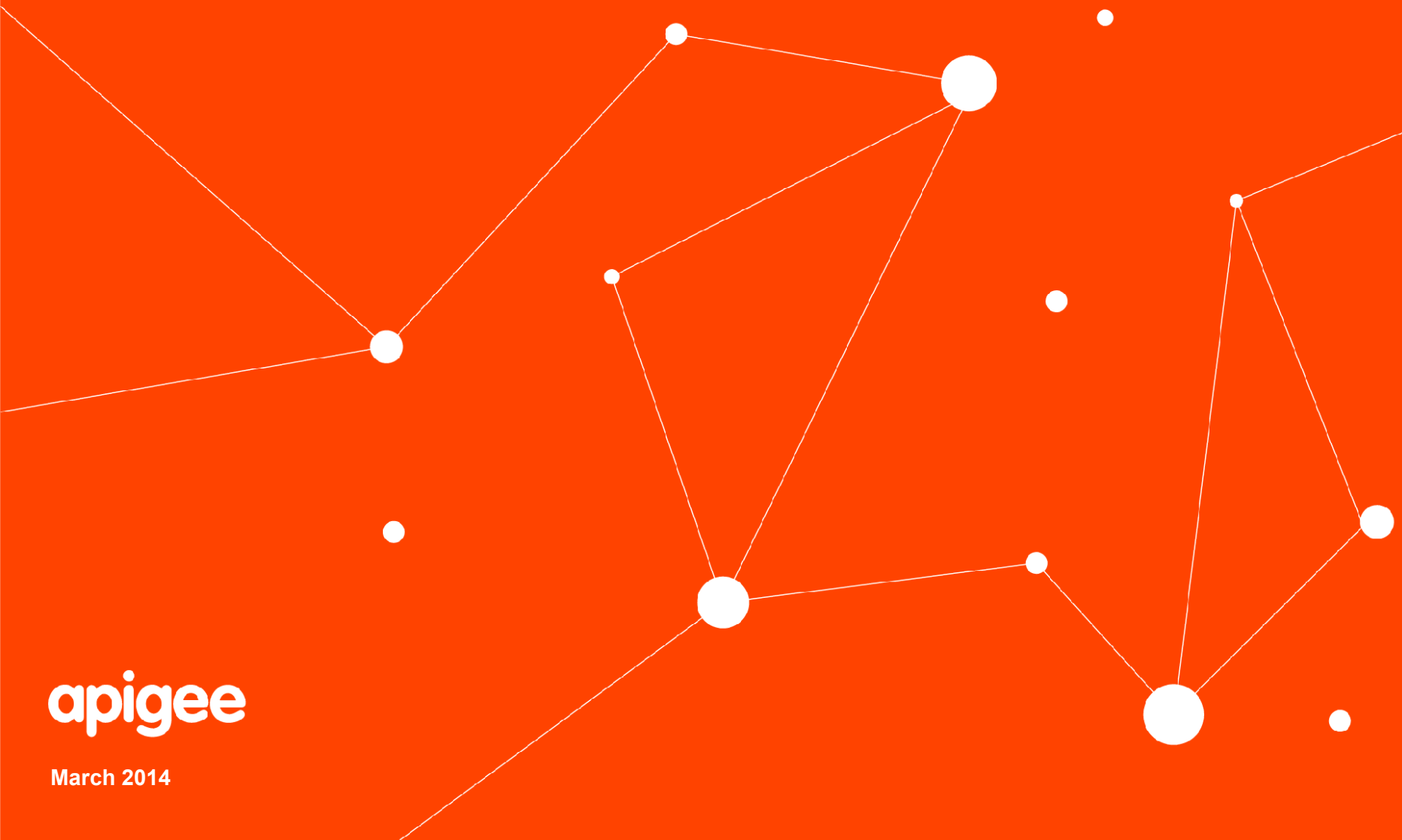
Tips and Tricks for Mediation Poilcies

- Inconsistent conversion of arrays and string fields containing numbers can be incorrectly converted by XMLToJSON and cause issues for app programmers
  - Can fix this with JavaScript using global search and replace functions
- Step 1: Set RecognizeBoolean, RecognizeNumber, RecognizeNull to true
- Step 2: Add ~STR~ to fields that should be strings but may be converted to non-strings ("~STR~" prefix forces elements to look like strings)
- Step 3: Add 2 ~ARRAY~ dummy array elements to each array (there will be a minimum of 2 elements per array, forcing conversion to JSON array instead of null or object)
- Step 4: If top-level JSON should be an array, use TOPARRAY as the XML root element
- Step 5: After XMLToJSON, clean up results with following JavaScript code:

```
// 1) replaces empty strings with null
// 2) removes remaining string specifiers
// 3) removes array placeholders, including trailing commas
// 4) removes array placeholders without trailing commas
// 5) if the return payload is meant to be an array (at the top level), return that
var jsonResponse = context.getVariable("jsonResponse").replace(/"~STR~"/,"null")
                                                      .replace(/~STR~/g,"").replace(/"~ARRAY~",/
g,"").replace(/"~ARRAY~"/g,"");
var jsonVar = JSON.parse(jsonResponse);
var jsonOut = jsonVar.ROOT;
if (jsonOut.TOPARRAY !== undefined) {
  jsonOut = jsonOut.TOPARRAY;
}
response.content = JSON.stringify(jsonOut);
```

Demo/Discussion

Thank you



apigee

March 2014